

2010



Universidad Nacional de Río Cuarto
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Departamento de Computación

Trabajo Final para optar al título de Licenciado en
Ciencias de la Computación.

GG-WSfinder:
Un Web Service que automatiza la invocación
de aplicaciones externas a un Workflow
optimizando la selección de las mismas en
base a característica de calidad.

Directora: Mg. Marcela Daniele
Co-Directora: Mg. Paola Martellotto

Autores:
Mauro Garat
Pablo Martin Gil



Agradecimientos

Queremos agradecer especialmente a nuestras directoras de Tesis Marcela Daniele y Paola Martellotto, quienes nos han conducido durante estos meses con un talento abierto y generoso, guiándonos sin ser directivas y mostrando en cada momento una inmejorable disposición ante las dudas que durante la realización del trabajo nos surgieron, aportando siempre valiosas observaciones que tutelaron esta investigación.

Además, debemos retribuir con cordial gratitud tanto a la Universidad Nacional de Río Cuarto como a la Facultad de Ciencias Exactas, Físico-Químicas y Naturales, y en especial al Departamento de Computación, por permitirnos desarrollar nuestras formaciones individuales de grado con excelente material académico, pero sobre todo humano, durante todos estos años.

Y fundamentalmente queremos agradecer el incondicional apoyo de nuestras novias, familiares y amigos, en especial durante los momentos difíciles que atravesamos al desarrollar la Tesis.

Índice de Contenidos

CAPÍTULO 1: Introducción	16
1.1 Introducción.....	17
1.2 Motivación y Propuesta.....	18
1.3 Objetivos.....	19
1.4 Estructura del documento.....	19
CAPÍTULO 2: Estado del Arte	22
2.1. Comunicación de los SGWF con las aplicaciones externas	23
2.2. Registración y búsqueda de Web Services	23
2.2.1 JUDDI Console	23
2.2.2 UDDI Browser	26
2.2.3 Woogie	28
2.3. Incorporación de las características de calidad a los Web Services.....	29
2.4. Conclusiones.....	30
CAPÍTULO 3: Workflows	33
3.1. Los Workflows	34
3.2. Workflow Management Coalition (WfMC)	35
3.2.1. Necesidad de Estandarizar	35
3.2.2. Definiciones de la WfMC	36
3.2.3. Conceptos sobre Workflows	37
3.2.4. Ventajas	39
3.3. Componentes genéricos de un SGWF	40
3.3.1 Herramienta de Definición de Procesos.....	41
3.3.2 Definición de Procesos	41
3.3.3. Workflow Enactment Service	41
3.3.4. Worklist (lista de trabajo).....	41
3.3.5. Manejador de Worklist.....	41
3.4. Invocación de Aplicaciones Externas a un SGWF según el Modelo de referencia de la WfMC	42
CAPÍTULO 4: Web Services	47
4.1. Los Web Services	48
4.2. Tipos y Arquitectura de los Web Services	53
4.2.1. Información del Negocio (Business Information).....	53
4.2.2. Transaccional (Transactional) para B2B o B2C	54

4.2.3. Externalización de procesos de negocio	54
4.2.4. Arquitectura SOA.....	55
4.3. El lenguaje WSDL	57
4.4 Características de Calidad (QoS) de los Web Services	59
4.5. Universal Description Discovery and Integration (UDDI)	61
4.5.1 UDDI: Un registro global de Web Services	61
4.5.2 WSDL y UDDI	63
CAPÍTULO 5: GG-WSfinder	66
5.1 Contexto de la herramienta	67
5.1.1. Introducción al problema	67
5.1.2. Descripción general de la herramienta GG-WSfinder.....	68
5.1.2.1. Proceso de selección de Web Services.....	69
5.2. Diseño de la herramienta	70
5.2.1. Arquitectura general	70
5.2.2. Modelo de Persistencia	76
5.2.3. Diagrama de Secuencia	79
5.3. Descripción del caso de estudio	82
5.3.1. Elección y Justificación	82
5.3.2. Plantillas de Entrada y de Salida.....	83
5.3.3. Las características de calidad en la selección de Web Services	85
5.4 Herramientas utilizadas.....	86
CAPÍTULO 6: Conclusiones y Trabajos Futuros.....	89
6.1 Conclusiones	90
6.2 Trabajos Futuros	92
Referencias Bibliográficas.....	94
ANEXO A: Tecnologías de los Workflows	100
A.1. Introducción	101
A.1.1. Workflow.....	101
A.1.2. Historia	101
A.1.3. Justificación	103
A.2. Definiciones	104
A.2.1. Procesos de Negocio (Business Process)	104
A.2.2. Workflow.....	104
A.3. Workflow y Reingeniería	105
A.3.1. Workflow como herramienta de Reingeniería	105

A.3.2. Requisitos y beneficios de la reingeniería.....	106
A.4. Workflow: Clasificación	106
A.4.1. Introducción	106
A.4.2. Workflow de Producción.....	107
A.4.3. Workflow de Colaboración	107
A.4.4. Workflow Administrativo	108
A.4.5. Workflow Ad-hoc	108
A.5. Modelando Workflow	109
A.5.1. Introducción	109
A.5.2. Conceptos.....	109
A.5.3. Arquitecturas.....	114
A.5.3.1 Componentes	114
A.5.3.2 Implementación del Workflow Enactment Service.....	116
A.5.3.3 Alternativas de aplicaciones clientes de Workflow	117
A.6. Modelo de la Workflow Management Coalition (WFMC)	118
A.6.1. Introducción	118
A.6.2. Necesidad de Estandarizar	118
A.6.3. Modelo de referencia de la WFMC.....	119
A.6.3.1. El Modelo de Workflow.....	119
A.6.3.2. Motor de Workflow (Workflow Engine).....	120
A.6.3.3. Tipos de Workflow Enactment Services.....	120
A.6.3.4. Proceso y estados de transición de las actividades	121
A.6.3.5. Workflow Application Programming Interface (WAPI)	123
A.6.3.6. Datos de: Control, Relevantes y de las Aplicaciones de Workflow	123
A.6.3.7. Intercambio de Datos	123
A.6.3.8. Definición de procesos (interface 1)	124
A.6.3.9. Interface del Workflow con aplicaciones clientes (interface 2).....	125
A.6.3.10. Aplicaciones Invocadas (interface 3).....	125
A.6.3.11. Interoperabilidad del Workflow. Workflow Enactment Services Heterogéneos.....	126
A.6.3.12. Funciones de Interoperabilidad WAPI (interface 4).....	129
A.6.3.13. Interface para la Administración y monitoreo (interface 5)	131
A.7. Estructura WAPIs, protocolos y su conformidad	132
A.7.1. WAPIs - Descripción general funcional de la API	132
A.7.2. Protocolo soporte de la WAPIs	132
A.7.3. Principios de la conformidad: ¿Qué quiere decir conformidad?	133

A.7.4. Interoperabilidad Clasificaciones y Niveles de Conformidad	133
A.7.4.1. Definición de herramientas - Incorporación de software Workflow	134
A.7.4.2. Aplicación cliente interoperando con la promulgación de servicios de Workflows	134
A.7.4.3. Herramienta de Integración y Aplicaciones	134
A.7.4.4. Interoperabilidad de servicios Workflow	134
A.7.4.5. Administración de un Workflow común	135
ANEXO B: Tecnologías de los Web Services	137
B.1. Introducción a los Web Services	138
B.1.1. ¿Qué es un Web Service?	138
B.1.2. Web Services por la W3C	140
B.1.3. Un poco de perspectiva	141
B.1.4. ¿Para qué sirven los Web Services?	142
B.1.5. Características generales de los Web Services	143
B.1.5.1. Ventajas de los Web Services	143
B.1.5.2. Inconvenientes de los Web Services	143
B.1.6. Razones para crear un Web Service	144
B.2. Conceptos Generales (Definiciones)	144
B.2.1. Protocolos, Estándares y Lenguajes utilizados por los Web Services ...	144
B.2.1.1. HTTP	144
B.2.1.2. XML	146
B.2.1.3. WSDL	147
B.2.1.4. SOAP	148
B.2.1.5. UDDI	150
B.2.1.6. BPEL	150
B.2.1.7. SOA	152
B.3. Aspectos arquitecturales de los Web Services	155
B.3.1. Operaciones de los Web Services	155
B.3.1.1. Descripción del “Servicio”	155
B.3.1.2. Descubrimiento del “Servicio”	158
B.3.1.3. Interacciones del “Servicio”	158
B.3.1.4. La combinación de Web Services: Composición	160
B.3.2. Los Web Services y la computación Distribuida	160
B.3.2.1. ¿Qué es el middleware?	160
B.3.2.2. ¿Qué es una plataforma EAI?	161

B.3.2.3. Web Services y EAI.....	162
B.3.3. Arquitectura global de los Web Services.	165
B.3.3.1. Las dos facetas de arquitecturas de los Web Services	165
B.3.3.2. Arquitectura Interna de un Web Service.....	167
B.3.3.3. Arquitectura Externa de un Web Service	169
ANEXO C: WSDL y UDDI.....	174
C.1. Web Service Description Language	175
C.1.1. XML: el lenguaje de los Web Services	175
C.1.1.1. Vista general de XML.....	175
C.1.1.2. Lenguajes de Marcas	175
C.1.1.3. Significado e Historia de XML	176
C.1.1.4. Estructura del XML	178
C.1.2. XML y los Web Services	179
C.1.3. WSDL y la documentación de Web Services	180
C.1.4. WSDL y la descripción de los Web Services	181
C.1.5. Estructura de un documento WSDL.....	182
C.1.5.1. El Elemento Types.....	185
C.1.5.2. El Elemento Interface	185
C.1.5.3. El Elemento Binding.....	187
C.1.5.4. El Elemento Service	188
C.1.5.5. Elemento message [WSDL 1.1].....	188
C.1.5.6. Elemento portType [WSDL 1.1]	189
C.1.5.7. Elementos de Extensibilidad.....	190
C.2. Universal Description Discovery and Integration	190
C.2.1. Estudio de UDDI	190
C.2.2. Estructuras de Datos UDDI.....	192
C.2.2.1. BusinessEntity.....	194
C.2.2.2. BusinessService.....	199
C.2.2.3. BindingTemplate.....	199
C.2.2.4. tModel	201
C.2.2.5. PublisherAssertion.....	202
C.2.3 APIs UDDI.....	203
C.2.3.1. API Inquiry	207
C.2.3.2. API Publication.....	207
C.2.3.4. API Security Policy	210

C.2.3.5. API Custody & Ownership Transfer	210
C.2.3.6. API Subscription.....	211
C.2.3.7. API Value Set.....	212
C.2.3.8. API Replication.....	212
C.2.4. Extensibilidad de UDDI	213

Índice de Figuras

Figura 2.1 Conjunto de operaciones definidas en UDDI	24
Figura 2.2 Mensaje SOAP de la operación find_service	24
Figura 2.3 Resultado del mensaje SOAP de la operación find_service	25
Figura 2.4 Mensaje SOAP de la operación get_serviceDetail.....	25
Figura 2.5 Respuesta del mensaje SOAP de la operación get_serviceDetail	25
Figura 2.6 Mensaje SOAP de la operación save_service	26
Figura 2.7 Pantalla de bienvenida UDDI Browser.....	27
Figura 2.8 Pantalla UDDI Registries	27
Figura 2.9 Pantalla Find advance search options	28
Figura 3.1. Componentes del modelado y de la ejecución de Workflows.....	38
Figura 3.2. Componentes genéricos de un Workflow.	40
Figura 3.3. Arquitectura del Modelo de Referencia de SGWF.	42
Figura 3.4. Componentes que participan en la Interfaz 3.	44
Figura 4.2. Arquitectura SOA	55
Figura 4.3. Dinámica de los Web Services	57
Figura 4.4. Elementos de documentos WSDL – Definición del Web Service.	58
Figura 5.1. Interacción entre la herramienta y las aplicaciones cliente.....	68
Figura 5.2. Modelo genérico de la herramienta GG-WSfinder.	69
Figura 5.3. Diagrama de clases de diseño.	71
Figura 5.4. Pantalla inicial de la herramienta.....	71
Figura 5.5. Operación GetWebService.	72
Figura 5.6. Invocación de la operación GetWebService.....	72
Figura 5.7. Estructura genérica de la operación getTemplate().	73
Figura 5.8. Estructura genérica de la operación getQoS().	74
Figura 5.9. Modelo persistente de la herramienta GG-WSfinder	76
Figura 5.10. Clase WebService.java	77
Figura 5.11. Clase Operation.java.....	77
Figura 5.12. Clase Parameter.java	78
Figura 5.13. Clase Record.java.....	79
Figura 5.14. Clase RecordOp.java	79
Figura 5.15. Diagrama de secuencia para el escenario 1.	80
Figura 5.16. Diagrama de secuencia para el escenario 2.	81
Figura 5.16. Plantilla Result e instancia WeatherResult.....	83

Figura 5.17. Clase WeatherResult.java	84
Figura 5.18. Consulta SQL a la BD.....	86
Figura A.1. AND-Split	110
Figura A.2. AND-Join	110
Figura A.3. Or-Split.....	110
Figura A.4. OR-Join	111
Figura A.5. LOOP	111
Figura A.6. Diagrama del Ejemplo.	113
Figura A.7. Modelo de componentes del Workflow.	114
Figura A.8. Implementación del Workflow Enactment Service.	116
Figura A.9. Aplicaciones Cliente del Workflow.....	118
Figura A.10. Interfaces del Modelo de Referencia.....	119
Figura A.11. Estados básicos de un proceso.....	121
Figura A.12. Estados básicos de una actividad.	122
Figura A.13. Definición de procesos.	124
Figura A.14. Interface del Workflow con aplicaciones clientes.	125
Figura A.15. Aplicaciones Invocadas.....	126
Figura A.16. Modelo Encadenado	127
Figura A.17. Modelo de Sub-procesos anidados	127
Figura A.18. Modelo de sincronización paralela	128
Figura A.19. Modelo PEER-TO-PEER.	128
Figura A.20. Funciones de Interoperabilidad WAPI.	129
Figura A.21. Control de Interacciones en Tiempo de Ejecución.	130
Figura A.22. Interface para la Administración y monitoreo.....	131
Figura B.1. Mensaje HTTP GET	146
Figura B.2. Respuesta del Servidor	146
Figura B.3. Ejemplo de documento XML.	147
Figura B.4. Ejemplo de mensaje SOAP	148
Figura B.5. La figura muestra las relaciones de los diferentes elementos de un documento SOAP.....	148
Figura B.6. Ejemplo del elemento Fault.	149
Figura B.7. Equivalencia de las clases y objetos luego de ser serializados donde las clases se convierten en esquemas y los objetos en documentos XML y viceversa.	149
Figura B.8. Ciclo de vida de un Web Service en el que se serializa y se de-serializa en cada uno de los pasos.....	150
Figura B.9. Descripción del Servicio y Pila de Descubrimiento.	156

Figura B.10. Pila de interacción de servicios.	158
Figura B.11. Los Web Services proveen un punto de entrada para el acceso a servicios locales.	164
Figura B.12. Los Web Services también pueden ser utilizados sin estar en función de aplicaciones EAI.	165
Figura B.13. Los Web Services requieren tanto de una Arquitectura interna como una externa.	166
Figura B.14. Middleware convencional como una plataforma integradora de aplicaciones y programas básicos.	168
Figura B.15. Arquitectura básica de un conjunto de Web Services.	169
Figura B.16 Arquitectura Externa de un Servicio WB.	170
Figura B.17. Arquitectura Externa de un Web Service basada en un protocolo peer-to-peer y en composición de servicios.	172
Figura C.1. Estructura del documento WSDL.	184
Figura C.2. Elemento description del WSDL.	185
Figura C.3. Elemento types del WSDL.	185
Figura C.4. Elemento interface del WSDL.	186
Figura C.5. Elemento binding del WSDL.	187
Figura C.6. Elemento service del WSDL.	188
Figura C.7. Posibles operaciones de WSDL.	189
Figura C.8. Esquema del registro UDDI.	191
Figura C.9. Estructura bussinessEntity	196
Figura C.10. Estructura contact de bussinessEntity.	197
Figura C.11. Estructura de identifierBag de BussinessEntity.	198
Figura C.12. Estructura de categoryBag de BussinessEntity.	198
Figura C.13. Estructura de BussinessService.	199
Figura C.14. Estructura de BindingTemplate.	200
Figura C15. Estructura del tModel.	202
Figura C.16. Estructura de publisherAssertion.	202
Figura C.17. Estructura de operationallInfo.	203
Figura C.18. Estructura del elemento dispositionReport.	204

Índice de Tablas

Tabla 3.1. APIs para la invocación de aplicaciones.....	45
Tabla 4.1. Objetivos de los Web Services.....	53
Tabla 4.2. Roles, componentes y operaciones de la Arquitectura SOA	56
Tabla 4.2. Características de calidad de los Web Services	60
Tabla 5.1. Resultado de la consulta para el tipo “weather”	86
Tabla C.1. Especificaciones de XML.....	179

CAPÍTULO 1: Introducción

El presente capítulo brinda al lector una introducción general a los principales temas abordados por este trabajo en la sección 1.1. Luego, en la sección 1.2 se presentan la motivación y la consecuente propuesta que dieron lugar a la realización de esta tesis. Más tarde se enuncian los objetivos en la sección 1.3. Y finalmente en la sección 1.4 se presenta la estructura del presente documento.

1.1 Introducción

La WfMC (Workflow Management Coalition) [WfMC] define a un Workflow o Flujo de Trabajo como “La automatización de un Proceso de Empresa, total o parcial, en la cual documentos, información o tareas son pasadas de un participante a otro a los efectos de su procesamiento, de acuerdo a un conjunto de reglas previamente establecidas, con el fin de culminar una meta común impuesta por la empresa”. De la definición anterior podemos extraer que los Procesos de Empresa son un punto crítico dentro de lo que es el Workflow y que el objetivo central de éste es automatizar dichos procesos. Para ello, el modelo de la WfMC define cinco interfaces, que permiten a las aplicaciones del Workflow la comunicación a distintos niveles: La interfaz de Definición de Procesos (1), la interfaz del Workflow con aplicaciones clientes (2), la interfaz de Aplicaciones Invocadas (3), la interfaz de funciones de interoperabilidad (4) y la interfaz para la administración y monitoreo (5). En particular, la Interfaz de las Aplicaciones Invocadas se define para la invocación de las aplicaciones externas a un Workflow y es el punto central de este trabajo.

En efecto de lo anterior, los Sistemas de Gestión de Workflow (en adelante SGWF) son sistemas que definen, administran y coordinan la ejecución de los Workflows a través del uso de software que se ejecuta sobre uno o más motores de ejecución del Workflow, los que, a su vez, interpretan la definición de los procesos, interactúan con los otros participantes del Workflow e invocan herramientas y aplicaciones externas [WfMC].

Por otra parte, uno de los conceptos de la última década que ha tomado importancia en el mundo del desarrollo de aplicaciones, es el de los Web Services. Desde el punto de vista más simple, se podría mencionar que los Web Services son Web Services, entendiendo por servicios “programas que pueden ser accedidos por la red” [SV07]. En este caso, lógicamente, la red sería la Web. Los Web Services 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.

Desde el punto de vista de un desarrollador, un Web Service es un componente independiente que posee un conjunto de funcionalidades que pueden ser accedidas desde cualquier lugar y plataforma. Desde cualquier lugar, quiere decir que estarán disponibles dichos servicios a través de un medio común de comunicación (la Web). Mientras que desde cualquier plataforma, se refiere a que los datos que se reciben y son enviados por los Web Services son independientes de la plataforma de origen o destino. Cabe mencionar que esto es logrado utilizando para la presentación de los datos el Lenguaje Extendido de Marcas (XML).

El módulo que implementa la invocación de aplicaciones de un SGWF representa software ya existente que el SGWF puede utilizar para la realización de ciertas actividades, teniendo en cuenta que, en principio, dicho software se puede encontrar en cualquier plataforma o lugar de la Web. La Interfaz 3 de un Workflow permite la comunicación entre este componente y el servicio de ejecución del Workflow, no sólo a nivel de invocación, sino de transformación de datos en formatos entendibles por ambos componentes. Hoy esta comunicación de los SGWF con las aplicaciones externas es estática. Es decir, el administrador de Workflow debe proporcionar al

motor tanto las direcciones como el tipo de las aplicaciones deseadas al momento de definir el Workflow y no en tiempo de ejecución. Esto evidentemente puede derivar en diversas problemáticas: cualquier acontecimiento producido en las aplicaciones externas se transferirá al Workflow al momento de su invocación, como por ejemplo la no disponibilidad de la aplicación ya sea temporal o no, o una actualización de la misma que conlleve un cambio de ubicación, e incluso una modificación del formato de respuesta que hasta entonces era entendible para el Workflow. Cualquiera de estos sucesos, provocará que el Workflow posponga o aborte la invocación de la aplicación.

Por otro lado, el creciente número de Web Services disponibles dentro de una organización y en la web, plantea un problema de búsqueda de los mismos. La tradicional búsqueda por palabra clave en la actualidad, es insuficiente en este contexto: los tipos específicos de las consultas que los usuarios requieren no son correctamente capturados y la estructura subyacente y la semántica de los Web Services no son explotados.

En consecuencia de lo expuesto hasta aquí, este trabajo se centra en proponer un Web Service que simplifique la invocación de aplicaciones externas a un SGWF. De modo que, en base a cierta caracterización del tipo de aplicación que el sistema requiera en un momento dado de su ejecución, éste logre abstraerse de cuál es la aplicación específica a invocar y la ubicación de la misma.

1.2 Motivación y Propuesta

La WfMC ha especificado un conjunto de WAPIs (Workflow Application Programming Interfaces) para la administración de los Workflows. 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 SGWF se conocen a priori las aplicaciones que se desean utilizar, también existen múltiples sistemas donde diferentes aplicaciones (de las cuales solo se conoce el tipo) que brindan un mismo servicio podrían ser requeridas por el motor del Workflow en cualquier momento de su ejecución. Esta tesis plantea que esta comunicación entre el SGWF y las aplicaciones de software externas se realice a través del uso de Web Services.

Los Web Services 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 normas de calidad de los Web Services (características no funcionales de los mismos), las cuales se refieren a la habilidad del servicio de responder a las invocaciones y llevarlas a cabo en consonancia con las expectativas tanto de quien lo provee como así también de quien lo usa. 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 Web Services existentes en la actualidad que proveen similares funcionalidades, aunque tienen diferente descripción sintáctica.

Este trabajo no sólo apunta a simplificar la comunicación del SGWF con las aplicaciones externas a través del uso de Web Services, sino también optimizando la selección de los mismos mediante la incorporación de las normas de calidad.

1.3 Objetivos

Como ya se ha dicho en el presente capítulo, el objetivo principal del trabajo es simplificar y optimizar la invocación de aplicaciones externas a un Workflow a través del uso de Web Services. En función de ello, se propone la realización e implementación de un Web Service que, dados el tipo de la aplicación que se necesite invocar, los requerimientos del sistema y ciertas normas de calidad, encuentre, ejecute y retorne el Web Service disponible (aplicación) más adecuado que cumpla con los requerimientos especificados. De esta forma, el Workflow se abstrae tanto de la ubicación específica como de la aplicación misma y su ejecución, simplificando su definición. A partir de esta meta general se desprenden los siguientes objetivos:

- Implementar un Web Service que, de acuerdo a los requerimientos del SGWF, proporcione la aplicación más adecuada que cumpla con estos. Y que a su vez, la obtención de la aplicación, como así también su ejecución, resulte transparente al motor de Workflow.
- Incorporar a la implementación del Web Service, las características de calidad (QoS), es decir, los requerimientos no funcionales de las aplicaciones.
- Proporcionar una salida estándar para cada tipo de aplicación que el SGWF invoque.

La especificación de la Interfaz de las Aplicaciones Invocadas con Web Services permite la distribución transparente de las aplicaciones externas. De esta manera, el motor del SGWF puede invocarlas sin la necesidad de conocer su ubicación exacta, con el beneficio de que las aplicaciones pueden cambiar su ubicación en la red sin que esto implique ningún cambio en su invocación. Al mismo tiempo, logrando que el SGWF interactúe con el Web Service propuesto en este trabajo, el motor se desliga de la costosa responsabilidad que la WfMC estipula a través de la especificación de la Interfaz 3. Esto es, implementar el proceso de búsqueda, selección e invocación de aplicaciones externas necesarias para cumplir con los requisitos del sistema.

Asimismo, con la incorporación de los atributos de calidad en la selección de los Web Services se hace posible evaluar la calidad de los mismos y de esta forma obtener los más adecuados.

1.4 Estructura del documento

La estructura de este trabajo respeta el orden de la siguiente enumeración:

- El capítulo 1 consta de la introducción al trabajo.
- El capítulo 2 hace un análisis del estado del arte respecto a la registración y búsqueda de Web Services.
- El capítulo 3 describe los Workflows y los SGWF, detallando la definición oficial provista por la Workflow Management Coalition (WfMC) y la

invocación de Aplicaciones Externas a un SGWF según el Modelo de referencia de la WfMC.

- El capítulo 4 contiene los conceptos generales de los Web Services, sus arquitecturas, el lenguaje WSDL utilizado para especificarlos y el protocolo UDDI utilizado para registrarlos y buscarlos. También introduce el concepto de las características de calidad en los Web Services.
- El capítulo 5 presenta la herramienta que implementa el proceso de búsqueda y ejecución de Web Services: GG-WSfinder.
- El capítulo 6 presenta las conclusiones de la Tesis y las principales líneas de trabajos futuros.
- El Anexo A consta de un resumen de las principales tecnologías de los Workflows, y detalla la especificación formal de los mismos por parte de la WfMC.
- El Anexo B presenta las tecnologías básicas que se utilizan en la definición y uso de los Web Services.
- El Anexo C detalla por separado las dos tecnologías de los Web Services en las que se basa este trabajo: el mecanismo de descubrimiento de los Web Services UDDI y el lenguaje de especificación con el que se los define WSDL.

CAPÍTULO 2: Estado del Arte

Este capítulo hace una revisión del estado del arte en lo que respecta a los siguientes puntos: comunicación de los Sistemas de Gestión de Workflow en la sección 2.1; registración y búsqueda de Web Services que se detalla en la sección 2.2; Y finalmente la incorporación de las características de calidad a los Web Services detalladas en la sección 2.3.

2.1. Comunicación de los SGWF con las aplicaciones externas

La Interfaz 3 de un Workflow permite la comunicación (en la actualidad estática) entre este tipo de sistemas y el software disponible en el exterior de los mismos, no sólo a nivel invocación sino también a la hora de transferir datos en formatos entendibles por ambos componentes y la obtención e interpretación de los resultados. Para llevar a cabo esta comunicación, la presente tesis se apoya en la propuesta del siguiente trabajo:

En [Martellotto10], la autora propone una Especificación Funcional y otra No funcional de la Interfaz de Aplicaciones Invocadas, y de esta forma mejorar la selección de aplicaciones en tiempo de ejecución, permitiendo al motor del SGWF invocarlas dinámicamente independizándose de la ubicación exacta de las mismas. En la especificación funcional plantea definir las WAPIs de la Interfaz de Aplicaciones Invocadas con Web Services que declaren la información de entrada, los datos de salida y los posibles errores en la invocación. Esto no sólo brinda mayor flexibilidad al motor del SGWF sino que también permite el requerimiento y la actualización de datos de la aplicación y otras funcionalidades importantes en tiempo de ejecución. En la especificación no funcional de la Interfaz de Aplicaciones Invocadas, propone incorporar al proceso de selección de Web Services el análisis de los atributos de calidad requeridos por el solicitante. De esta forma se complementa la selección en tiempo de ejecución con la obtención del mejor Web Service disponible que satisfaga las necesidades del solicitante.

2.2. Registración y búsqueda de Web Services

Existen diversas herramientas que implementan formas de registrar y buscar Web Services tanto en servidores locales como en servidores distribuidos en la web. A continuación se mencionan y describen las tres herramientas de mayor jerarquía que fueron estudiadas en este trabajo:

2.2.1 JUDDI Console

[Juddi-console] es una herramienta web que trabaja localmente sobre un servidor de aplicaciones como Tomcat, el cual consta de 3 APIS y un conjunto de operaciones definidas en UDDI implementadas en cada una de ellas tal y como se muestra en la figura 2.1.

jUDDI Console	
jUDDI API (proprietary)	UDDI Publish API
get_registryInfo	get authToken
find_publisher	get registeredInfo
get_publisherDetail	discard authToken
save_publisher	save business
delete_publisher	save service
UDDI Inquiry API	save binding
find_business	save tModel
find_service	delete business
find_binding	delete service
find tModel	delete binding
find_relatedBusinesses	delete tModel
get_businessDetail	add_publisherAssertions
get_businessDetailExt	set_publisherAssertions
get_serviceDetail	get_publisherAssertions
get_bindingDetail	delete_publisherAssertions
get tModelDetail	get assertionStatusReport

Figura 2.1 Conjunto de operaciones definidas en UDDI

Particularmente, la operación `find_service()` correspondiente a la API inquiry realiza una búsqueda sintáctica sobre los Web Services alojados en el servidor local. La figura 2.2 muestra el mensaje SOAP con el que la aplicación realiza la mencionada búsqueda.

jUDDI Console

find_service

The [find_service](#) API call returns a [serviceList](#) message that contains zero or more [serviceInfo](#) structures matching the criteria specified in the argument list. If an error occurs while processing this API call, a [dispositionReport](#) element will be returned to the caller within a [SOAP Fault](#) containing information about the [error](#) that was encountered.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <find_service businessKey="***" generic="2.0" xmlns="urn:uddi-org:api_v2">
      <findQualifiers>
        <findQualifier>***</findQualifier>
      </findQualifiers>
      <name>***</name>
      <categoryBag>
        <keyedReference tModelKey="***" keyName="***" keyValue="***" />
      </categoryBag>
      <tModelBag>
        <tModelKey>***</tModelKey>
      </tModelBag>
    </find_service>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.2 Mensaje SOAP de la operación `find_service`

El mensaje SOAP resultado de la búsqueda se muestra en la figura 2.3.


```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body xmlns="urn:uddi-org:api_v2">
    <serviceList generic="2.0" operator="autentia">
      <serviceInfos>
        <serviceInfo businessKey="668FA0E0-8EC3-11DF-AA7B-C96305994741" serviceKey="8D494380-8EC3-11DF-AA7B-F5FB778A5A9D">
          <name>Conversor a Braille</name>
        </serviceInfo>
        <serviceInfo businessKey="61E5DD00-8EC5-11DF-AA7B-D280032C0C23" serviceKey="EA1FB6E0-8EC6-11DF-AA7B-BE7E6FAB6210">
          <name>GlobalWeather</name>
        </serviceInfo>
        <serviceInfo businessKey="61E5DD00-8EC5-11DF-AA7B-D280032C0C23" serviceKey="F1016800-A960-11DF-A8BC-C963C14774EF">
          <name>WeatherForecast</name>
        </serviceInfo>
      </serviceInfos>
    </serviceList>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.3 Resultado del mensaje SOAP de la operación find_service

Por otro lado, la operación `get_serviceDetail()`, también perteneciente a la API *inquiry*, como su nombre lo especifica, retorna el detalle de un Web Service determinado identificado por su clave. Las figuras 2.4 y 2.5 muestran cuales son los mensajes SOAP que la herramienta maneja para realizar dicho proceso de obtención de detalles de un Web Service.

jUDDI Console

get_serviceDetail

The [get_serviceDetail](#) API call is used to request full information about a known [businessService](#) structure. If an error occurs while processing this API call, a [dispositionReport](#) element will be returned to the caller within a [SOAP Fault](#) containing information about the [error](#) that was encountered.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <get_serviceDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
      <serviceKey>***</serviceKey>
    </get_serviceDetail>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.4 Mensaje SOAP de la operación `get serviceDetail`

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body xmlns="urn:uddi-org:api_v2">
    <serviceDetail generic="2.0" operator="autentia">
      <businessService businessKey="61E5DD00-8EC5-11DF-AA7B-D280032C0C23" serviceKey="EA1FB6E0-8EC6-11DF-AA7B-BE7E6FAB6210">
        <name>GlobalWeather</name>
        <description>Dada una localidad valida, podemos obtener su nivel de viento, visibilidad, temperatura, humedad, presion atmosferica, etc.</description>
        <bindingTemplate bindingKey="EA20A140-8EC6-11DF-AA7B-A7450944CF07" serviceKey="EA1FB6E0-8EC6-11DF-AA7B-BE7E6FAB6210">
          <accessPoint URLType="http">http://www.webservicex.net/globalweather.asmx</accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4">
              <instanceDetails>
                <overviewDoc>
                  <overviewURL>http://www.webservicex.net/globalweather.asmx?wsdl</overviewURL>
                </overviewDoc>
              </instanceDetails>
            </tModelInstanceInfo>
          </tModelInstanceDetails>
        </bindingTemplate>
      </businessService>
    </serviceDetail>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.5 Respuesta del mensaje SOAP de la operación `get_serviceDetail`

Finalmente, otra operación de importancia que cabe destacar es la `save_service()` de la API *publish*, la cual se encarga de almacenar y publicar un Web Service en el servidor. La figura 2.6 muestra su correspondiente mensaje SOAP.

jUDDI Console

save_service

The `save_service` API call is used to add or update one or more `businessService` elements. If an error occurs while processing this API call, a `dispositionReport` element will be returned to the caller within a `SOAP Fault` containing information about the `error` that was encountered.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_service generic="2.0" xmlns="urn:uddie-org:api_v2">
      <authInfo>***</authInfo>
      <businessService businessKey="***" serviceKey="">
        <name>***</name>
        <description>***</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="">
            <accessPoint URLType="http">***</accessPoint>
            <tModelInstanceDetails>
              <tModelInstanceInfo tModelKey="***">
                <instanceDetails>
                  <overviewDoc>
                    <overviewURL>***</overviewURL>
                  </overviewDoc>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>
        </bindingTemplates>
      </businessService>
    </save_service>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.6 Mensaje SOAP de la operación `save_service`

La aplicación [Juddi-console] es de una gran utilidad ya que permite al desarrollador visualizar y comprender de forma transparente como se manipulan los Web Services en un servidor UDDI, particularmente uno instanciado de manera local.

2.2.2 UDDI Browser

[Uddi-browser] es otra herramienta de registración y búsqueda de Web Services, pero que esta vez se trata de una herramienta no web, sino implementada en Java y que además brinda la posibilidad de elegir distintos servidores UDDI sobre los cuales realizar las búsquedas. La figura 2.7 muestra la pantalla de bienvenida de la aplicación.

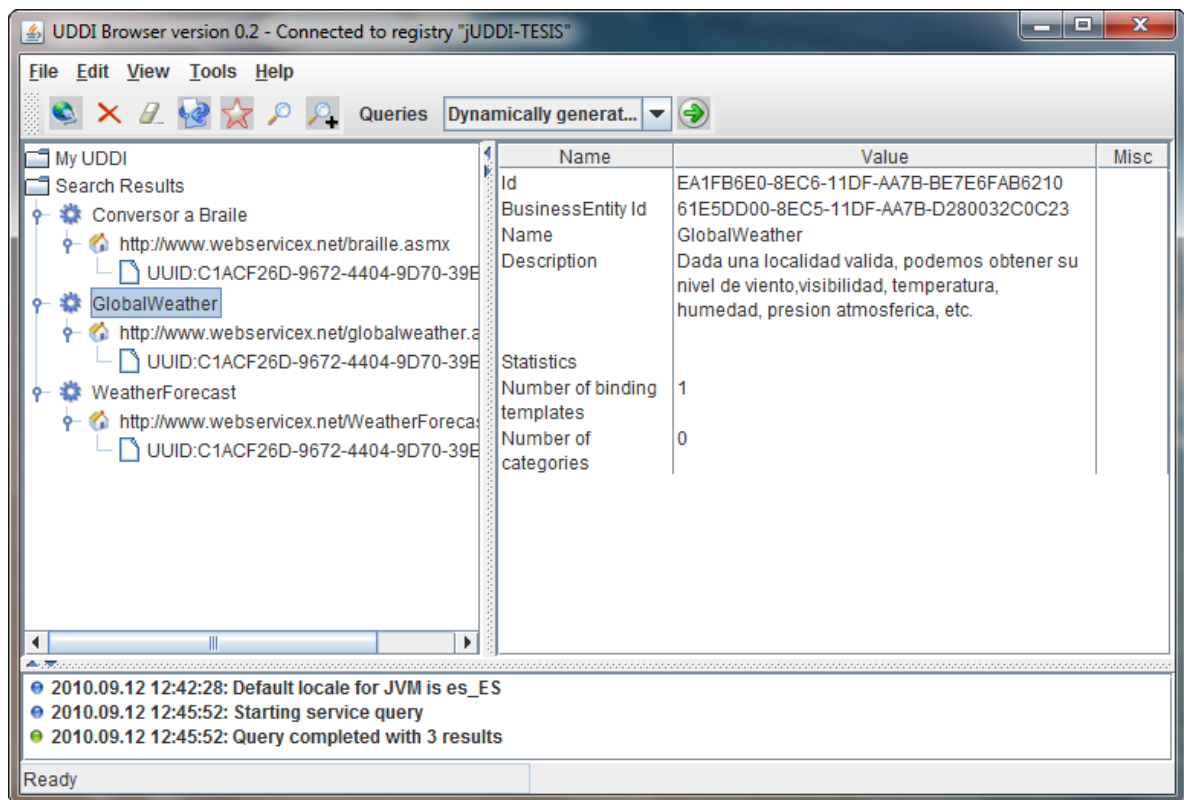


Figura 2.7 Pantalla de bienvenida UDDI Browser

Asimismo, los registros UDDI sobre los que opera se muestran en la figura 2.8.

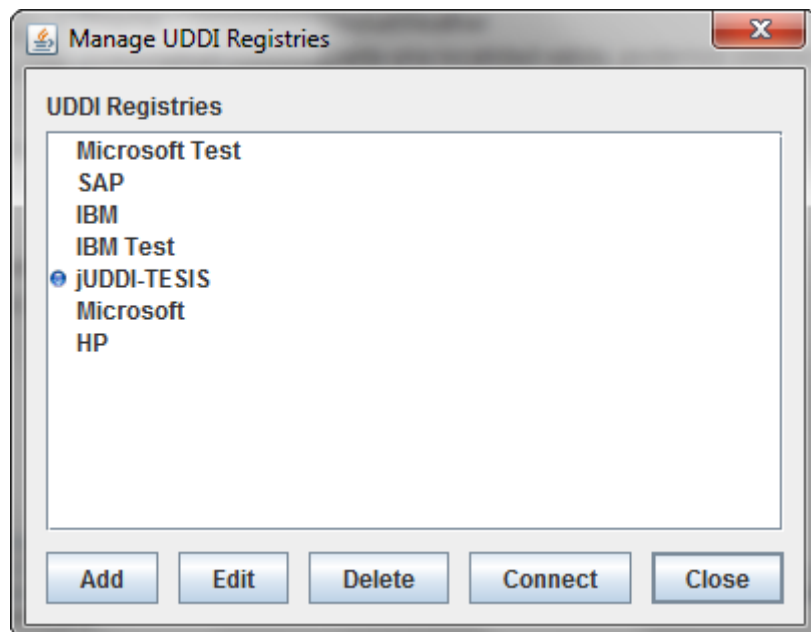


Figura 2.8 Pantalla UDDI Registries

La búsqueda se realiza completando una pantalla que se describe en la figura 2.9.

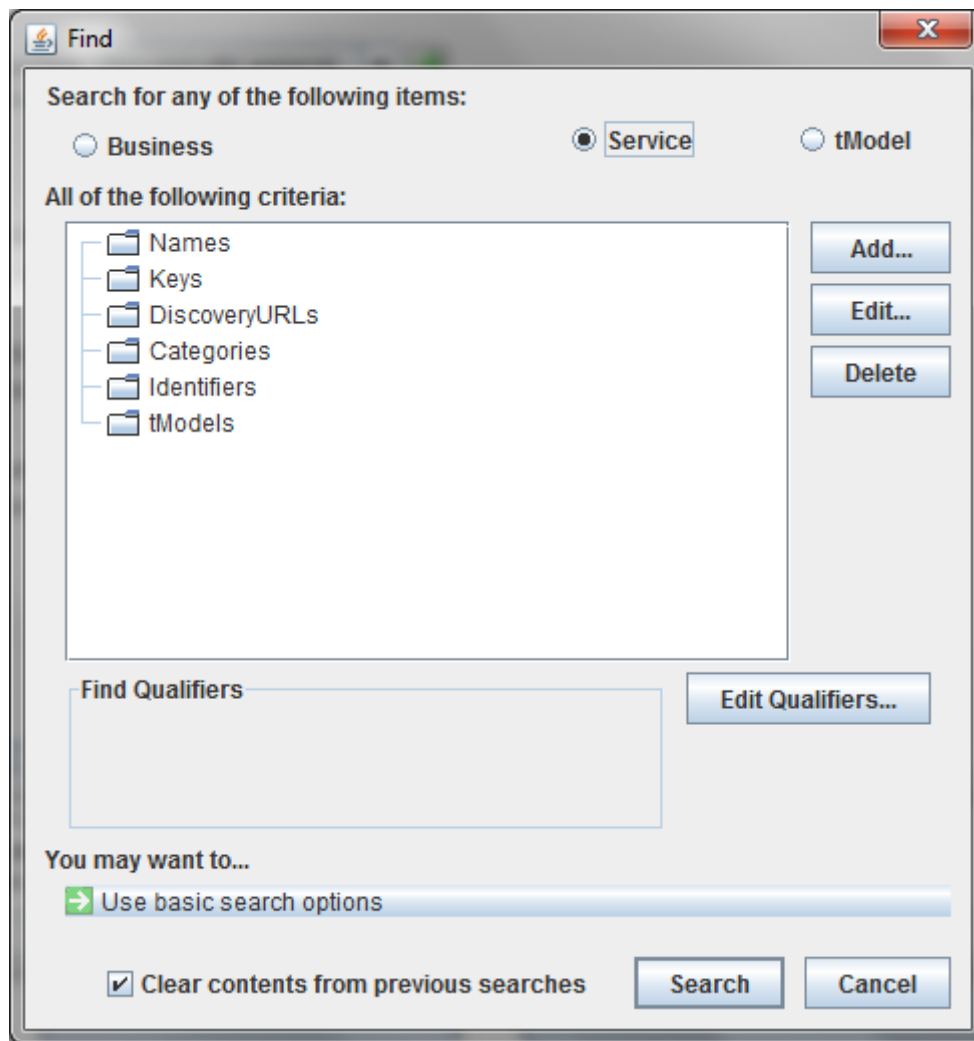


Figura 2.9 Pantalla Find advance search options

UDDI Browser es un proyecto de código abierto que proporciona una interfaz fácil de usar. Permite a los usuarios ver y manipular el contenido de los registros UDDI. Soporta toda la API de UDDI para la gestión de registros, incluyendo la consulta de todo el conjunto de APIs, así como también la creación, modificación y eliminación de las entidades en UDDI. También tiene una serie de características para simplificar a los usuarios y administradores de UDDI la operación sobre los registros, incluyendo la persistencia de las consultas, el soporte UDDI, y las utilidades del administrador para la mantención de los registros.

2.2.3 Woogle

En [Woogle] se definen los Web Services como componentes de software débilmente acoplados, publicados, que se encuentran e invocan a través de la web. Este trabajo acomete al problema de búsqueda y selección de Web Services a través de la descripción de algoritmos en los que se basa el motor de búsqueda Woogle para encontrar Web Services. Woogle soporta la búsqueda de similitudes en los Web Services, tales como encontrar similares operaciones de diferentes servicios publicados

en Internet. Es decir, se basa en un análisis semántico de los Web Services y sus operaciones y en función de ello, se describen técnicas para soportar este tipo de búsquedas.

Woogole es un ejemplo preciso de que la selección automatizada de Web Services es compleja y que se encuentra en plena discusión en la actualidad.

2.3. Incorporación de las características de calidad a los Web Services

Finalmente, teniendo en cuenta no sólo el registro y publicación de los Web Services sino los atributos de calidad de los mismos a la hora de la descripción y selección de estos, existen diversos trabajos que valen la pena destacar los cuales fueron vistos en esta tesis:

[KOK05] presenta un framework para mejorar el descubrimiento de los Web Services. Este modelo agrupa clientes según sus preferencias y a cada grupo se le asigna un Personal Agent (PA). El trabajo de cada PA es procesar todas y cada una de las actividades que los Web Services deben realizar, es decir, comunicación con los registros UDDI, bindings, requerimientos y respuestas. Entonces, una vez que cada PA reciba y acepte los requerimientos de sus clientes, enviará esa información al Matching Agent (MA), el cual no solo procesa esta información sino que también administra el registro de los Web Services disponibles. El MA busca los Web Services que satisfagan los requerimientos transferidos por el solicitante y le retorna la lista de ellos en conjunto con aquellos PA que hayan buscado con anterioridad los Web Services de la lista. Por otra parte, el MA mantiene un historial de requerimientos solicitados por cada PA. Luego de recibir la lista, el PA solicita información a sus pares que ya han utilizado los Web Services de la lista y, si los Web Services 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 Web Services retornados. Si la puntuación es baja, la relevancia de los Web Services 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 Web Services combinando las puntuaciones del matching, las recomendaciones y las evaluaciones. Finalmente, se invoca el mejor Web Service, se miden los parámetros de las características de calidad, y se graban los resultados.

En [HSD08], los autores proponen perfeccionar el proceso de descubrimiento de Web Services 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 de las características de calidad. Este framework se basa en los componentes básicos de un Web Service (Proveedor del servicio, Consumidor del servicio, Registro UDDI) con la capacidad adicional de almacenar las características de calidad 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 Web Services 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 de calidad especificados. El Discovery Agent comenzará entonces un test inicial y enviará el resultado al consumidor. El Discovery Agent puede usar la historia de los Web Services 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 Web Service de la puntuación de sus atributos de calidad, la recomendación del Reputation Manager y los resultados del test inicial, y finalmente selecciona el Web Service más apropiado de acuerdo a su análisis. Este modelo no requiere modificar el modelo de Web Services actual y permite compartir el conocimiento entre el proveedor y el consumidor del servicio a través de los agentes que participan.

2.4. Conclusiones

Lograr una óptima y simple comunicación entre los SGWF y las aplicaciones externas es una de las principales metas de esta tesis. Entonces, al estudiar diversos trabajos, se optó por analizar cuidadosa y detalladamente aquellos cuyos objetivos eran similares. En efecto, el análisis que se ha realizado sobre ellos, resultó en un aporte fundamental para la culminación y el cumplimiento de todos los objetivos planteados para la presente tesis.

Es el caso del trabajo [Martelloto10] presentado en la sección 2.1, que en orden de optimizar la mencionada comunicación entre los SGWF y las aplicaciones externas, propone la especificación funcional y la especificación no funcional de la interfaz 3 definida por la WfMC con Web Services. Las ventajas que se observan son numerosas y consecuentes con las prestaciones de los Web Services. Sin embargo y a pesar de ello, el trabajo requiere de cambios en la especificación de la Interfaz de las Aplicaciones Invocadas del Modelo de Referencia de Workflow. Esto es muy costo, de una complejidad relativamente alta y de una compatibilidad limitada respecto a los SGWF actuales.

Más tarde, se presentan una serie de herramientas que cooperaron al desarrollo de la registración y búsqueda de Web Services. Lógicamente, por un lado existen los trabajos que plantean (e implementan) estas búsquedas de acuerdo, y en consecuencia, a las especificaciones de los servidores UDDI disponibles en la actualidad. Se trata de búsquedas sintácticas que pueden realizarse por nombre, descripción o proveedor de los Web Services (ver [Uddi-browser] o [Juddi-console]). Y, por otra parte, los trabajos que proponen el uso de ontologías para describir la semántica de los Web Services (a modo de ejemplo se describió brevemente [Woogle]). Pero, si bien ciertas informaciones son eminentemente adecuadas para ser representadas por medio de ontologías, otras con certeza no. En consecuencia, se han argumentado que las ontologías no son apropiadas para muchas aplicaciones del mundo real una vez que adquieren un determinado nivel de complejidad. Por otra parte, resultan difíciles de mantener y materializan un punto de vista particular del dominio de conocimiento.

Es importante remarcar que 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 Web Service 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 está desarrollando un programa que utiliza un Web Service, el desarrollador consulta manualmente un servidor UDDI y selecciona el Web Service adecuado de acuerdo a su descripción en lenguaje natural. La información del Web Service 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.

Ahora bien, con respecto a la incorporación de las características de calidad en la descripción de los Web Services, el principal inconveniente que se observa en las propuestas analizadas en la sección 2.3, es la complejidad que las mismas requieren. Se trata de modelos con una considerable cantidad de estructuras de datos, y por ende magnitudes de programación costosas.

Finalmente, cabe destacar que existe otra línea en lo que a incorporación de características de calidad en los Web Services refiere. Sin embargo, el factor común de esta línea radica en modificar el modelo subyacente de descubrimiento de Web Services. Y, de igual manera que [Woogole] y algunos otros trabajos que incorporan el concepto de análisis semántico y ontologías en los Web Services, escapa a los lineamientos trazados para esta tesis. Algunos de ellos son: [Ran03], [D'Ambrogio06], [VHA06], [ZHYJ07], [BSSB08].

CAPÍTULO 3: Workflows

En este capítulo se describen los Workflows, los Sistemas de Gestión de Workflows, sus componentes y la estandarización que provee la Workflow Management Coalition haciendo especial hincapié en la interfaz 3 de los Workflows: la Interfaz de Invocación de Aplicaciones Externas.

La estructura del capítulo es la siguiente: en la sección 3.1 se introducen los conceptos generales de los Workflows y su contexto tecnológico. La sección 3.2 describe el lenguaje mundial utilizado al hablar de Workflows provisto por la Workflow Management Coalition. Luego en la sección 3.3 se presentan los componentes genéricos de un Sistema de Gestión de Workflow. Y finalmente la sección 3.4 describe las nociones genéricas a tener en cuenta a la hora de implementar la invocación de aplicaciones externas a un Workflow.

3.1. Los Workflows

Uno de los problemas que se encuentra habitualmente en el desarrollo de aplicaciones para Negocios, es que las tareas o procesos que se desarrollan en el entorno laboral de las mismas quedan inmersos en el código de la aplicación que resuelve la problemática de la Negocio. Está claro que la gran mayoría de los usuarios no tienen conocimiento de estas tareas, las mismas están ocultas a sus ojos y se realizan automáticamente. El hecho de realizar cambios en dichas tareas o procesos resulta muy costoso, y es muy factible que dichos cambios redunden en realizar nuevamente la aplicación.

Una buena solución al problema anterior es separar los procedimientos y asociarlos a los Workflows realizados dentro de la Negocio. Vemos entonces, que el Workflow se relaciona con la automatización de los procedimientos donde los documentos, la información o tareas son pasadas entre los participantes del sistema de acuerdo a un conjunto de reglas previamente establecidas. El fin de lo anterior es llegar a culminar una meta común impuesta por la Negocio.

Se puede ver al Workflow como un conjunto de métodos y tecnologías que ofrece las facilidades para modelar y gestionar los diversos procesos que ocurren dentro de una Negocio. El Workflow 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 competencia.

Es evidente que el contar con un sistema de Workflow 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 Workflow. A continuación se enumeran algunas razones por las cuales las organizaciones podrían considerar adoptar una solución de Workflow:

- ✓ Eficiencia en los procesos y estandarización de los mismos. Esto conduce a una reducción de costos dentro de una Negocio.
- ✓ La estandarización de los procesos lleva a tener un mayor conocimiento de los mismos, lo que a su vez conduce a obtener una mejor calidad de estos.
- ✓ Control de los Procesos (Process Management). Utilizando la tecnología de Workflow es posible monitorear el estado actual de las tareas así como también observar cómo evolucionan los planes de trabajo realizados. Permite ver cuáles son los embotellamientos dentro del sistema, es decir aquellas tareas o decisiones que están requiriendo de tiempo no planificado y se tornan en tareas o decisiones críticas.
- ✓ Asignación de tareas a la gente. La asignación de tareas se realiza mediante la definición de roles dentro de la Negocio, eliminando la tediosa tarea de asignar los trabajos caso por caso.
- ✓ Recursos disponibles. Se asegura que los recursos de información (aplicaciones y datos) van a estar disponibles para los trabajadores cuando ellos los requieran.

- ✓ Diseño de procesos. Se fomenta a pensar los procesos de una manera distinta a la tradicional forma jerárquica que se utiliza para diseñarlos en la actualidad.

Hay además muchos aspectos operacionales por los cuales es deseable contar con una tecnología de Workflow ya que cosas como la secuencia de tareas, quienes la realizan, mecanismos de control y monitoreo, son implementadas por software de Workflow.

El Workflow entonces permite automatizar diferentes aspectos del flujo de la información: rutear los trabajos en la secuencia correcta, proveer acceso a datos y documentos, y manejar ciertos aspectos de la ejecución de un proceso.

La diversidad de procesos que puede haber en una organización lleva a pensar en la existencia de diferentes tipos de software de Workflow. El Workflow entonces, da a un Negocio la posibilidad de automatizar sus procesos, reducir costos, y mejorar servicios, obvios beneficios. Así pues, organizaciones que no hayan evaluado esta tecnología podrían encontrarse con desventajas en un futuro.

3.2. Workflow Management Coalition (WfMC)

La WfMC es una agrupación compuesta por compañías, vendedores, organizaciones de usuarios, y consultores. El objetivo de esta agrupación es ofrecer una forma de "diálogo" común a todos. De esta forma las diferentes herramientas que se implementen en esta área podrán tener cierto nivel de interoperabilidad, es decir, podrán comunicarse entre ellas para poder realizar las distintas tareas involucradas en un sistema de Workflow.

3.2.1. Necesidad de Estandarizar

Se estima que actualmente los distintos productos de Workflow que hay en el mercado sobrepasan los cien. Cada uno de ellos se enfoca sobre distintos aspectos funcionales, como ser, herramientas de diseño visual, en las cuales se ofrecen ciertos diagramas para representar la realidad. Otras se enfocan en la integración de los datos con las aplicaciones.

El desarrollo de estándares para la interoperabilidad de las diversas herramientas, permitiría la elección de los mejores productos, según el enfoque que se le dé a cada aplicación. Por ejemplo, se puede comprar una herramienta que ofrezca un entorno amigable para realizar el análisis y diseño del problema, y por otro lado comprar otra herramienta que resuelva la auditoria de los datos y poder trabajar en forma integrada con los dos componentes.

Una de las estrategias que siguen actualmente los Negocios, es rediseñar sus procesos, esta metodología es denominada como Reingeniería de los Procesos de Negocios (Business Process Re-engineering). Esto puede ser causa de cambios organizacionales, legislativos, cambios en los objetivos del negocio, etc. En esta situación, muchas veces es necesario relacionarse con otras organizaciones. Pero para poder hacer esto debe existir la posibilidad de que los productos de un vendedor puedan comunicarse con los de otro, pues es claro que cada Negocio u organización comprará los productos que crea conveniente para su caso.

Vemos entonces la necesidad de estandarizar la forma de comunicación entre los distintos componentes de un producto de Workflow. De modo de poder tener flexibilidad a la hora de operar con distintos productos. Esta necesidad se justifica además por las proyecciones que se tienen actualmente, sobre la penetración de la tecnología de Workflow en el mercado en los próximos años. Por esto se debe atacar el problema de potenciales incompatibilidades de antemano, y no cuando existan miles de productos en esta área, cada uno con sus particularidades.

3.2.2. Definiciones de la WfMC

- Procesos de Negocio (Business Process):

"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].

Si se observa cualquier Negocio de la actualidad, se puede apreciar que la misma está constituida por varias partes. Son estas partes quienes tienen que relacionarse fluidamente para que se cumpla el cometido del Negocio. La definición anterior plantea este contexto, en el cual para que todo funcione correctamente, se deben definir previamente la forma en que se comunicarán dichas partes. Actualmente no existe un buen grado de formalización de los procesos de Negocio. Un área con gran potencial es rediseñar procesos existentes, eliminar redundancias, identificar embotellamientos y entender porque es que se hace lo que se hace.

- Workflow:

"Un Workflow es la automatización de los procesos de negocio, total o parcial, 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 Negocio". Workflow Management Coalition [WfMCa].




De la definición anterior se puede extraer que los Procesos de Negocio son un punto crítico dentro de lo que es el Workflow y que el objetivo central de éste es automatizar dichos procesos. No se lo puede ver como algo totalmente librado al azar, debe existir una estructura que le dé la lógica de coordinación entre los participantes. Es necesario que existan dos o más individuos para poder hablar de Workflow, y además, estos individuos deben cooperar para alcanzar una meta común.

- Sistemas de Gestión de Workflow:

"Los Sistemas de Gestión de Workflow (SGWF) son sistemas que definen, crean y administran la ejecución de los Workflows a través del uso de software que se ejecuta sobre uno o más motores de ejecución del Workflow, los que interpretan la definición de los procesos, interactúan con los otros participantes

del Workflow e invocan herramientas y aplicaciones”. Workflow Management Coalition [WfMCa].

Los SGWF presentan muchas características en común, las cuales se consideran la base para el desarrollo de capacidades de integración e interoperabilidad entre diferentes productos. En el nivel más alto de abstracción, todos los SGWF proveen tres tipos de funcionalidades:

-  Funciones en tiempo de construcción: dedicadas a la definición y modelado de un Workflow junto con todas las actividades que se deban definir.
-  Funciones de control en tiempo de ejecución: dedicadas al control del Workflow 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 Workflow puedan llevar a cabo sus tareas o actividades.

En la ejecución de estas funcionalidades participa uno de los componentes más importantes de un SGWF, su motor. El motor de Workflow es el responsable de interpretar la definición de procesos, interactuar con los participantes del mismo cuando esto sea requerido, e invocar el uso de herramientas y aplicaciones de tecnología de información.

3.2.3. Conceptos sobre Workflows

De acuerdo a la definición de la WfMC, en un SGWF existen dos actividades bien diferenciadas. Por un lado se tiene la definición del Workflow que implementa el proceso de negocio, lo que se denomina modelado del Workflow. Este modelado 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 SGWF.

Y por otro lado está la ejecución de dicho modelo. La ejecución del Workflow en un SGWF consiste en la creación, manipulación y destrucción de instancias de proceso, que representan al Workflow 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 Workflow, 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 Workflow, donde se define por quienes deben ser ejecutadas.
- *Invocación de Aplicaciones*: es la representación de las aplicaciones del Workflow que son invocadas por el SGWF para automatizar una actividad.

Los componentes básicos de un Workflow y sus relaciones, para las dos fases descriptas, la de modelado y la de ejecución, se ilustran en la Figura 3.1.

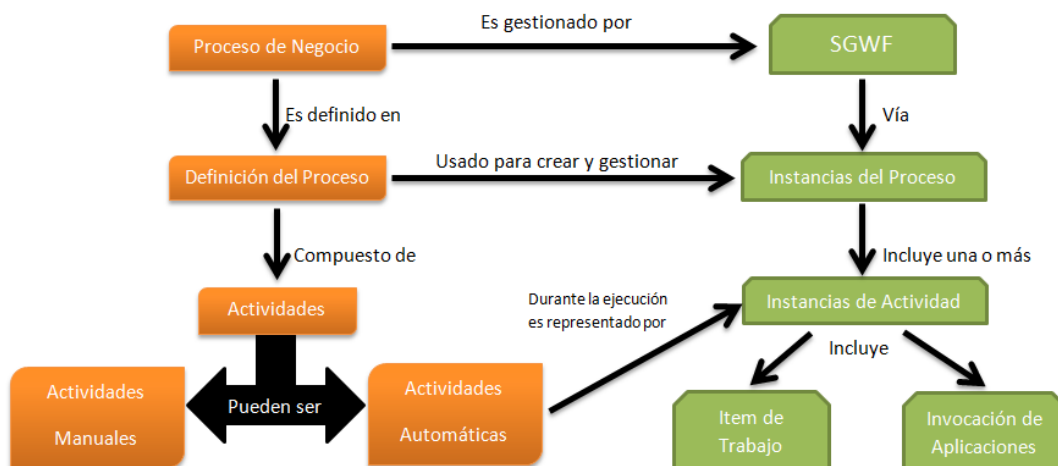


Figura 3.1. Componentes del modelado y de la ejecución de Workflows.

3.2.4. Ventajas

Cualquier usuario de un SGWF se asegura de que el trabajo no se extravíe o detenga, de que los gerentes puedan enfocarse en el personal y problemas de negocio. Además los procedimientos son formalmente documentados, la mejor persona (o máquina) es asignada para hacer cada trabajo y el procesamiento paralelo es práctico. Estas pueden considerarse como las principales ventajas al utilizar SGFW, sin embargo hay más.

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 SGWF 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 otro lado, los Workflows 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 Workflows tienen como misión apoyar procesos estructurados orientados a la administración caso a caso, en los que intervienen varios actores.

Es evidente que el contar con un SGWF 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 Workflows:

- 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 menos intervención de administración, y menos posibilidad a sufrir 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. Además, 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 Negocio, para el cliente y para los usuarios.

3.3. Componentes genéricos de un SGWF

A pesar de la gran variedad de productos de Workflow 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 Workflow, 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 Workflow puedan interactuar entre sí.

En un producto de software de Workflow genérico se identifican una serie de componentes e interfaces. La implementación de esta estructura puede ser realizada de varias formas diferentes entre sí. Éste es un punto de desencuentro de los productos existentes. A continuación se dará una descripción de las principales componentes de un sistema de Workflow genérico.

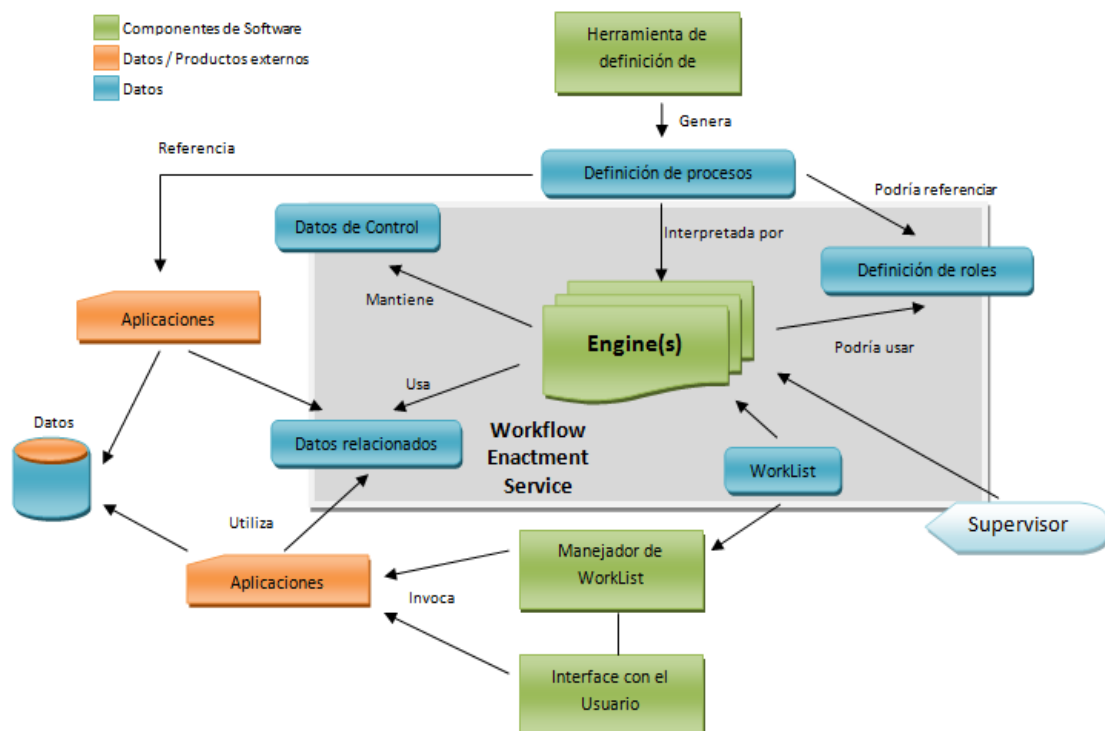


Figura 3.2. Componentes genéricos de un Workflow.

En este modelo genérico se encuentra tres tipos de componentes:

- De software: Proveen soporte para gran cantidad de funciones del sistema de Workflow.
- Datos y Definición de procesos: Usados por los componentes de software.
- Aplicaciones externas.

A continuación se describirán en sub secciones los elementos más importantes mostrados en la figura 3.2.

3.3.1 Herramienta de Definición de Procesos

Forma parte de los componentes de software del Workflow y la podemos ver en el borde superior de la figura. Es utilizada para crear una descripción de los procesos en una forma procesable para una computadora. Esta herramienta podría estar basada en un lenguaje de definición de procesos formal, en un modelo de interacción entre objetos, o simplemente en un conjunto de reglas de ruteo para transferir información entre los participantes. Esta herramienta puede ser proporcionada como parte de un producto de software orientado a Workflow, o podría simplemente existir por si sola y tener integración con diferentes productos de Workflow.

3.3.2 Definición de Procesos

Luego del componente anterior se encuentra la Definición de Procesos, que forma parte de los datos del Workflow. Contiene, toda la información necesaria acerca de los procesos, incluye información de comienzo de actividades, condiciones, y reglas de navegación. Podría tener referencias a la definición de roles, donde se almacena información de la estructura organizacional. Esto quiere decir que en la definición de procesos se puede mencionar que en cierto proceso participa cierto rol, el cual está definido en la definición de roles.

3.3.3. Workflow Enactment Service

Este componente interpreta la descripción de procesos y controla las diferentes instancias de los procesos, secuencia de actividades, adiciona ítems (elementos) a la lista de trabajo de los usuarios (Worklist), e invoca aplicaciones necesarias. Todas estas tareas son hechas por uno o más motores de Workflow (engines), los cuales manejan la ejecución de las distintas instancias de varios procesos. Este software puede ser implementado como un sistema centralizado con un único motor de Workflow, 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 redundante en mayores beneficios.

3.3.4. Worklist (lista de trabajo)

La Worklist forma parte de los datos del Workflow y la puede apreciar en la parte inferior de la figura. Ya que la interacción con los usuarios es necesaria en algunos casos, el motor de Workflow utiliza una Worklist manejada por un manejador de Worklist para controlar tal interacción. El motor deposita en la Worklist ítems a ser ejecutados para cada usuario. La Worklist puede ser visible o invisible para los usuarios depende del caso, muchas veces se deja que el usuario seleccione ítems y los procese en forma individual.

3.3.5. Manejador de Worklist

Luego del componente anterior encontramos el Manejador de la Worklist. Es un componente de software el cual maneja la interacción entre los participantes del Workflow y el Workflow Enactment Service, vía la Worklist. El manejador soporta en general un amplio rango de interacción con otras aplicaciones clientes. En la figura la interface con el usuario es mostrada como una componente separada del manejador de Worklist. En algunos sistemas estas dos componentes están agrupadas como una única entidad funcional.

3.4. Invocación de Aplicaciones Externas a un SGWF según el Modelo de referencia de la WfMC

La WfMC ha estandarizado la automatización de los procesos de negocio. Dicho estándar define un marco genérico para la construcción de SGWF, permitiendo la interoperabilidad entre ellos y con otras aplicaciones. Entonces, el modelo de referencia de Workflow fue desarrollado desde estructuras genéricas de aplicaciones de Workflow, identificando las interfaces con estas estructuras, de forma de permitir a los productos comunicarse a distintos niveles. Todos los sistemas de Workflow contienen componentes genéricos que interactúan de forma definida. Para poder tener cierto nivel de interoperabilidad entre los diversos productos de Workflow, es necesario definir un conjunto de interfaces y formatos para el intercambio de datos entre dichas componentes.

En la figura 3.3 se muestran las distintas interfaces y componentes que se pueden encontrar en la arquitectura del Workflow.

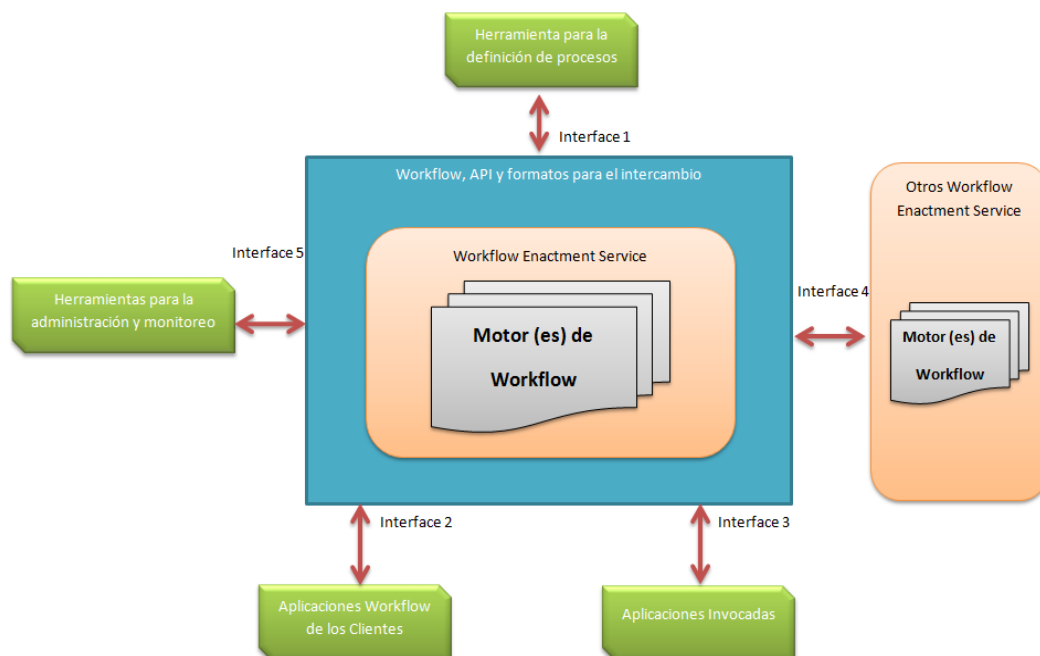


Figura 3.3. Arquitectura del Modelo de Referencia de SGWF.

En el modelo adoptado hay una separación entre los procesos y el control de la lógica de las actividades. Esta lógica está dentro de lo que ya definimos como el Workflow Enactment Service. Esta separación permite la integración de las diversas herramientas con una aplicación particular. La interacción del Enactment Service con los recursos externos se da por una de las dos interfaces siguientes:

- ✓ La interface de las Aplicaciones de los Clientes, a través de la cual el Motor de Workflow interactúa con el manejador de la Worklist, responsable de organizar el trabajo por intermedio de un recurso de usuario. Es responsabilidad del manejador del Worklist elegir y hacer progresar cada elemento de la lista de trabajo (Worklist).

- ✓ La interface de las Aplicaciones Invocadas, la cual le permite al motor de Workflow activar una herramienta para realizar una actividad particular. Esta interface podría ser basada en un servidor, es decir no existe la interacción con el usuario.

El Motor de Workflow (Workflow Engine) es el software que provee el control del ambiente de ejecución de una instancia de Workflow. Típicamente dicho software provee facilidades para:

- ✓ Interpretación de la definición de procesos.
- ✓ Control de las instancias de los procesos: creación, activación, terminación, etc.
- ✓ Navegación entre actividades.
- ✓ Soporte de interacción con el usuario.
- ✓ Pasaje de datos al usuario o a aplicaciones.
- ✓ Invocación de aplicaciones externas.

Por otra parte, las WAPI pueden ser vistas como un conjunto de llamadas API (Application Programming Interface) y funciones de intercambio de datos soportadas por el Workflow Enactment Service. Las APIs son un conjunto de llamadas a funciones de software que permiten a las aplicaciones acceder a funciones de un programa. Las WAPI permiten la interacción del Workflow Enactment Service con otros recursos y aplicaciones.

El intercambio de datos relevantes y datos de las aplicaciones, es requerido a través de las WAPI para soportar los 3 siguientes tipos de interacciones:

- ✓ Aplicaciones Workflow de los Clientes (interface 2) → ver anexo A.
- ✓ Aplicaciones Invocadas (interface 3).
- ✓ Intercambio entre los motores de Workflow (interface 4) → ver anexo A.

Aplicaciones Invocadas (Interface 3)

El componente Aplicaciones Invocadas representa software o aplicaciones ya existentes que un SGWF 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 Workflow, 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 Workflow 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 el motor del Workflow, 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 Workflow, 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 (es decir la dirección específica dentro de la red).

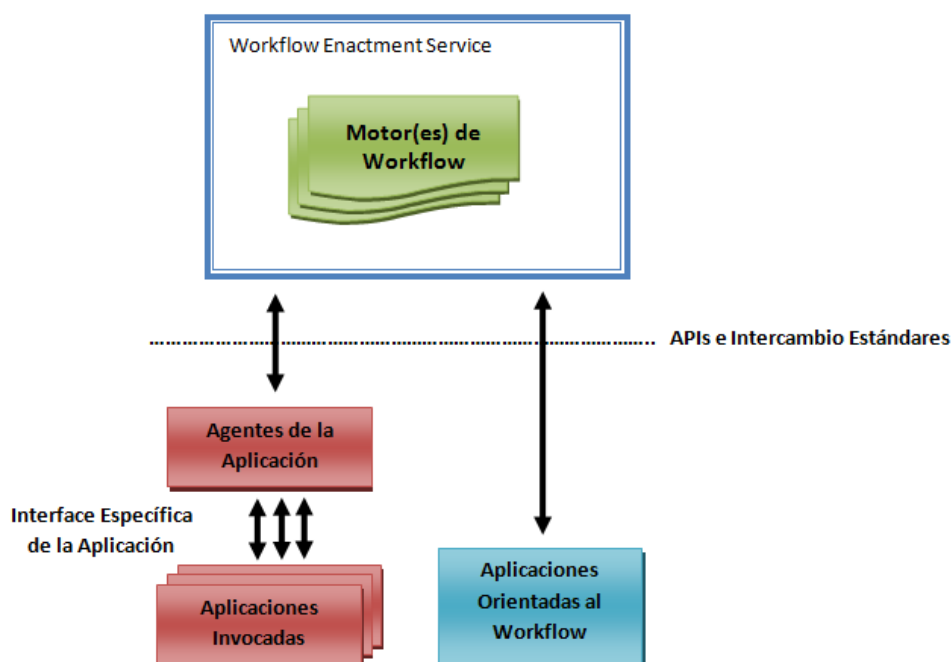


Figura 3.4. Componentes que participan en la Interfaz 3.

Hay SGWF que tratan con aplicaciones más restrictivas, es decir, aquellas donde los datos son fuertemente tipados y pueden estar directamente asociados con una herramienta particular, por ejemplo: Microsoft Word o Microsoft Excel. En otros casos la invocación de una operación puede lograrse a través de mecanismos de intercambio estándares. Se pueden desarrollar herramientas que usen un conjunto de APIs estándar para comunicarse con el Workflow. Estas APIs permiten la comunicación eficaz entre el motor del Workflow y aquellas aplicaciones que éste ha de ejecutar para que sirvan de soporte a las actividades correspondientes. En la tabla 3.1 se muestran las APIs asociadas a esta interfaz [WfMCb].

Categoría	APIs
Establecimiento de sesión	<ul style="list-style-type: none"> • Conexión/desconexión de una sesión de aplicación o agente de aplicación.
Funciones de Manipulación de Actividades	<ul style="list-style-type: none"> • 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)
	<ul style="list-style-type: none"> • consulta de atributos de la actividad (de la aplicación al motor)
Funciones de Manipulación de Datos	<ul style="list-style-type: none"> • provisión de datos relevantes del Workflow • provisión de datos de la aplicación

Tabla 3.1. APIs para la invocación de aplicaciones

La propuesta de este trabajo consiste de hacer uso de estos mecanismos de intercambio estándares con software o aplicaciones del tipo Web Service. La especificación de la Interfaz de Aplicaciones Invocadas con Web Services permitirá la distribución transparente de las aplicaciones externas. De esta manera, el motor del SGWF puede invocarlas sin la necesidad de conocer su ubicación exacta, con el beneficio de que las aplicaciones pueden cambiar su ubicación en la red sin que esto implique ningún cambio en su invocación. Al mismo tiempo, logrando que el SGWF interactúe con un Web Service, el motor se desliga de la costosa responsabilidad que la WfMC estipula a través de la especificación de la Interfaz 3. Esto es, implementar el proceso de búsqueda, selección e invocación de aplicaciones externas necesarias para cumplir con los requisitos del sistema.

CAPÍTULO 4: Web Services

En este capítulo se describen los Web Services, sus tecnologías, los lenguajes asociados a este tipo de aplicaciones y las estructuras subyacentes. También se introducen los conceptos correspondientes a las características de calidad y su inserción en el mundo de los Web Services.

La estructura del capítulo es la siguiente: en la sección 4.1 se introducen las nociones generales de los Web Services y las tecnologías subyacentes más relevantes. La sección 4.2 describe la arquitectura SOA que caracteriza a los Web Services y muestra los ejemplos de uso más conocidos. Luego en la sección 4.3 se presenta WSDL (Web Services Description Language), que es el lenguaje utilizado para especificar y describir los Web Services. En la sección 4.4 se presentan las características de calidad y como pueden encajarse en la especificación de los Web Services. Y finalmente la sección 4.5 muestra el registro mundial donde los Web Services son almacenados: UDDI (Universal Description Discovery and Integration).

4.1. Los Web Services

El término Web Service se utiliza muy a menudo hoy en día, aunque no siempre con el mismo significado. Los conceptos y tecnologías subyacentes de los Web Services son en gran medida independientes de la forma en que puedan ser interpretadas. Las definiciones existentes para el concepto de Web Services, van desde la muy genérica y con todo incluido a la muy específica y restrictiva.

A menudo, un Web Service es visto como una solicitud de acceso a otras aplicaciones a través de la Web, pero esta es una definición muy abierta, en virtud de la cual, cualquier cosa que tenga una dirección URL (Uniform Resource Locator, es decir localizador uniforme de recursos) sería un Web Service. Puede incluir, por ejemplo un script CGI¹, o también puede referirse a un programa accesible a través de la Web.

Entonces, se puede comenzar por definir a un Web Service como un estándar de comunicación entre procesos y/o componentes, diseñado para ser multiplataforma y multilenguaje, es decir, no importa en qué lenguaje esté programado un Web Service como ser Visual Basic, C# o java, o en qué plataforma esté corriendo, ya sea Windows, UNIX o Linux éstos serán accesibles y utilizables por otras aplicaciones desarrolladas en otras plataformas o lenguajes de programación. Es decir, diversas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los Web Services para intercambiar datos tanto en redes locales como en Internet. Esta interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones, esencialmente la W3C, son los comités responsables que se encargan de la arquitectura y la reglamentación de los Web Services.

Precisamente la W3C define a los Web Services como *“Aplicaciones de software identificadas mediante una URI (Uniform Resource Identifier), cuyas interfaces (y uso) son capaces de ser definidas, descritas y descubiertas mediante artefactos XML, y soportar interacciones directas con otras aplicaciones software usando mensajes basados en XML y protocolos basados en Internet”*.

Entonces, los Web Services pueden pensarse fundamentalmente como un cambio de las reglas del comercio Web. Conectan unas aplicaciones a otras a través de puntos distantes del mundo, transportando grandes cantidades de datos más eficientemente y más barato que nunca antes, utilizando un lenguaje en común. El resultado es una más rápida, mejor y más productiva comunicación tanto para los negociantes como los consumidores.

En el mundo de los SGWF, tanto los usuarios como los administradores de Workflow usualmente requieren 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 como obtener fechas, climas o cálculos matemáticos provistos desde la web.

¹ Interfaz de entrada común (en inglés Common Gateway Interface, abreviado CGI) es una importante tecnología de la World Wide Web que permite a un cliente (navegador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa.

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 SGWF 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. Por ende, en orden de permitir que el motor del SGWF invoque una aplicación externa sin necesidad exclusiva de conocer la ubicación de la misma, e incluso tampoco su nombre específico, esta tesis plantea una solución usando los Web Services.

Existen numerosas ventajas que pueden aprovecharse a partir de la utilización de los Web Services. Es que, explayando y precisando su definición, un Web Service es un componente abierto auto contenido y auto descripto que soporta la composición rápida de aplicaciones distribuidas. De ésta y las anteriores definiciones mencionadas, se desprende que uno de sus principales beneficios, es que permite a las aplicaciones que sean modulares y desacopladas, facilitando la reutilización de las mismas en distintas plataformas o lenguajes de programación. Por ello, los Web Services proveen un medio estándar de comunicación entre diferentes aplicaciones de software. Al tratarse de aplicaciones que pueden ser publicadas, localizadas e invocadas a través de la Web, su uso en la red global se ha extendido rápidamente. Esto se debe a la creciente necesidad de comunicación e interoperabilidad entre aplicaciones en la última década. Una vez desarrolladas, otras aplicaciones u otros Web Services, pueden descubrir e invocar el servicio determinado.

Hasta el surgimiento y posterior distribución de los Web Services, la reutilización de sistemas de componentes desarrollados y programados en diferentes proyectos, se limitaba a un lenguaje de programación determinado y/o a una plataforma en particular. Ahora, con el uso de los Web Services, los desarrolladores pueden solucionar este gran inconveniente. 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 está al alcance de cualquiera. De la misma forma que antes se incluían 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.

Por otra parte, los Web Services pueden ser agrupados según las funcionalidades que ofrecen en: de Información de negocios (pronóstico del tiempo, calendario, noticias, chequeo de crédito de una tarjeta, subastas, cotizaciones de acciones, planificación de vuelos, etc.); de Transacciones (reservas aéreas, acuerdos para rentar un auto, orden de compra, gestión de una cadena de suministro, etc.); o de Externalización de procesos de negocio (vínculos comerciales a nivel de Workflow, etc.). Esta catalogación por tipo se detalla en la sección 4.2.

Como lo enuncia la definición formal provista por la W3C, los Web Services se basan en estándares abiertos. Esta es la razón por la cual permiten a las aplicaciones comunicarse más libremente e independientemente de las plataformas en que se están ejecutando. Otra razón, y característica fundamental de los Web Services, es que

se describen en términos de sus operaciones y de los mensajes de entrada y salida de cada una de esas operaciones que provee. Tales definiciones se expresan en el lenguaje de marcado extensible XML (Extensible Markup Language) [XML], usando un lenguaje de descripción de Web Services: WSDL (Web Service Description Language) [WSDL2.0-0] [WSDL2.0-1]. La noción de describir un Web Service 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. Este lenguaje se retoma y detalla en la sección 4.3.

Así, el uso de protocolos estándares como WSDL en el contexto de los Web Services es necesario sin excepciones para poder lograr la interoperabilidad en estos ambientes heterogéneos, con independencia del sistema operativo, el lenguaje de programación, etc. Otros estándares que se utilizan al hablar de Web Services son:

- ✓ Protocolo SOAP (Simple Object Access Protocol) [SOAP]: Se describe detalladamente en el anexo B. Se utiliza para la representación de los mensajes.
- ✓ Protocolo HTTP (Hiper Text Transport Protocol) [HTTP]: Se utiliza para el transporte de los mensajes. Si bien es de público conocimiento, se describe brevemente también en el anexo B.
- ✓ UDDI (Universal Description, Discovery and Integration) [UDDI]: Se utiliza para el registro y la localización de cualquier Web Service. Se trata de un directorio o catálogo de Web Services distribuido que permite que se listen, busquen y descubran este tipo de aplicaciones de software. Se trata cuidadosamente en la sección 4.5.

Cada uno de estos protocolos se asocian a una capa, las cuales en conjunto, definen la infraestructura de los Web Services. La Figura 4.1 muestra justamente, la separación por capas que define esta infraestructura [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 Web Services.

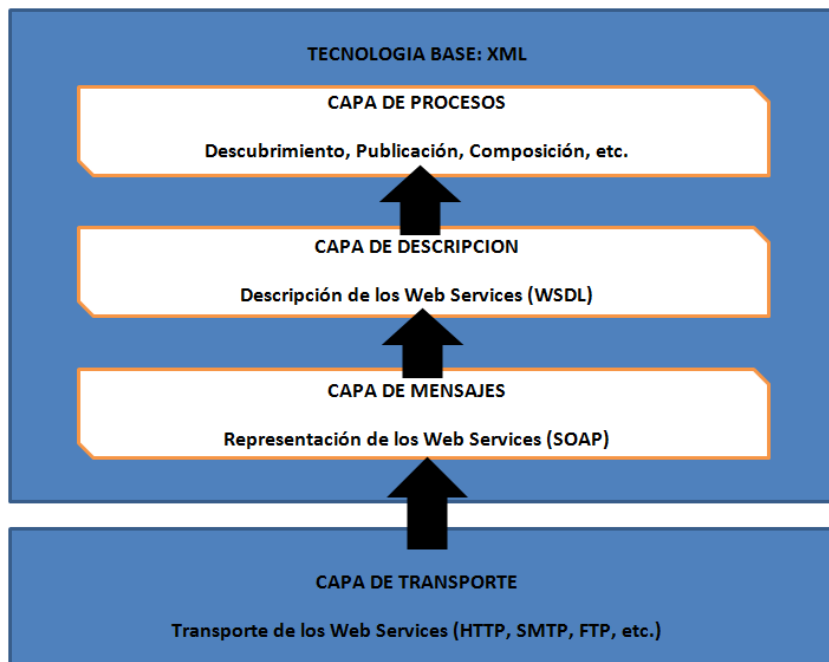


Figura 4.1. Capas y estándares de los Web Services

En general, los Web Services 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. Sin embargo, la mayoría de ellos proveen un método de acceso gráfico disponible para la vista de un usuario persona, que no es más que una página web perceptible gracias a los navegadores web. Incluso una gran cantidad de Web Services proporcionan su descripción WSDL. Sin embargo, el navegador deberá tener instalados los componentes o extensiones correspondientes en orden de que la persona pueda comprender su sintaxis.

Si bien estos tipos de aplicaciones de software, basados en estándares, pueden variar en la complejidad de las funcionalidades que ofrecen, es decir funcionalidades que van desde operaciones simples, tales como ejecutar una suma de dos números enteros, a sistemas complejos tales como la reserva online de un viaje aéreo, siempre persiguen los mismos objetivos para poder cumplir con cualquier requerimiento de diferentes usuarios [Labra06]. Los objetivos de los Web Services se resumen en la tabla 4.1.

Modularidad y Reusabilidad	Los desarrolladores dentro de una organización y entre organizaciones pueden tomar el código desarrollado para las aplicaciones existentes, exponerlos como Web Services, y luego reutilizarlos para atender nuevos requerimientos de negocio. Esto implica ahorro en el costo y en el tiempo de desarrollo.
Independencia del lenguaje y de la plataforma	Separación entre la especificación y la implementación en el que se desarrolle el Web Service. Usando: XML, SOAP,

	WSDL, UDDI, tecnologías estándares.
Interoperabilidad	Los clientes y los servicios se comunican y se entienden entre sí, sin importar sobre qué plataforma se ejecutan. Los Web Services alcanzan este objetivo sólo si ellos y sus clientes tienen una forma estándar de comunicación entre sí, que sea consistente a través de plataformas, sistemas, y lenguajes. Es por ello que los Web Services están siendo muy aceptados y, por ende, ampliamente utilizados.
Escalabilidad	Las aplicaciones que utilizan Web Services 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.
Acoplamiento débil	Un servicio asincrónico ejecuta su procesamiento sin forzar al cliente a permanecer esperando hasta que termine el procesamiento. Por otro lado, un servicio sincrónico fuerza al cliente a que espere. Entonces, la 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.
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 de los Web Services,

	permite que las aplicaciones resulten flexibles y fáciles de evolucionar conforme cambian los requerimientos.
--	---

Tabla 4.1. Objetivos de los Web Services.

Entonces, se puede concluir que el objetivo fundamental de los Web Services es permitir que diferentes aplicaciones, realizadas con diferentes tecnologías, y ejecutándose en una variedad de entornos, puedan comunicarse e integrarse con suma normalidad y sencillez. También, la reutilización de desarrollos sin importar la plataforma en que funcionan o el lenguaje en el que estén escritos. Así, los Web Services se establecen como una nueva capa de aplicaciones de software de forma tal que pueden interactuar entre ellos usando, por una parte tecnologías estándares para comunicarse, y por otro lado el contexto de Internet como medio de transporte.

4.2. Tipos y Arquitectura de los Web Services

A continuación se presentan algunos ejemplos de uso de Web Services, acaso los de mayor conocimiento y distribución en la Web, cada uno discriminado por su tipo. Luego, a partir de ellos, se describirá la arquitectura subyacente sobre la cual se constituyen estas aplicaciones.

4.2.1. Información del Negocio (Business Information)

Estos Web Services 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 Web Service. Sólo la petición y recepción del resultado del lado del usuario. Algunos ejemplos, acaso los más conocidos, son:

- ✚ *Estado del tiempo:* un Web Service 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 o semana.
- ✚ *Calendario:* un Web Service 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 Web Services 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 Web Service 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.

4.2.2. Transaccional (Transactional) para B2B o B2C

Estos Web Services permiten la interacción de un usuario o empresa con uno o varios Web Services, para conseguir un resultado de valor (como reservas en líneas aéreas, alquiler de autos, orden de compra de un producto, etc.). Dichos Web Services 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. Estos Web Services pueden ser sobre:

- ✚ *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 Web Services 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 Web Service que permite acordar el período de tiempo, fecha de retiro y devolución del vehículo, vehículo elegido, etc.
- ✚ *Órdenes de compra:* un Web Service 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.

4.2.3. Externalización de procesos de negocio

No solo promover la integración de aplicaciones o sistemas es el objetivo de los Web Services, también apuntan 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 Web Service, al incorporarse a la red global, permitan a diferentes organizaciones el poder diseñar sus aplicaciones con foco en sus propias actividades de negocio y contratar servicios externos especializados cuya finalidad es interoperar con otras compañías. Así, las empresas establecen vínculos comerciales a un nivel de Workflow con otras empresas, lo cual permite una total integración a nivel de los procesos de negocio.

4.2.4. Arquitectura SOA

Los ejemplos provistos en las subsecciones previas, comparten una misma arquitectura. Es la Arquitectura Orientada a Servicios (Service Oriented Architecture) [SOA], que representa una manera de manipular y administrar un conjunto de Web Services a través del uso de SOAP, WSDL y UDDI.

SOA ve a los Web Services como unidades lógicas, definidas en términos de la funcionalidad (operaciones) que proveen y que están descritas por documentos WSDL que sólo proporcionan los detalles necesarios para su invocación, los tipos de datos que manejan, las operaciones y sus parámetros, los protocolos que soportan y la ubicación correspondiente. Dicha invocación se realiza mediante el intercambio de mensajes de acuerdo a los estándares antes mencionados.

SOA se basa en las interacciones entre el proveedor, las agencias de descubrimiento y el solicitante como muestra la 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 de cada Web Service, 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.

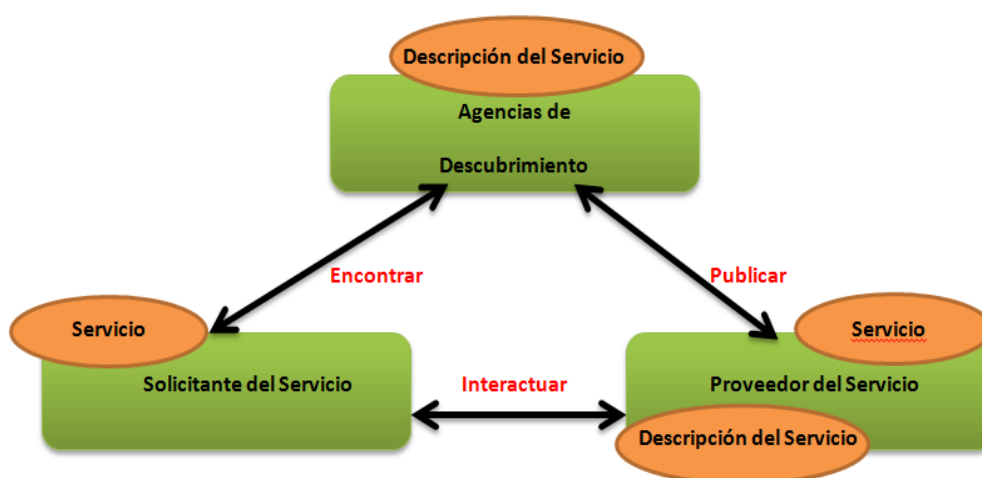


Figura 4.2. Arquitectura SOA

La tabla 4.2 describe la figura 4.2:

ROLES	COMPONENTES	OPERACIONES
<u>Proveedor del Servicio</u>	<u>Servicio</u>	<u>Publicar</u>
Es el dueño del servicio. Es la plataforma que permite el acceso al servicio. Por ejemplo: Un servidor de	Es un módulo de software desarrollado sobre plataformas accesibles desde la web, y provisto	Para poder ser accesible desde la web, un Web Service necesita publicar su descripción, de manera tal

aplicaciones.	por un proveedor de servicios. El mismo es invocado o interactúa con el solicitante del servicio. También puede funcionar como un solicitante.	que el solicitante puede encontrarlo. Lo que se publica es la descripción del servicio.
<u>Solicitante del Servicio</u>	<u>Descripción del Servicio</u>	<u>Encontrar</u>
Es quien requiere cierta funcionalidad que ofrece el servicio. Es la aplicación que está buscando e iniciando una interacción con el servicio. Por ejemplo: Un browser conducido por una persona o un programa sin una interface de usuario (otro Web Service).	Contiene los detalles de la interface e implementación del Web Service. Incluye los tipos de datos, las operaciones, la información del tipo de enlace (binding) y la ubicación en la red. Su consta de un conjunto de documentos de descripción XML. La descripción del servicio puede publicarse directamente por un proveedor o por una Agencia de Descubrimiento.	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.
<u>Agencia de Descubrimiento</u>	----	<u>Interactuar</u>
Es un conjunto de descripciones de Web Service donde los proveedores publican sus descripciones. Puede ser distribuida o centralizada. Los solicitantes encuentran los servicios y obtienen información del enlace (binding), para realizar la invocación.		En tiempo de ejecución, el que requiere el servicio inicia una interacción al invocar el Web Service, haciendo uso de la descripción para la ubicación, contacto e invocación del servicio.

Tabla 4.2. Roles, componentes y operaciones de la Arquitectura SOA

Finalmente, a continuación se describe y grafica como se desarrolla una interacción entre un cliente y el Web Service requerido en un escenario típico. Es decir, la dinámica de los Web Services:

- I. El Cliente se comunica con el Registro o Servidor UDDI para encontrar un Web Service.

- II. El Registro o Servidor UDDI le indica al cliente un documento WSDL (descripción del Web Service buscado).
- III. El Cliente accede al documento WSDL.
- IV. El fichero WSDL le provee lo necesario para interactuar con el Web Service.
- V. El cliente envía un requerimiento usando SOAP o HTTP al servidor de aplicaciones que aloja el Web Service.
- VI. El Web Service retorna una respuesta SOAP.

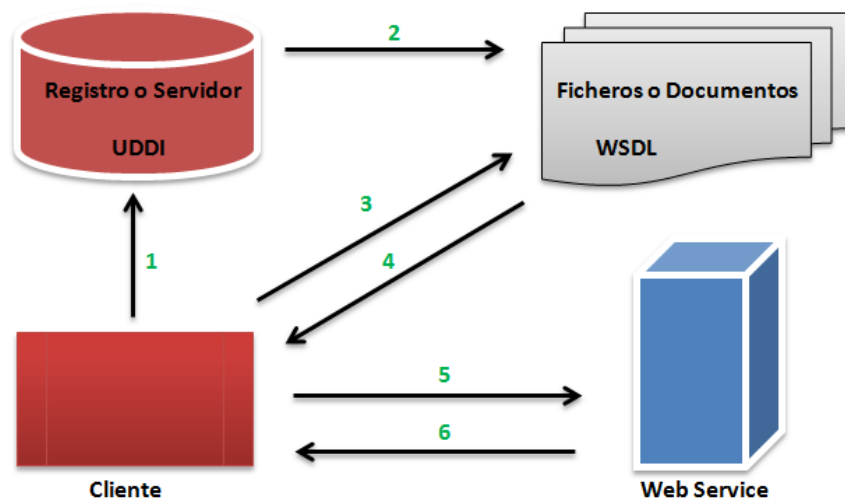


Figura 4.3. Dinámica de los Web Services

Si bien la interacción que muestra la figura 4.3 se retoma y describe en detalle en el próximo capítulo, pero ya aplicada a un caso de estudio, puede apreciarse que 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 Web Service, que indique su funcionalidad. Así los posibles clientes al obtener e interpretar esta descripción, sabrán cómo invocar el Web Service y qué esperar como resultado. Lo interesante de las tecnologías que usan XML (WSDL), SOAP, HTTP y UDDI, es que pueden integrarse en aplicaciones ya existentes sin demasiada complejidad.

4.3. El lenguaje WSDL

WSDL (Web Service Description Language) es el lenguaje de especificación utilizado para definir los Web Services. Se trata de un documento XML que se utiliza para describir los mensajes SOAP y cómo estos mensajes son intercambiados. Es decir, supongamos que creamos un Web Services y queremos que otras aplicaciones lo utilicen, las otras aplicaciones deberán acceder a un documento WSDL en donde podrán conocer los métodos que expone nuestro Web Service y cómo acceder a ellos, es decir, cuáles son los nombres de los métodos y qué tipos de parámetros espera cada uno de ellos. Entonces, en este sentido, cualquier documento WSDL permite

proveer información acerca de la ubicación de cierto Web Service y su interfaz. Así, con esta información respecto a la descripción y especificación del Web Service, el usuario sabrá básicamente cómo llevar a cabo la interacción con el servicio.

Hay que tener en cuenta que todo documento WSDL separa la descripción de un servicio en dos partes [WSA]. Dentro de cada una de estas partes, 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. La figura 4.4 muestra estos elementos y los componentes que se definen en cada uno de ellos.

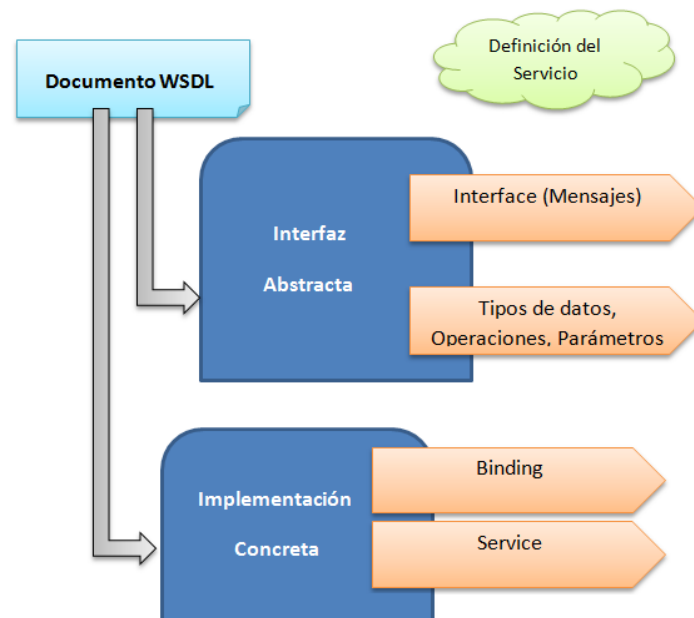


Figura 4.4. Elementos de documentos WSDL – Definición del Web Service.

Por un lado, en la Interfaz Abstracta los dos elementos a definir son:

- 1) **types:** provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar, sus operaciones los parámetros de entrada o salida de cada operación.
- 2) **interface:** describe la secuencia de mensajes que el servicio envía o recibe.

Y por el otro, los dos elementos a definir en la implementación concreta son:

- 1) **binding:** define los detalles de implementación concretos del servicio.
- 2) **service:** se usa para asociar un binding con el URL donde está realmente el servicio (endpoint).

Entonces cualquier documento WSDL especifica tanto la definición como una descripción del Web Service que representa. Pero también define los dos componentes principales de estos, es decir, sus características funcionales y sus

características no funcionales [Papazoglou08]. La descripción funcional se refiere a las características operacionales que definen el comportamiento general de un Web Service. En otras palabras, 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. Y el otro componente, la descripción no funcional, se concentra en los atributos o características de calidad (Quality of Service - QoS) del Web Service.

Ambos componentes, es decir, tanto las características funcionales como las no funcionales, se retoman e implementan en un caso de estudio particular en el próximo capítulo.

4.4 Características de Calidad (QoS) de los Web Services

Como ya ha sido expresado en este trabajo, el incremento del número de Web Services que proveen similares funcionalidades evidencia la importante necesidad de encontrar el mejor Web Service que satisfaga ciertas necesidades del solicitante del servicio. Es por eso que la especificación de las características de calidad ofrecidas por un Web Service se convierte en un tema de alta prioridad tanto para el proveedor como para sus clientes. La descripción no funcional en un documento WSDL de un Web Service se concentra en describir estas características de calidad del servicio.

Por otro lado y en orden de establecer claras diferencias, se recuerda que la descripción sintáctica del Web Service focaliza en los aspectos funcionales del mismo. Ahora bien, dado que un requerimiento de las aplicaciones basadas en SOA es operar de forma tal que resulten fiables y reutilizables, los requerimientos de un Web Service no deben focalizar solamente en las propiedades funcionales del mismo, sino que también se deben concentrar en describir el ambiente en el que se desarrolla, 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 como: disponibilidad, performance y escalabilidad, políticas de seguridad y privacidad, etc., todas las cuales deben ser descriptas. La Tabla 4.2 detalla los elementos clave que se tienen en cuenta para soportar las características de calidad en el mundo de los Web Services. En particular, se tienen en cuenta las características relacionadas al tiempo de ejecución del servicio:

Característica de Calidad (QoS)	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.	
	<table> <tr> <td>Rendimiento</td><td>Número de requerimientos de Web Services atendidos en un período de tiempo dado.</td></tr> </table>	Rendimiento
Rendimiento	Número de requerimientos de Web Services 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 Web Service para procesar su secuencia de actividades.
	Tiempo de Transacción	Representa el tiempo que transcurre mientras el Web Service está completando una transacción.
Disponibilidad		Probabilidad de que el Web Service esté 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 4.2. Características de calidad de los Web Services

De esta forma las características de calidad de un Web Service se refieren a la habilidad del servicio de responder a las invocaciones que sufra y llevarlas a cabo en consonancia con las expectativas tanto del proveedor como de los clientes [Papazoglou08]. Ahora bien, los factores de calidad que suelen reflejar las expectativas del cliente en condiciones naturales son la disponibilidad, la conectividad o la alta respuesta de un servicio. Estos factores resultan ser claves para mantener un negocio competitivo y viable. Así, las características 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 Web Services. Como consecuencia, tener en cuenta las características de calidad de un Web Service es un desafío tan crítico como significativo en el ambiente dinámico 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 estos atributos de calidad de un Web Service.

Tradicionalmente, las QoS se miden por el grado en el cual las aplicaciones, sistemas, redes y otros elementos de Internet soportan la disponibilidad de los servicios a un nivel requerido de performance, en todos los accesos y condiciones de descarga. En el contexto de los Web Services, 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 propone una forma de incorporar las características de calidad en los Web Services a partir de la necesidad de tenerlas en cuenta a la hora de optimizar el proceso de selección del Web Service más adecuado. Esta incorporación ofrece importantes beneficios, tanto para los usuarios como los proveedores. A los usuarios les permite expresar sus necesidades y utilizar el Web Service que mejor se ajusta sus necesidades, mientras que los proveedores pueden publicar mejor las capacidades de sus servicios y lucirse con las mismas.

4.5. Universal Description Discovery and Integration (UDDI)

Hasta ahora, se ha explicado que es y de que consta un Web Service con algunas situaciones reales o ejemplos, y describiendo las tecnologías subyacentes que ellos involucran. Lo que esta sección trata, es definir cómo se dará a conocer el Web Service para que los clientes interesados puedan descubrirlo fácilmente y utilizarlo en sus aplicaciones. En la actualidad, ya existe un mecanismo de descubrimiento que cumple estos requisitos: UDDI (Universal Description Discovery and Integration), una iniciativa del sector para hacer compatible el descubrimiento de Web Services con todo tipo de tecnologías y plataformas.

4.5.1 UDDI: Un registro global de Web Services

UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios. Se puede utilizar sistemas taxonómicos estándar para clasificar estos datos y poder encontrarlos posteriormente en función de la categorización. Lo más importante es que UDDI contiene información sobre las interfaces técnicas de los servicios de una empresa. A través de un conjunto de llamadas a API XML basadas en SOAP, se puede interactuar con UDDI tanto en tiempo de diseño como de ejecución para descubrir datos técnicos de los servicios que permitan invocarlos y utilizarlos. De este modo, UDDI sirve como infraestructura para una colección de software basado en Web Services.

La justificación respecto a la necesidad de contar con un registro de este tipo, se encuentra fácilmente si se tiene en cuenta que existe una colección de software de miles (quizás millones) de Web Services. Ahora bien, se plantean varias cuestiones difíciles de resolver:

- ✓ ¿Cómo se descubren los Web Services?
- ✓ ¿Cómo se categoriza la información de forma coherente?
- ✓ ¿Cómo repercute esto en la localización?
- ✓ ¿Cómo afecta a las tecnologías de propietario? ¿Cómo se puede garantizar la interoperabilidad del mecanismo de descubrimiento?

- ✓ ¿Cómo se puede interactuar en tiempo de ejecución con este mecanismo de descubrimiento cuando mi aplicación depende de un Web Service?

La iniciativa UDDI surgió como respuesta a estas preguntas. Varias empresas, incluidas Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP y unas trescientas más (para obtener un listado completo, consulte UDDI: Community [en inglés]), unieron sus esfuerzos para desarrollar una especificación basada en estándares abiertos y tecnologías no propietarias que permitiera resolver los retos anteriores. El resultado, cuya versión beta se lanzó en diciembre de 2000 y estaba en producción en mayo de 2001, fue un registro empresarial global alojado por varios nodos de operadores en el que los usuarios podían realizar búsquedas y publicaciones sin coste alguno.

A partir de la creación de esta infraestructura para Web Services, los datos sobre estos servicios se pueden encontrar de forma sistemática y confiable en una capacidad universal totalmente independiente de proveedores. Se pueden llevar a cabo búsquedas categóricas precisas utilizando sistemas de identificación y taxonómicos extensibles. La integración de UDDI en tiempo de ejecución se puede incorporar a las aplicaciones. Como resultado, se fomenta el desarrollo de un entorno de software de Web Services. El funcionamiento de UDDI es el siguiente:

La información de UDDI se aloja en nodos de operador, empresas que se han comprometido a ejecutar un nodo público conforme a la especificación que rige el consorcio UDDI.org. En la actualidad existen dos nodos públicos que se ajustan a la versión 1 de la especificación UDDI: Microsoft aloja uno e IBM el otro. Hewlett Packard se ha comprometido a alojar un nodo bajo la versión 2 de la especificación. Los operadores del host deben replicar datos entre ellos a través de un canal seguro, para conseguir la redundancia de la información en el registro UDDI. Se pueden publicar los datos en un nodo y descubrirlos en otro tras la réplica. Actualmente, la réplica se produce cada 24 horas. En el futuro, este intervalo entre réplicas se reducirá, ya que habrá más aplicaciones que dependan de los datos de UDDI.

Resulta importante observar que no existen requisitos de propietario respecto al modo en que el operador del host implementa su nodo. El nodo sólo se debe ajustar a la especificación UDDI. El nodo de Microsoft (<http://uddi.microsoft.com/default.aspx> [en inglés]), por ejemplo, se ha escrito por completo en C# y se ejecuta en producción en tiempo de ejecución en lenguaje común .NET Beta 2. El código de base se beneficia claramente de la compatibilidad nativa con SOAP y de la serialización que ofrecen las clases de sistema .NET. En el lado del servidor, el nodo del operador Microsoft utiliza Microsoft® SQL Server 2000 como almacén de datos. IBM, en cambio, utiliza tecnologías completamente diferentes para ejecutar su nodo. No obstante, los dos nodos se comportan exactamente igual, ya que se ajustan al mismo conjunto de llamadas a API XML basadas en SOAP. Las herramientas de los clientes pueden interoperar con ambos nodos sin problemas. Por eso, el nodo público UDDI constituye un claro ejemplo de que el modelo de Web Services XML funciona en entornos heterogéneos.

El próximo paso para comprender la iniciativa UDDI consiste en ver qué datos se almacenan en UDDI y cómo se estructuran. UDDI es relativamente ligero; se ha diseñado como registro, no como depósito. La diferencia, aunque sutil, resulta

esencial. Un registro redirige al usuario a recursos, mientras que un depósito sólo almacena información. El registro Microsoft® Windows® puede servir de ejemplo: contiene las configuraciones y parámetros básicos pero, en última instancia, su función es la de dirigir la aplicación a un recurso o binario. Buscar un componente COM basándose en su identificador de programa conducirá a un identificador de clase, que a su vez dirigirá a la ubicación del binario.

UDDI se comporta de forma similar: como el registro de Windows, se basa en identificadores únicos globales (GUID) para garantizar la capacidad de búsquedas y determinar la ubicación de recursos. En última instancia, las consultas a UDDI conducen a una interfaz (un archivo .WSDL, .XSD, .DTD, etc.) o a una implementación (como un archivo .ASMX o .ASP) ubicadas en otro servidor. Por tanto, UDDI responde a cualquier usuario interesado en sus registros a preguntas del estilo de:

- ¿Qué interfaces de Web Services basadas en WSDL se han publicado y establecido para un sector determinado?
- ¿Qué empresas han escrito una implementación basada en una de estas interfaces?
- ¿Qué Web Services, categorizados de algún modo, se ofrecen actualmente?
- ¿Qué Web Services ofrece una empresa determinada?
- ¿Con quién se debe poner en contacto el usuario para utilizar los Web Services de una empresa?
- ¿Cuáles son los detalles de implementación de un Web Service concreto?

4.5.2 WSDL y UDDI

WSDL se ha convertido en una pieza clave de la pila de protocolos de los Web Services. Por eso, es importante saber cómo colaboran UDDI y WSDL y por qué la idea de interfaces frente implementaciones forma parte de cada protocolo. WSDL y UDDI se diseñaron para diferenciar claramente los metadatos abstractos y las implementaciones concretas. Para entender cómo funcionan WSDL y UDDI resulta esencial comprender las consecuencias de esta división.

Por ejemplo, WSDL distingue claramente los mensajes de los puertos: los mensajes (la sintaxis y semántica que necesita un Web Service) son siempre abstractos, mientras que los puertos (las direcciones de red en las que se invoca al Web Service) son siempre concretos. No es necesario que un archivo WSDL incluya información sobre el puerto. Un archivo WSDL puede contener simplemente información abstracta de interfaz, sin facilitar datos de implementación concretos, y ser válido. De este modo, los archivos WSDL se separan de las implementaciones.

Una de las consecuencias más interesantes de esto es que pueden existir varias implementaciones de una única interfaz WSDL. Este diseño permite que sistemas dispares escriban implementaciones de la misma interfaz, para garantizar así la comunicación entre ellos. Si tres empresas diferentes implementan el mismo archivo WSDL y una parte del software de cliente crea el código auxiliar/proxy a partir de esa interfaz, dicho software se podrá comunicar con las tres implementaciones con el mismo código de base, cambiando simplemente el punto de acceso.

UDDI establece una distinción similar entre la abstracción y la implementación con el concepto de tModels. La estructura tModel, abreviatura de "Technology Model" (modelo de tecnología), representa huellas digitales técnicas, interfaces y tipos abstractos de metadatos. El resultado de los tModels son las plantillas de enlace, que son la implementación concreta de uno o más tModels. Dentro de una plantilla de enlace se registra el punto de acceso de una implementación particular de un tModel. Del mismo modo que el esquema de WSDL permite separar la interfaz y la implementación, UDDI ofrece un mecanismo que permite publicar por separado los tModels de las plantillas de enlace que hacen referencia a ellos. Por ejemplo, un grupo industrial o de estándares publica la interfaz canónica para un sector particular y, a continuación, varias empresas escriben implementaciones de esta interfaz. Obviamente, cada una de estas implementaciones haría referencia al mismo tModel. Los archivos WSDL constituyen un ejemplo perfecto de tModel de UDDI.

CAPÍTULO 5: GG-WSfinder

Este capítulo presenta la herramienta *GG-WSfinder*. Una aplicación Web Service que encuentra, ejecuta y devuelve el resultado de un Web Service, disponible en un servidor UDDI, que cumple con ciertos requerimientos especificados.

La estructura de este capítulo es la siguiente: la sección 5.1 describe el contexto en el que la herramienta se desarrolla. Por su parte, en la sección 5.2 se muestra el diseño general de la herramienta. La sección 5.3 presenta el caso de estudio elegido, y finalmente, la sección 5.4, enumera las distintas herramientas que se utilizaron en el desarrollo.

5.1 Contexto de la herramienta

5.1.1. Introducción al problema

En [Martellotto10] se plantea la especificación funcional de la Interfaz de Aplicaciones Invocadas de un Workflow utilizando Web Services. El objetivo es mejorar la selección de aplicaciones en tiempo de ejecución incorporando la especificación no funcional de esta interfaz. Esto posibilita al motor de Workflow invocarlas independizándose de la ubicación exacta de las mismas. En la práctica, esta especificación se traduce en que cada motor de Workflow tenga la responsabilidad de realizar la búsqueda y selección de la mejor aplicación disponible y ejecutar la misma.

Este trabajo plantea la realización de un Web Service que se ocupe de ello. Es decir, un Web Service que pueda ser invocado por cualquier Workflow, de manera que, dados el tipo de la aplicación que se necesite invocar, los requerimientos del sistema y ciertas normas de calidad, encuentre, ejecute y retorne el resultado de esa ejecución de la aplicación. Es decir, el Web Service disponible más adecuado que cumpla con los requerimientos especificados. De esta manera cualquier implementación de un SGWF se desliga de este asunto.

La esencia de implementar una aplicación del tipo Web Service, radica puntualmente en **la automaticidad que este proporciona en orden de que pueda ser invocado por cualquier otro tipo de aplicación, en particular un Workflow.** Sin embargo, esta automatización es efectivamente aprovechada siempre y cuando se sepa con anticipación y precisión que Web Service se desea invocar, es decir, tanto su nombre como su ubicación en la red, entre otros atributos. Esto no implica necesariamente un problema de magnitud considerable cuando el factor humano está presente. Cualquier buscador de Web Services provee dicha información de los mismos. Así, por ejemplo, hoy en día el administrador de Workflow puede buscar la aplicación que desee, probarla para asegurarse de que esta le provee realmente lo que busca y luego sí, como ya se ha mencionado anteriormente, configurar al Workflow o SGWF (en la actualidad estáticamente) para que realice la invocación necesaria.

Para lograr que el Web Service planteado en este trabajo realice todo este proceso satisfactoriamente, es necesario mecanizarlo, tanto en la interoperabilidad respecto a la búsqueda y selección de aplicaciones (otros Web Services), como en garantizar que el resultado del proceso completo sea exitoso.

Ahora bien, la descripción sintáctica de un Web Service en el protocolo UDDI, focaliza en los aspectos funcionales del mismo, sin contener ninguna descripción de los atributos de calidad. Con el incremento del número de Web Services que proveen similares funcionalidades, es importante poner más énfasis en encontrar el mejor Web Service que satisface las necesidades de quien lo solicite, es decir el usuario. Incluyendo tanto sus requerimientos funcionales como los no funcionales. Las descripciones no funcionales fuerzan entonces al usuario del servicio, a especificar en tiempo de ejecución los atributos de calidad que pueden influir en la elección de un Web Service ofrecido por un proveedor.

Al mismo tiempo, 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 ejecución del

servicio. Los requerimientos no funcionales, es decir, las características de calidad, no son soportadas por los registros actuales de UDDI [Ran03]. Por ello, parte de esta propuesta no solo consiste en incorporar al proceso de selección de Web Services el análisis de los atributos de calidad requeridos por el usuario o solicitante, sino que también dar soporte al descubrimiento de servicios previo al tiempo de ejecución.

5.1.2. Descripción general de la herramienta GG-WSfinder

En virtud de lograr el cumplimiento de los objetivos enunciados en la sección 1.3, y solucionando al mismo tiempo los problemas planteados en la sección 5.1.1, una gran cantidad de factores han sido tenidos en cuenta para finalmente llegar a una solución acorde a las expectativas.

GG-WSfinder utiliza dos plantillas genéricas. La primera es de entrada. Consiste de un conjunto de parámetros que son los encargados de transportar los requerimientos del usuario a la herramienta. Y dado que este usuario será en reiteradas ocasiones un SGWF, el conjunto de parámetros ha sido planteado simple y estándar. Esta plantilla está formada por los siguientes atributos: tipo de Web Service, el o los nombres posibles del mismo, nombres de la operación a ejecutar con sus respectivos parámetros (valores para los cuales se deseen obtener los resultados correspondientes) y las normas de calidad que el Web Service debe cumplir. Esta parametrización le posibilitará a la herramienta acceder de manera estándar a los requerimientos del sistema siempre de una misma forma, para luego comenzar el proceso de búsqueda con los criterios y filtros correspondientes.



Figura 5.1. Interacción entre la herramienta y las aplicaciones cliente.

La salida de la herramienta depende del tipo de Web Service seleccionado. Para cada uno de estos tipos se define una plantilla de salida que consiste del conjunto de atributos necesarios para poder retornar el resultado obtenido de la ejecución. De esta manera, para un mismo caso, es decir un mismo tipo de Web Services, no importa que aplicación provea el resultado, siempre el retorno de la herramienta será una instancia de la plantilla.

La figura 5.1 muestra el sentido en que es llevada a cabo la interacción entre la herramienta GG-WSfinder y las aplicaciones cliente externas a través del uso de las plantillas descritas.

Es importante destacar, que la herramienta provee al usuario las diferentes alternativas que desee, en cuanto al tipo de Web Service (con su respectiva plantilla de salida) y las características de calidad que implementa.

5.1.2.1. Proceso de selección de Web Services

El proceso de selección de Web Services con sus respectivas ejecuciones, consiste en la transformación de los requerimientos del usuario en una instancia de la plantilla de salida. Este proceso lo realiza el motor de la herramienta (figura 5.2) y se describe en los siguientes puntos:

Fase 1. Obtención de Parámetros:

En esta fase la herramienta obtiene desde la plantilla de entrada los parámetros. En la figura 5.2, los números del mensaje de entrada se corresponden a estos parámetros. Ellos significan: 1 = tipo del Web Service; 2 = nombre del Web Service; 3 = nombre de las operaciones; 4 = nombre de los parámetros; 5 = característica de calidad.

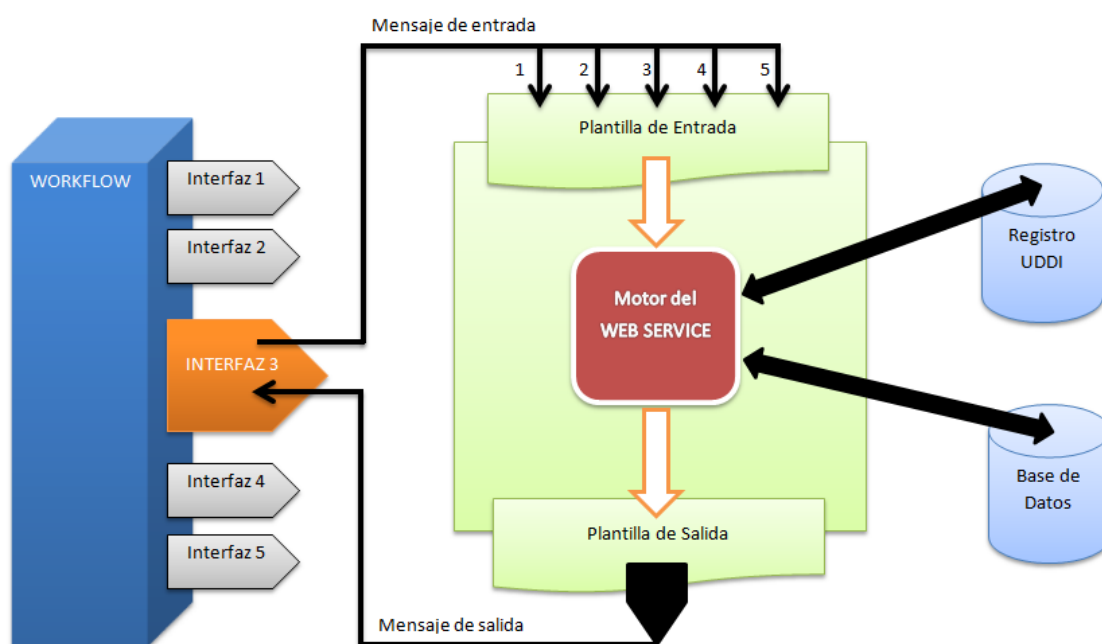


Figura 5.2. Modelo genérico de la herramienta GG-WSfinder.

Fase 2. Obtención de Web Services de la base de datos:

La base de datos representa el historial de los Web Services alguna vez ejecutados por la herramienta. Se compone de la toda información de estos Web Services (desde su nombre hasta sus parámetros).

En esta fase la herramienta, en base al tipo y las características de calidad ingresadas por el usuario en la plantilla de entrada, obtiene de la base de datos la lista de descripciones de los Web Services.

Fase 3. Filtro por nombres de Web Service:

A partir del conjunto de Web Services generado en la fase previa, la herramienta procede a filtrarlo teniendo en cuenta los nombres que se especificaron en la plantilla de entrada. Para el grupo de Web Services que pasan con éxito el filtro, la herramienta obtendrá de la base de datos las operaciones y parámetros correspondientes de cada uno.

Fase 4. Filtro por nombres de operaciones y parámetros:

En esta fase la herramienta realiza un nuevo filtro sobre el conjunto de Web Services, pero considerando los nombres de operaciones y la cantidad de parámetros introducidos por el usuario en la plantilla de entrada.

Fase 5. Ejecución de Web Services:

En la ejecución de Web Services se selecciona el primer Web Service del conjunto. Una ejecución consiste en la invocación del Web Service con los parámetros que el usuario pasa en la plantilla de entrada. Si el resultado de esta ejecución es *exitoso*, la herramienta retorna la plantilla de salida cuyos datos son los obtenidos de la ejecución.

Una ejecución se considera *exitosa* cuando los datos obtenidos de la invocación del Web Service completan el conjunto de atributos necesario de la plantilla de salida. Para ello la herramienta consulta un archivo de propiedades a quien usa de diccionario para asociar las diversas etiquetas del resultado del Web Service con los atributos de la plantilla de salida.

Si la ejecución no es exitosa se toma el siguiente Web Service y se repite esta fase. Por otro lado, si el conjunto de Web Services resultante de las fases 1 a 4 es vacío, o se agotaron (no hubo ninguna ejecución exitosa), la herramienta procede a realizar una búsqueda de Web Services en el servidor UDDI, teniendo en cuenta los nombres ingresados por el usuario en la plantilla de entrada. A partir de esta búsqueda, se obtiene una lista de Web Services en la cual se descartan aquellos que hayan sido obtenidos de la base de datos previamente. Para cada Web Service resultante, la herramienta realiza el proceso descrito previamente en esta fase. Solo que esta vez, si el Web Service exitoso no se encuentra en la base de datos, este es almacenado para ser tenido en cuenta en una ejecución futura.

5.2. Diseño de la herramienta

5.2.1. Arquitectura general

El diagrama de clases de diseño de la herramienta GG-WSfinder ilustrado en la figura 5.3, muestra cómo se relacionan las principales clases.

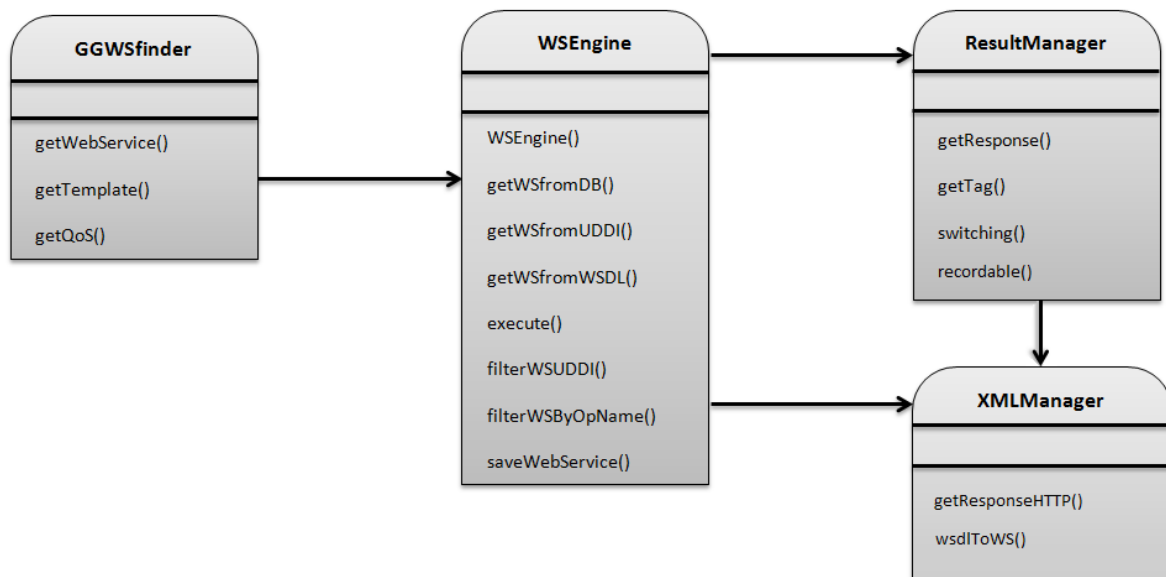


Figura 5.3. Diagrama de clases de diseño.

A continuación se describen las clases de la figura:

- La clase *GGWSfinder* controla la secuencia de pasos o fases descritos en la sección 5.1.2.1, invocando a las operaciones de la clase *WSEngine*. Además, a través de las plantillas de entrada y salida, es la clase encargada interactuar con el usuario. Contiene las tres operaciones implementadas por el Web Service GG-WSfinder tal como muestra la figura 5.4. Estas operaciones son:

GG-WSfinder

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [GetWebService](#)
Get the best answers to your requests.
- [GetTemplate](#)
Get all the templates of the web service.
- [GetQoS](#)
Get all the QoS of the web service.

Figura 5.4. Pantalla inicial de la herramienta.

- ✓ Service Description: Es el enlace a la descripción formal del Web Service en el lenguaje WSDL.
- ✓ GetWebService: Esta operación es la encargada de tomar los parámetros de la plantilla de entrada introducidos por el usuario, y una vez realizado el proceso de selección de Web Services, retornar el resultado en la correspondiente plantilla de salida. Es la principal operación en la ejecución de la herramienta ya que su responsabilidad central es marcar el paso del proceso de selección de Web Services.

La figura 5.5 muestra cómo se visualiza la operación y la 5.6 parte del código fuente que se ejecuta al invocarla.

GG-WSfinder

Click [here](#) for a complete list of operations.

GetWebService

Get the best web service response

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
Type:	<input type="text"/>
* WS Name:	<input type="text"/>
* Operation:	<input type="text"/>
* Parameter:	<input type="text"/>
* QoS:	<input type="text"/>

* Enter the fields, separated by a comma.

Figura 5.5. Operación GetWebService.

```
31  /**
32   * Web service operation
33   */
34  @WebMethod(operationName = "getWebService")
35  public String getWebService(@WebParam(name = "type")
36  String type, @WebParam(name = "wsname")
37  String wsname, @WebParam(name = "operation")
38  String operation, @WebParam(name = "parameter")
39  String parameter, @WebParam(name = "qos")
40  String qos) {
41      try{
42          String finalRequest = "<ws>" +
43                                "<type>"+type+"</type>" +
44                                "<name>"+wsname+"</name>" +
45                                "<operation>"+operation+"</operation>" +
46                                "<param>"+parameter+"</param>" +
47                                "<qos>"+qos+"</qos>" +
48                                "<ws/>";
49
50          Conexion.createConnection();
51          /** CREO EL WSENGINE Y CONVIERTO EN CLASE LA CONSULTA **/
52          WSEngine wse = new WSEngine(finalRequest);
53  }
```

Figura 5.6. Invocación de la operación GetWebService.

Los parámetros de entrada que toma el método web getWebService() se describen a continuación:

- Type: En este campo el usuario debe especificar el tipo de Web Service que desee encontrar. Los distintos tipos de Web Services se obtienen al ejecutar la operación GetTemplate.
La herramienta utiliza este atributo para obtener de la base de datos, el conjunto de Web Services que sean de este tipo.
- Name: Es el nombre del Web Service, o parte de él. El usuario puede ingresar más de uno, en cuyo caso deberá separarlos por coma.

La herramienta utiliza este campo para realizar el filtro por nombre de la base de datos. En caso de que ningún Web Service tenga una ejecución exitosa, este campo será la clave fundamental para realizar la búsqueda en el servidor UDDI.

- Operation: Es el nombre de la operación del Web Service a ejecutar, o parte de él. El usuario puede ingresar más de nombre de operación, en cuyo caso deberá separarlos por coma.

Este campo es tomado para realizar un filtro sintáctico sobre los nombres de cada una de las operaciones de los Web Services.

- Parameter: Este campo cumple dos funciones. Por un lado es el valor semántico que se va a usar como parámetro para ejecutar la operación de un Web Service. Y por otro lado, la herramienta lo utiliza como filtro en cuanto a la cantidad de parámetros de las operaciones. Por este motivo es que el usuario puede ingresar más de un parámetro separados por coma.
- QoS: En este campo el usuario debe especificar las características de calidad que el Web Service debe cumplir. Las distintas normas de calidad implementadas por la herramienta se obtienen al ejecutar la operación GetQoS.

- ✓ GetTemplate: Esta operación consiste en brindarle al usuario los distintos tipos de Web Service que la herramienta soporta, junto con la estructura de la plantilla de cada uno. La figura 5.7 muestra el resultado de la ejecución de esta operación.



Figura 5.7. Estructura genérica de la operación getTemplate().

- ✓ GetQoS: Esta operación consiste en mostrarle al usuario los distintos tipos de características de calidad que la herramienta soporta. El resultado de la ejecución de esta operación se ilustra en la figura 5.8.



Figura 5.8. Estructura genérica de la operación *getQoS()*.

✚ La clase *WSEngine* es el motor del Web Service GG-WSfinder. Es la clase que contiene las operaciones más importantes para llevar a cabo el proceso de selección. A continuación se describen todas sus operaciones mostradas en la figura 5.3:

- ✓ *WSEngine()*:
Es el constructor de la clase. En él se inicia la conexión a la base de datos.
- ✓ *getWSfromDB()*:
Es la operación encargada de realizar la consulta a la base de datos que obtiene los Web Services según el tipo y las características de calidad. Luego realiza el filtro correspondiente al nombre del Web Service.
- ✓ *getWSfromUDDI()*:
Es el método que, a partir de los nombres de Web Service introducidos por el usuario, se encarga de obtener del servidor UDDI todos los Web Services que coincidan con esos nombres.
- ✓ *getWSfromWSDL()*:
Dada una lista de puntos de acceso, este método se encarga de obtener para cada uno el Web Service a partir de su descripción WSDL.
- ✓ *execute()*:
Esta operación primero realiza el filtro de Web Services por nombre de operación y parámetros. Luego ejecuta la primera operación resultante del

filtrado. Si el resultado no es exitoso repite la ejecución hasta agotar todas las operaciones. Cuando el resultado es exitoso, verifica si el Web Service al cual pertenece la operación existe en la base de datos y si no está, lo graba. Finalmente retorna el resultado de la ejecución.

✓ `filterWSUDDI()`:

Su responsabilidad es filtrar los Web Services traídos del servidor UDDI, con aquellos que se encuentran en la base de datos.

✓ `filterWSByOpName()`:

Es la encargada de realizar el filtro por nombre de operación y cantidad de parámetros sobre el Web Service que se le pasa como parámetro. Esta operación se invoca en `execute()`.

✓ `saveWebService()`:

Esta operación es invocada por `execute()` cuando la ejecución de una operación es exitosa. Se encarga de guardar en la base de datos el Web Service, sus operaciones con los parámetros y los tiempos de demora de ejecución de cada operación y del Web Service completo (historial de ejecución).

✚ La clase *ResultManager* es la responsable de crear el resultado y la instancia de la estructura correspondiente al tipo especificado por el usuario. Sus principales operaciones son:

✓ `getResponse()`:

Esta operación se utiliza en la clase *WSEngine* por el método `execute()` para realizar la ejecución de las operaciones resultantes del filtrado: Crea el resultado para el tipo específico de Web Service. Obtiene la respuesta y verifica que esta sea sintácticamente correcta.

✓ `getTag()`:

Es la encargada de asociar, utilizando el archivo de propiedades, cada etiqueta del resultado obtenido en `getResponse()`, con las etiquetas de la estructura de la plantilla de salida para el tipo específico. En caso de que existan coincidencias se invoca al método `switching()`.

✓ `switching()`:

Es responsable de guardar en la estructura de la plantilla los valores de las etiquetas coincidentes.

✓ `recordable()`:

Esta operación determina si la plantilla de salida, una vez completada, posee los atributos necesarios para poder retornar el resultado obtenido de la ejecución y grabar el Web Service en la base de datos.

✚ La clase *XMLManager* es la responsable de manipular todos los documentos XML involucrados en la ejecución de la herramienta. Es la clase que interactúa con los Web Services tanto en la invocación de los mismos como en la obtención del archivo WSDL. Sus principales operaciones se enumeran a continuación:

✓ `getResponseHTTP()`:

Esta operación se encarga de ejecutar una operación de Web Service con sus respectivos parámetros. Es invocada por la operación `getResponse()` de la clase *ResultManager*.

✓ `wSDLToWS()`:

Este método crea un elemento Web Service a partir de la descripción WSDL del mismo. La clase Web Service se describe en la sección 5.3.2.

5.2.2. Modelo de Persistencia

El diagrama de clases persistentes de la herramienta GG-WSfinder se muestra en la figura 5.9. Este conjunto de clases es responsable de almacenar la información de los Web Services por cada ejecución.

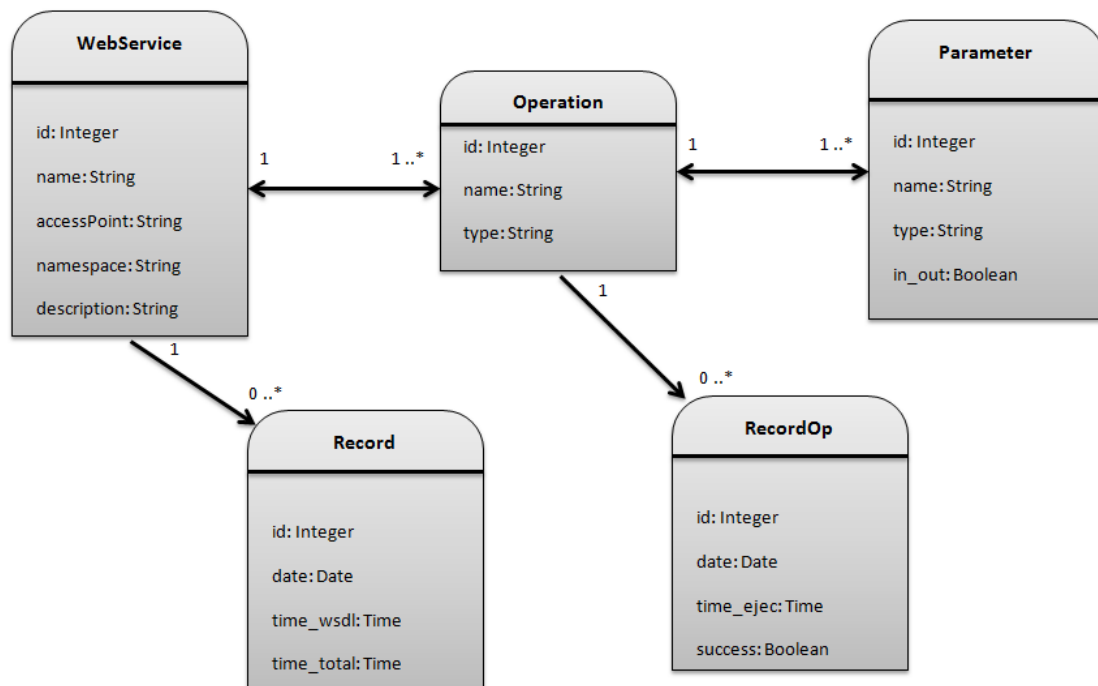


Figura 5.9. Modelo persistente de la herramienta GG-WSfinder

A continuación se describen las clases de la figura:

- **WebService:**

La clase *WebService* se encarga de mantener la información sintáctica básica de cualquier Web Service: tipo, nombre, descripción, punto de acceso y espacio de nombres (name, description, accesspoint y namespace).

```

4
5 public class WebService {
6
7     private List<Operation> operations;
8     private String name;
9     private String namespace;
10    private String accessPoint;
11    private String description;
12    private Integer id;
13    private Long timeWSDL = new Long(0);
14    private Long timeWS = new Long(0);
15
16    public static final String TABLA = "webservice";
17    public static final String AT_ID = "id";
18    public static final String AT_DESCRIPTION = "description";
19    public static final String AT_NAME = "name";
20    public static final String AT_ACCESSPOINT = "accessPoint";
21    public static final String AT_NAMESPACE = "namespace";
22
23    public WebService(){
24
25    }
26
27    /** A continuacion se declaran las operaciones
28     * getters y setters de cada atributo **/
29

```

Figura 5.10. Clase WebService.java

Una instancia de esta clase se crea cada vez que la ejecución de un nuevo Web Service proveniente de UDDI sea exitosa. La figura 5.10 muestra la clase WebService.java que representa a la tabla en la base de datos.

- *Operation:*

La clase Operation se ocupa de almacenar la información sintáctica de las operaciones de un Web Service. Cada vez que se almacena un Web Service, por cada operación del mismo, se crea una nueva instancia de esta clase.

```

6 public class Operation {
7
8     private List<String> paramEntrada;
9     private List<String> paramSalida;
10    private List<Parameter> listParameter;
11    private String name;
12    private Integer id;
13    private Integer idWS;
14    private String type;
15    private Long timeEjec = new Long(0);
16    private Integer success = 0;
17
18    public static final String TABLA = "operation";
19    public static final String AT_ID = "id";
20    public static final String AT_ID_WS = "id_ws";
21    public static final String AT_NAME = "name";
22    public static final String AT_TYPE = "type";
23
24    public Operation(){
25        listParameter=new ArrayList<Parameter>();
26    }
27
28    public Operation(String nombre){
29        name = nombre;
30    }
31
32    /** A continuacion se declaran las operaciones
33     * getters y setters de cada atributo **/

```

Figura 5.11. Clase Operation.java

Los datos que guarda son: el nombre de la operación, el tipo de la misma y un identificador del Web Service al cual pertenecen. La figura 5.11 ilustra la correspondiente clase java.

- *Parameter:*

Cada vez que se almacena una operación, por cada parámetro de la misma, se crea una nueva instancia de esta clase. La información que almacena es el nombre del parámetro, el tipo, si se trata de un parámetro de entrada o salida, y el identificador de la operación a la que pertenece. La figura 5.12 muestra la clase Parameter.java.

```
2
3 public class Parameter {
4
5     private String name;
6     private boolean in_out;
7     private String type;
8     private Integer id;
9     private Integer idOp;
10
11     public static final String TABLA = "parameter";
12     public static final String AT_ID = "id";
13     public static final String AT_ID_OP = "id_op";
14     public static final String AT_TYPE = "type";
15     public static final String AT_NAME = "name";
16     public static final String AT_INOUT = "in_out";
17
18     public Parameter() {
19
20     }
21
22     /** A continuacion se declaran las operaciones
23      * getters y setters de cada atributo */
24
```

Figura 5.12. Clase Parameter.java

- *Record:*

La clase Record contiene información sobre las ejecuciones de un Web Service con el fin de mantener un historial de los tiempos de ejecución del mismo (como se describe en la sección 5.1.2.1). Los datos que guarda son: el identificador del Web Service, la fecha y hora de la ejecución, el tiempo que requiere obtener el archivo WSDL del servicio y el tiempo total que se demora en ejecutar las operaciones del mismo.

Una nueva instancia de la clase Record se crea cada vez que un Web Service es ejecutado. La figura 5.13 muestra la clase java correspondiente a la tabla Record.

```

6 public class Record {
7
8     private Integer id;
9     private Timestamp date;
10    private Long timeWSDL;
11    private Long timeTotal;
12    private Integer idWS;
13
14    public static final String TABLA = "record";
15    public static final String AT_ID = "id";
16    public static final String AT_ID_WS = "id_ws";
17    public static final String AT_DATE = "date";
18    public static final String AT_TIMEWSDL = "time_wsdl";
19    public static final String AT_TIMETOTAL = "time_total";
20
21    public Record(){}
22
23    /** A continuacion se declaran las operaciones
24     *  getters y setters de cada atributo **/
25

```

Figura 5.13. Clase Record.java

- **RecordOp:**

La clase RecordOp contiene información sobre las ejecuciones de una Operación con el fin de mantener un historial de los tiempos de ejecución de las mismas. Los datos que guarda son: un identificador de la operación a la que pertenece, la fecha y hora de ejecución, el tiempo de demora de dicha ejecución y un valor que define si la ejecución ha sido exitosa o ha fallado.

Una nueva instancia de la clase RecordOp se crea cada vez que una operación es ejecutada. La figura 5.14 muestra la clase java que representa la tabla RecordOp.

```

6 public class RecordOp {
7
8     private Integer id;
9     private Timestamp date;
10    private Integer success;
11    private Long timeEjec;
12    private Integer idOp;
13
14    public static final String TABLA = "recordop";
15    public static final String AT_ID = "id";
16    public static final String AT_ID_OP = "id_op";
17    public static final String AT_DATE = "date";
18    public static final String AT_SUCCESS = "success";
19    public static final String AT_TIMEEJEC = "time_ejec";
20
21    public RecordOp(){}
22
23    /** A continuacion se declaran las operaciones
24     *  getters y setters de cada atributo **/
25

```

Figura 5.14. Clase RecordOp.java

5.2.3. Diagrama de Secuencia

Los diagramas de secuencia ilustrados en las figuras 5.15 y 5.16, muestran la interacción que destaca la ordenación temporal de los mensajes durante la ejecución de la herramienta GG-WSfinder. Presentan el conjunto de objetos intervinientes y los mensajes enviados y recibidos por ellos.

Escenario 1:

Un Workflow (o aplicación cliente) realiza una petición a la herramienta GG-WSfinder ejecutando la operación GetWebService. Dados los parámetros especificados en la plantilla de entrada, la herramienta encuentra en la base de datos un Web Service que cumple con dichos parámetros. Luego lo ejecuta y el resultado es exitoso. Finalmente completa la plantilla de salida con ese resultado y lo retorna al Workflow.

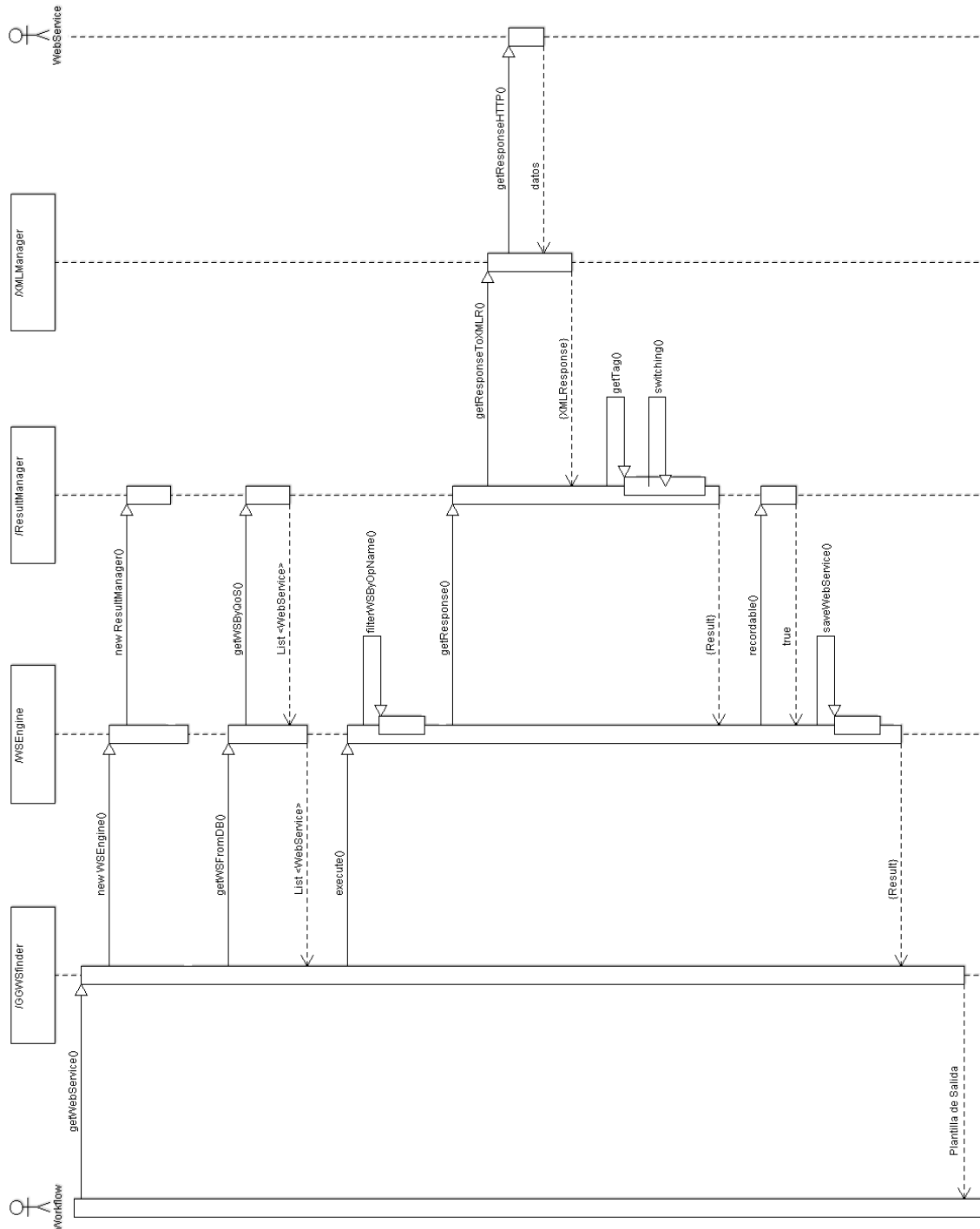


Figura 5.15. Diagrama de secuencia para el escenario 1.

Escenario 2:

Un Workflow (o aplicación cliente) realiza una petición a la herramienta ejecutando la operación GetWebService. Dados los parámetros especificados en la plantilla de entrada, pueden ocurrir dos escenarios. En el primero, la herramienta no encuentra en la base de datos ningún Web Service que cumpla con dichos parámetros. En el segundo, los Web Services encontrados son ejecutados, pero ninguno es exitoso. Por tal motivo, la herramienta procede a buscar en el servidor UDDI un nuevo Web Service, lo encuentra y ejecuta, su resultado es exitoso, lo graba en la base de datos y lo retorna al Workflow.

En este escenario, la secuencia de pasos es idéntica al escenario 1 hasta el punto en que se invoca a la operación recordable() cuyo resultado esta vez es falso. Para simplificar el gráfico se comienza a partir de este punto.

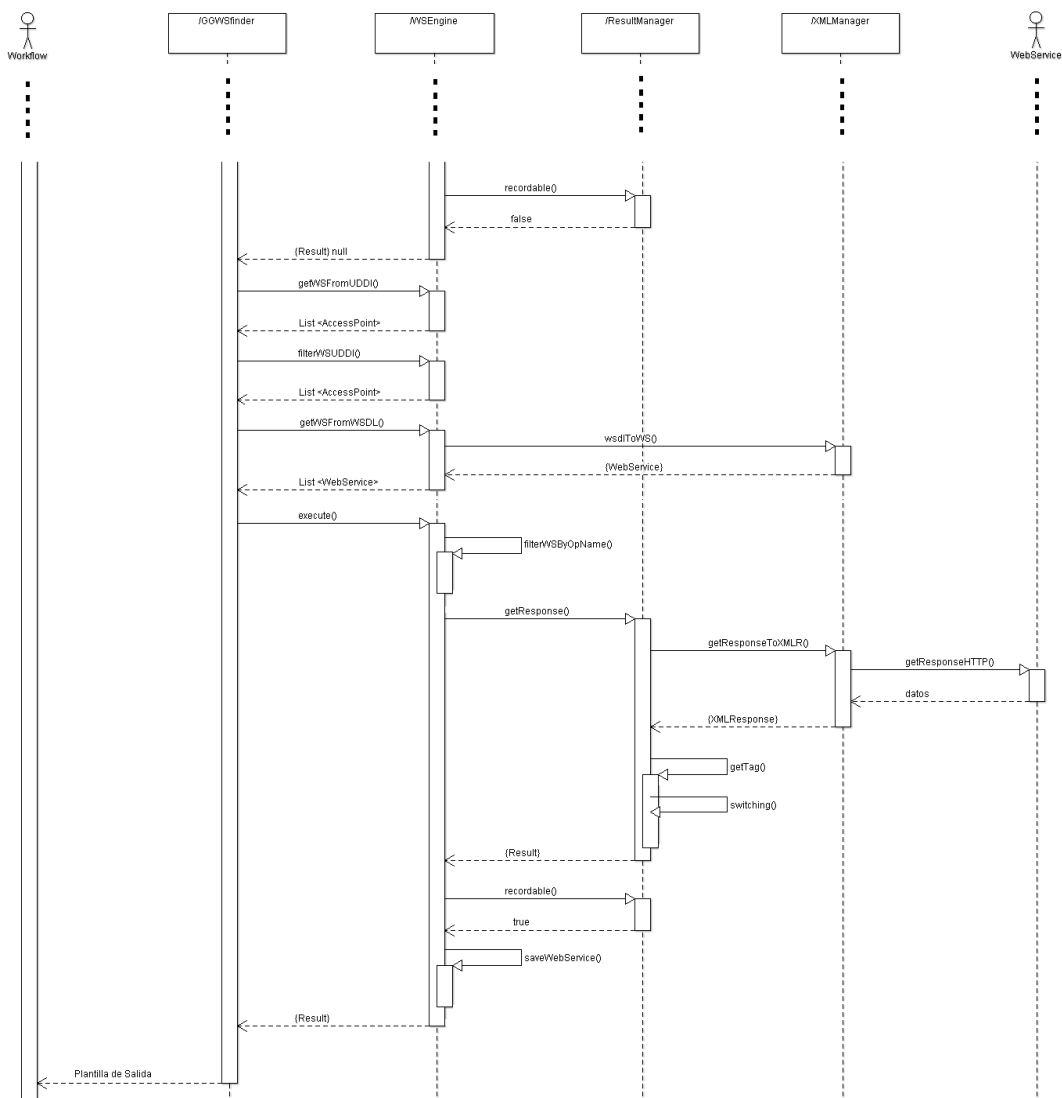


Figura 5.16. Diagrama de secuencia para el escenario 2.

5.3. Descripción del caso de estudio

5.3.1. Elección y Justificación

Como caso de estudio para esta tesis, se opta por implementar la búsqueda y selección de aplicaciones del tipo Servicio Meteorológico. Estas aplicaciones retornan el estado del tiempo de una determinada ciudad del mundo, en base a diversos parámetros como pueden ser el nombre de la ciudad o el código de la misma. En otros casos, inclusive es necesario parametrizar también el país al cual la ciudad pertenece. Otras veces en cambio, alcanza tan solo con pasar como parámetro la latitud y longitud de la ciudad. Por otro lado, en concordancia con esta diversidad de formas de invocar un mismo tipo de aplicación, las salidas o retornos son también muy diferentes. En efecto, van desde algo tan simple como un archivo de imagen con la condición climática para la ciudad requerida, hasta un fichero XML con una serie de atributos (en general del tipo cadena de caracteres) describiendo separada y detalladamente el pronóstico del tiempo actual, extendido, temperaturas, etc.

[SEEKDA] es el motor de búsqueda de Web Services en Internet que se utilizó para seleccionar y generar el conjunto de Web Services del tipo Servicio Meteorológico sobre el cual se operó en este trabajo.

Se trata de un motor de búsquedas gratuito pero no libre de Web Services públicos y sus respectivos proveedores en Internet. Su plataforma tecnológica sólida, no solo es un potente motor de búsquedas sino que también se encarga de recoger información sobre los servicios disponibles en la Web y realizar un seguimiento diario sobre ellos. También permite a sus usuarios editar ciertos datos sobre los proveedores o los servicios que publiquen.

El motor de búsqueda de Web Services [SEEKDA] ayuda a encontrar los Web Services basándose en un catálogo de más de 28.000 descripciones de los servicios. Los Web Services que aparecen en [SEEKDA] cubren un amplio rango de funcionalidades. Por ejemplo, ellos han identificado servicios que pueden enviar mensajes vía fax o SMS, validar direcciones de correos electrónicos o que permiten traducir un texto. Para encontrar Web Services el usuario sólo tiene que escribir la palabra o la frase entera deseada como en cualquier buscador conocido, o bien buscar Web Services y/o sus proveedores por otros criterios de búsqueda que ellos han definido.

De esta manera, el criterio de búsqueda que se elige para generar el conjunto de Web Services deseado es la palabra "*weather*". El resultado que se obtiene arroja más de 70 Web Services, de los cuales tan sólo un 10% aproximadamente se usa en el desarrollo de este trabajo.

Si bien existen numerosos Web Services distribuidos en Internet, de las más diversas índoles y agrupados por diferentes atributos como son el proveedor o la ubicación de la aplicación, se decide escoger los del Servicio Meteorológico por la claridad con la que evidencian el típico problema que existe en la actualidad entre Web Services, particularmente de un mismo tipo. Es decir, las desigualdades sintácticas existentes tanto en sus nombres, como en sus descripciones, operaciones o parámetros, e incluso en las diversas maneras de retornar el resultado de sus ejecuciones, hacen que un proceso de búsqueda y selección exitoso sea complejo y muy inconsistente. La inconsistencia se debe a que bajo un mismo criterio de

búsqueda, en distintas oportunidades, se encontrarán diferentes Web Services. En cambio, la complejidad radica en que se debe garantizar que el resultado de la ejecución del Web Service sea el esperado.

Justamente para poder resolver los tipos de inconvenientes mencionados, es que se selecciona el caso de estudio Servicio Meteorológico, además de la simplicidad técnica al momento de operar con este tipo de Web Services, y por su gran disponibilidad en la web.

5.3.2. Plantillas de Entrada y de Salida

Ahora bien, una vez que se tiene definido el servicio meteorológico como caso de estudio, el usuario debe hacer uso del mismo proporcionándole a la herramienta la palabra “weather” como parámetro en el campo type de la plantilla de entrada. De esta forma, la herramienta trabaja únicamente sobre el conjunto de Web Services del tipo servicio meteorológico tal y como estipula el caso de estudio. El resto de la plantilla de entrada, sirve para ir refinando el proceso de selección de Web Services de este tipo. Así, los valores que el usuario introduce en los campos name, operation y parameter, son utilizados para ir reduciendo el conjunto de Web Services para finalmente quedarse con aquellos que en algún lugar de su sintaxis contienen todos los parámetros de la plantilla de entrada. Con ello, la herramienta no solo se ahorra trabajo y tiempo en cuanto a la cantidad de invocaciones de Web Services que debe hacer, sino que también gana en calidad de respuestas. En otras palabras, los Web Services resultantes de aplicar sucesivamente los filtros, son realmente buenos candidatos y el resultado que se obtiene al ejecutarlos será exitoso.

El resultado final de la ejecución de la herramienta, es una instancia de la plantilla de salida para el tipo de Web Services elegidos como caso de estudio, es decir, la plantilla para el Servicio Meteorológico. La figura 5.16 ilustra la plantilla de salida llamada Result y su instancia denominada WeatherResult (el nombre se debe al caso de estudio). Así, una vez que la herramienta decide que Web Service es el adecuado a invocar, realiza dicha invocación y, no importa de cual se trate específicamente, siempre el retorno final de la herramienta es una instancia de esta plantilla estándar.

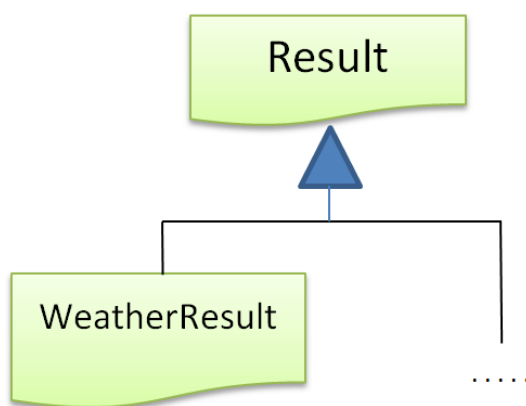


Figura 5.16. Plantilla Result e instancia WeatherResult.

Entonces esta instancia de la clase WeatherResult, que se detalla en la figura 5.17, consiste de una serie de atributos conteniendo los valores típicos que cualquier Web Service del tipo del caso de estudio debe proveer. Es completada o rellena por la herramienta con los valores del resultado obtenido de la correspondiente ejecución del Web Service que la herramienta selecciona para invocar teniendo en cuenta los sinónimos definidos en el archivo de propiedades para este caso de estudio. Con esto, no solo se logra estandarizar el resultado para cualquier tipo de Web Service especificado, en particular el del caso de estudio, sino que también se posibilitan distintos tipos de salidas según los tipos de Web Services requeridos. Y además tiene el adicional de que, sin importar que aplicación es la que provee el resultado, el usuario puede gozar de la consistencia de acceder siempre al mismo valor en la misma posición de la plantilla.

```
9 public class WeatherResult {
10
11     public final static int LOCATION      = 0;
12     public final static int WIND         = 1;
13     public final static int TIME         = 2;
14     public final static int VISIBILITY   = 3;
15     public final static int TEMPERATUREC = 4;
16     public final static int TEMPERATUREF = 5;
17     public final static int PRESSURE     = 6;
18     public final static int SKYCOND      = 7;
19     public final static int HUMIDITY     = 8;
20     public final static int MAXTEMPC     = 9;
21     public final static int MINTEMPC     = 10;
22     public final static int MAXTEMPF     = 11;
23     public final static int MINTEMPF     = 12;
24     public final static int WEATHERIMAGE = 13;
25
26     /*
27      * Resultados
28      */
29     public String root;
30     public String location;
31     public String wind;
32     public String time;
33     public String visibility;
34     public String temperatureC;
35     public String temperatureF;
36     public String pressure;
37     public String skyCond;
38     public String humidity;
39     public String maxTempC;
40     public String minTempC;
41     public String maxTempF;
42     public String minTempF;
43     public String weatherImage;
44
45     public WeatherResult() {
46
47     }
48
49     /** A continuacion se declaran las operaciones
50      * getters y setters de cada atributo **/
```

Figura 5.17. Clase WeatherResult.java

Para comprender mejor tanto el propósito de la plantilla de salida, como así también el rol que cumple dentro del funcionamiento de la herramienta, se proporciona el siguiente ejemplo:

Si la herramienta no retornase siempre una misma estructura resultado, estándar y genérica, el usuario o solicitante se encuentra con el grave inconveniente de tener distintos resultados para una misma invocación en momentos diferentes. A modo de ejemplo, suponga la ejecución de la herramienta para obtener el clima de la ciudad de Barcelona, España. El proceso comienza y, al final de cuentas, GG-WSfinder se queda con la ejecución de un Web Service, proveniente de Estados Unidos, que entre otros atributos retorna las condiciones climáticas en la etiqueta SkyCondition. Hasta aquí, todo marcha bien. La herramienta transfiere el resultado y el usuario, en SkyCondition, encuentra el valor de las condiciones climáticas de Barcelona.

Ahora bien, en una segunda invocación a GG-WSfinder y de nuevo con el valor Barcelona como objetivo, esta vez el Web Service seleccionado para invocar proviene de Argentina, y su ejecución retorna las condiciones climáticas de Barcelona bajo la etiqueta miCondicionClimatica. Es evidente el problema. El usuario tiene para diferentes instancias de GG-WSfinder, pero para una misma ciudad (Barcelona), el resultado de la condición climática en diferentes etiquetas, y operar con ellas sería todo un inconveniente. Imagine el lector la gravedad de este problema para 10, 20, 100 o 1000 llamados que cumplan con el escenario expuesto.

De esta forma, retornar siempre una misma sintaxis, asegura una estandarización en orden de posibilitar, para cualquier tipo de utilidad que se le dé a la herramienta, que el cliente acceda a cierto valor del resultado, como SkyCondition en el ejemplo anterior, siempre en el mismo atributo o etiqueta de la plantilla de salida.

5.3.3. Las características de calidad en la selección de Web Services

En el capítulo 4 se presentaron las características de calidad con mayor importancia en la actualidad (ver tabla 4.2). En este trabajo se elige implementar como caso de estudio el mejor promedio de *Tiempo de Respuesta* de los Web Services para cada ejecución. El tiempo de respuesta es el tiempo esperado entre que se envía un requerimiento y se recibe una respuesta. En inglés, Best Average Response Time (BART).

Para implementar esta característica de calidad se utiliza el historial de tiempos de ejecución de los Web Services almacenado en las clases Record y RecordOp, planteado en las secciones anteriores. El momento en que la característica de calidad se hace efectiva durante el proceso de selección de Web Services, es en la fase 2 en donde se obtienen los Web Services de la base de datos.

Este proceso se lleva a cabo con la consulta a la base de datos que se muestra en la figura 5.18:

```
SELECT t1.*, AVG (t2.time_total) AS prom FROM WebService AS t1, Record AS t2,
Operation AS t3 WHERE (t1.id = t2.id_ws AND t3.type = 'type' AND
t1.id = t3.id_ws) GROUP BY t3.id_ws ORDER BY prom;
```

Figura 5.18. Consulta SQL a la BD

Esta consulta selecciona todos los Web Services cuyas operaciones son del tipo *type*, y los ordena por el promedio de tiempo total demorado en sus ejecuciones. En donde *type* es el parámetro ingresado por el usuario. De esta manera, la herramienta GG-WSfinder garantiza al usuario que se invocan con prioridad aquellos Web Services con menor tiempo de demora.

Para el caso de estudio seleccionado, la consulta SQL debe instanciarse con *type*="weather". El resultado obtenido que se muestra en la tabla 5.1, determina en la herramienta cual debe ser el orden de ejecución de los Web Services de tipo weather.

Id	Name	AccessPoint	NameSpace	Prom
1	Weather	http://ws.cdyne.com/WeatherWS/Weather.asmx	http://ws.cdyne.com/WeatherWS/	2119.1818
2	WeatherService	http://netpub.cstudies.ubc.ca/dotnet/WebServices/WeatherService.asmx	http://netpub.cstudies.ubc.ca/dotnet/webservices/weatherservice	2376.6000
4	WeatherForecast	http://www.webserviceX.net/WeatherForecast.asmx	http://www.webserviceX.net/	5491.5000
3	GlobalWeather	http://www.webserviceX.net/globalweather.asmx	http://www.webserviceX.NET	7323.5714

Tabla 5.1. Resultado de la consulta para el tipo "weather".

5.4 Herramientas utilizadas

En esta sección describiremos las diferentes herramientas y tecnología utilizadas a lo largo de la construcción de este trabajo:

- NetBeans IDE es un entorno de desarrollo - una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java - pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.
- Java v.jdk1.6.0_20 - es un lenguaje de programación orientado a objetos que pertenece a la empresa Sun Microsystems. Entre sus características podemos nombrar: simple, distribuido, robusto, portable, interpretado, multitareas. Se utilizó como lenguaje de programación base dentro de la herramienta NetBeans.

- GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL.
- MySQL es un sistema de gestión de bases de datos (SGBD) multiusuario, multiplataforma y de código abierto que pertenece a la compañía sueca MySQL AB, a la que le pertenece casi todos los derechos del código fuente. El servidor de bases de datos MySQL es la base de datos de fuente abierta más popular en el mundo. Su arquitectura lo hace extremadamente rápido y fácil de adaptar.
- jUDDI es un servidor UDDI de código abierto que tiene que ser integrado con el servidor de aplicaciones Tomcat de Apache. jUDDI proporciona una consola (Juddi-console) a través del cual se pueden publicar y consultar los Web Services.
- Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems.
- UDDI Browser es un proyecto de código abierto que proporciona una interfaz fácil de usar que permite a los usuarios explorar y manipular el contenido de los registros UDDI. Está escrito en Java con las bibliotecas Swing. Actualmente, el navegador es compatible con la versión 2.0 registros UDDI.
- APIs y librerías java utilizadas para el desarrollo del proyecto: qname, jdom, wsl4j.

CAPÍTULO 6: Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones finales de la tesis. Primero, en la sección 6.1 se exponen las principales reflexiones en cuanto a como funciona en la actualidad el proceso de Invocación de Aplicaciones Externas a los Workflows, el planteo de esta comunicación utilizando Web Services, la incorporación de las características de calidad a estos últimos y, finalmente, el desarrollo de una herramienta capaz de llevar a cabo este proceso de manera independiente: GG-WSfinder. Luego, en la sección 6.2 por un lado se plantean las posibles líneas de investigación derivadas de este trabajo final; y por el otro se resumen las extensiones más interesantes respecto la herramienta realizada.

6.1 Conclusiones

Si se puntualiza en el contexto de los Workflows, se puede concluir que en la actualidad la automatización de los procesos de negocio se ha convertido en un objetivo primordial para el crecimiento y desarrollo de las organizaciones. El conflicto de hoy en día en orden de lograr una plena automatización de los procesos, radica en la comunicación de los mismos. Y es en este sentido que la interfaz 3 de un Workflow, la de Invocación de Aplicaciones Externas, toma preponderante significado e importancia. Por ello, en función de aportar una significativa mejoría a esta comunicación, esta tesis ha llevado a cabo las siguientes propuestas expuestas con detalle a lo largo de los capítulos anteriores:

- ✚ El uso de Web Services no solo como parte de la Interfaz de las Aplicaciones Invocadas de los SGWF, sino como un módulo externo al sistema capaz de manipular y automatizar la mencionada comunicación. Es decir, un Web Service encargado de implementar la especificación de la Interfaz 3.
- ✚ La automatización en la búsqueda, selección, ejecución e interpretación de resultados de Web Services.
- ✚ La incorporación de las características de calidad para describir los aspectos no funcionales de los Web Services y con ello la optimización de las invocaciones externas a un Workflow en cuanto a este tipo de aplicaciones.
- ✚ La aplicación e implementación de todo lo propuesto a un caso de estudio concreto.

Por otro lado, la utilización de Web Services en el contexto de la Interfaz de Aplicaciones Invocadas, tiene el principal beneficio o ventaja de aprovechar Internet. Es decir, tener la posibilidad de que las aplicaciones externas al SGWF estén distribuidas en la red, en cualquier parte del mundo, y que ello resulte completamente transparente al SGWF, e incluso al motor del mismo. Así se fundamenta que tanto la búsqueda, como la selección y la ejecución de estas aplicaciones, se realice de forma dinámica y automática, durante la propia ejecución del Workflow y sin la necesidad de una interacción usuario-máquina previo a tal ejecución.

Otra contribución importante de esta Tesis es la optimización de la invocación (y con ello la comunicación) entre los distintos SGWF y las aplicaciones con quienes deben interactuar. Esta optimización consiste en tener en cuenta a la hora de la selección de aplicaciones los atributos no funcionales, o características de calidad, de los Web Services. Ello permite compartir recursos distribuidos a gran escala, lo cual puede implicar por ejemplo la integración entre Workflows.

El uso de Web Services plantea una forma unificada de acceder a las aplicaciones. Este acceso posibilita la interacción con servicios y/o métodos comunes favoreciendo la interoperabilidad a través de las diferentes plataformas. Existen numerosas ventajas que benefician la distribución de las aplicaciones y la interoperabilidad de los sistemas al utilizar este tipo de aplicaciones:

- ✓ 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.

Por todas estas razones es que se convierten en un excelente recurso para permitir la distribución transparente de las aplicaciones externas a los SGWF.

El Web Service GG-WSfinder y su distribución en la red, permite al motor del Workflow abstraerse de la ubicación exacta de las aplicaciones externas a él, de manera que estas aplicaciones puedan cambiar su ubicación en la red y ello no implique modificación alguna en la interacción entre el motor y la herramienta.

Al mismo tiempo, la herramienta a través del uso de la plantilla de salida estándar propuesta, logra desligar al Workflow de la responsabilidad de interpretar las diferentes sintaxis de los resultados obtenidos. Esto no solo reduce la complejidad que implica implementar en un Workflow la interfaz de invocación de aplicaciones, sino que también logra estandarizar el resultado para cualquier tipo de Web Service especificado, en particular el del caso de estudio.

Así, la herramienta posibilita distintos tipos de salidas según los tipos de Web Services requeridos. Y además tiene el adicional de que, sin importar que aplicación sea la que provee el resultado, el Workflow (o cualquier otro tipo usuario) puede gozar de la consistencia de acceder siempre al mismo valor en la misma posición de la plantilla.

Por otra parte, la incorporación de las características de calidad en el proceso de selección de Web Services de GG-WSfinder, impacta altamente en la calidad del resultado del proceso mencionado. La realidad marca que existen múltiples Web Services en cualquier registro UDDI que proveen similares funcionalidades. Por ello las características de calidad se pueden utilizar para afinar y mejorar considerablemente la búsqueda. Con su incorporación, se hace posible que la herramienta evalúe la calidad de los Web Services, filtre la lista de todos los descubiertos y obtenga los más adecuados, con la ventaja de que esta incorporación no requiere modificar el modelo de Web Service actual definido por el protocolo UDDI.

Finalmente, se considera que la elección del tipo de Web Services Servicio Meteorológico como caso de estudio, presenta un escenario ideal para que la herramienta presentada en este trabajo, pueda retornar resultados adecuados respecto de los requerimientos especificados. En su implementación queda ejemplificado que pueden aplicarse los mismos conceptos en cualquier otro tipo de Web Services.

6.2 Trabajos Futuros

A continuación se enumeran las posibles extensiones para este trabajo:

- ✚ En primer lugar, se considera importante que la herramienta sea apta para manipular intercambio de mensajes SOAP, más precisamente las peticiones SOAP que esta debiera hacerle al servidor UDDI, y así dar la posibilidad de que esta comunicación no se realice únicamente a través del protocolo HTTP.
- ✚ También es necesario que la herramienta otorgue diferentes posibilidades en cuanto a la selección de las características de calidad deseadas en la aplicación que se quiera buscar. Mediante un caso de estudio particular, se demuestra que la incorporación de estos atributos es necesaria para optimizar la selección de Web Services y garantizar un resultado adecuado. Es por eso que resulta muy interesante ampliar este espectro y así permitir al usuario de la herramienta escoger entre un conjunto de características de calidad.
- ✚ Por otra parte, podría ser necesario reforzar el proceso de búsqueda, comparación y posterior selección de los Web Services con técnicas de transformación de grafos como se plantea en [Martellotto10]. Este trabajo estipula que se debe analizar con más detalle cuál es la información que necesita registrar el motor del SGWF en cuanto a las características de calidad mínimas requeridas por el propio motor para aceptar un servicio ofrecido en la web. El uso de las técnicas de transformación de grafos en pos de la selección de Web Services, otorgaría la posibilidad de ir más allá de lo sintáctico para entrar a estudiar conceptos de selección dinámica semántica de los Web Services. En otras palabras, esto permitiría que los resultados de las consultas en el protocolo UDDI no estén limitados por las palabras clave definidas en tiempo de diseño.
- ✚ Asimismo, es interesante incorporar los conceptos y objetivos abordados en este trabajo, en las otras interfaces del Modelo de Referencia de Workflow de la WfMC. En particular, a la Interfaz de las Aplicaciones de Clientes (Interfaz 2) y a la Interfaz de Intercambio entre los motores de Workflow (Interfaz 4), cuyas especificaciones resultan aptas para poder llevarlo a cabo.
- ✚ Una variante a lo propuesto en este trabajo, es el hecho de especificar las características de calidad del Web Service en el mismo lenguaje WSDL en el que se describe. Hoy este documento solo soporta la especificación de los aspectos funcionales del servicio y nada dice de los no funcionales. Esto otorgaría 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 podría realizar en un mismo paso y con un mismo criterio de contrastación.

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.
- [BSSB08] Braun I., Strunk A., Stoyanova G., Buder B., ConQo – A Context-And QoS-Aware Service Discovery. Freiburg. IADIS. In proceedings of www/Internet. 2008.
- [Coleman97] Coleman D. Groupware Collaborative Strategies for Corporate and Intranets. Editorial Prentice Hall. ISBN: 0-13-727728-8. 1997.
- [Dambrogio06] D'Ambrogio A. A Model-driven WSDL Extension for Describing the QoS of Web Services. In IEEE proceedings of ICWS. Páginas 789-796. ISBN: 0-7695-2669-1. 2006.
- [FeaturesUDDI3] OASIS. UDDI Version 3 Features List.
http://uddi.org/pubs/uddi_v3_features.htm. Último acceso Abril 2010.
- [FC05] Feldgen M. Clúa O. Web Services, segunda parte. Facultad de Ingeniería. Universidad de Buenos Aires. EGRIET. 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. 2008.
- [HTTP] Hypertext Transfer Protocol - HTTP/1.1.
<http://www.ietf.org/rfc/rfc2616.txt>. Último acceso Abril 2010.
- [JUDDI-Console] Apache jUDDI. Versión 2.0rc6.
<http://juddi.apache.org/releases.html>.
Último acceso Enero 2010.
- [Khoshafian05] Khoshafian S. Introduction to Groupware, Workflow and WorkGroup Computing. Editorial Wiley, 1995. ISBN: 0-471-02946-7.
- [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)". Pages 61-66, Amsterdam. 2005.
- [Labra06] Labra Gayo J.E. Introducción a los servicios web.
<http://www.di.uniovi.es/~labra/cursos/Web20/ServiciosWeb.pdf>
2006.

- [Martellotto10] Martellotto P. Definición de Reglas de Transformación de Grafos y QoS para la Interfaz entre Aplicaciones Externas a un Workflow con Servicios Web. Tesis de postgrado de la Maestría en Ingeniería de Software. Facultad de Ciencias Físicas, Matemáticas y Naturales. Universidad Nacional de San Luis. 2010.
- [Papazoglou08] Papazoglou M. Web Services: Principles and Technology. Ed. Pearson. Education Limited, Prentice Hall, First Edition. 2008.
- [Espina07] Espina P. Extensibilidad de UDDI. Trabajo de Investigación del Doctorado de Lenguajes y Sistemas Informáticos. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. 2007.
- [Ran03] Ran S. A Model for Web Services Discovery with QoS, ACM SIGecom Exchanges 4(1): pp. 1-10, 2003.
- [Klein] Manganelli R., Klein M. Workflow: Como Hacer Reingeniería. Grupo Editorial Norma. ISBN: 958-0430-25-x.
- [SV07] Sanchez S., Vasquez R. Web Services. Trabajo de Investigación Magíster en Tecnologías de la Información. Departamento de Informática. Universidad Técnica Federico Santa María. 2007.
- [SEEKDA] Seekda. Web Service Search Engine.
<http://webservices.seekda.com/>. Último acceso Junio 2010.
- [SOA] OASIS. Reference Model for Service Oriented Architecture 1.0.
<http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>.
Último acceso Marzo 2010.
- [SOAP] World Wide Web Consortium. SOAP Version 1.2 Part 1: Messaging Framework. <http://www.w3.org/TR/soap12-part1/>.
Último acceso Mayo 2010.
- [SpecUDDI3] OASIS. UDDI Version 3.0.2 UDDI Spec Technical Committee Draft, Dated 20041019.
<http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>.
Último acceso Abril 2010.
- [Toledano] Toledano M. Web Services, Introducción y Escenarios para su uso.
<http://www.moisesdaniel.com/articles.html>. Último acceso Febrero 2010.
- [UDDI] OASIS. Universal Description, Discovery and Integration of Web Services–UDDI. Versión 3.0.2.
http://uddi.org/pubs/uddi_v3.htm. Último acceso Mayo 2010.

- [UDDI-Browser] UDDI Browser. Versión 0.2 (beta). <http://uddibrowser.org/>.
Último acceso Enero 2010.
- [VHA06] Vu L., Hauswirth M., Aberer, K. Towards P2P – Based Semantic WebService Discovery with QoS Support. In proceedings of 3rd BusinessProcess Management Workshops. Páginas 18-31, ISBN 978-3-540-32595-6 2006.
- [WFMC] Workflow Management Coalition. The Workflow Reference Model. WfMC-TC00-1003. <http://www.wfmc.org/reference-model.html>. Último acceso Mayo 2010.
- [WfMCa] 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 2010.
- [Woogle] Web Services Search Engine. Halevy A., Dong L., Madhavan J., Zhang J., Nemes E., Lam D.
<http://db.cs.washington.edu/woogle.html>.
Último acceso Abril 2010.
- [WSA] World Wide Web Consortium. Web Services Architecture.
<http://www.w3.org/TR/ws-arch/>. Último acceso Mayo 2010.
- [WSDL1.1] World Wide Web Consortium. Web Services Description Language (WSDL) Version 1.1, <http://www.w3.org/TR/wsdl>.
Último acceso Mayo 2010.
- [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 2010.
- [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 2010.
- [WSDL2.0-2] World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts.
<http://www.w3.org/TR/wsdl20-adjuncts/>.
Último acceso Mayo 2010.
- [XML] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Fifth Edition). <http://www.w3.org/TR/xml>.
Último acceso Mayo 2010.

[ZHYJ07]

Zhang W., Huang Ch. Yang, Y. Jia X. QoS – driven Service SelectionOptimization Model and Algorithms for Composite Web Services. InIEEE proceedings of COMPSAC'07. 2007.

ANEXO A: Tecnologías de los Workflows

Este anexo extiende las definiciones, características y tecnologías en relación a los Workflows y a los Sistemas de Gestión de Workflows. También brinda un exhaustivo desarrollo del Modelo de referencia de la Workflow Management Coalition.

La estructura es la siguiente: En las secciones A.1 y A.2 se introducen los conceptos principales de los Workflows, su historia, las definiciones formales más importantes, etc. La sección A.3 presenta a los Workflows como herramientas de reingeniería. En la A.4 se los clasifica para luego, en la sección A.5 describir sus principales arquitecturas. Finalmente la sección A.6 proporciona el Modelo de la Workflow Management Coalition, y en la sección A.7 se dan las estructuras WAPIs y protocolos más relevantes en los Workflows.

A.1. Introducción

A.1.1. Workflow

Uno de los problemas que se encuentra habitualmente en el desarrollo de aplicaciones para empresas, es que las tareas o procesos que se desarrollan en el entorno laboral de las mismas quedan inmersos en el código de la aplicación que resuelve la problemática de la empresa. Está claro que la gran mayoría de los usuarios no tienen conocimiento de estas tareas, las mismas están ocultas a sus ojos y se realizan automáticamente. El hecho de realizar cambios en dichas tareas o procesos resulta muy costoso, y es muy factible que dichos cambios redunden en realizar nuevamente la aplicación.

Una buena solución al problema anterior es separar los procedimientos y asociarlos a los Workflows realizados dentro de la empresa. Vemos entonces, que el Workflow se relaciona con la automatización de los procedimientos donde los documentos, la información o tareas son pasadas entre los participantes del sistema de acuerdo a un conjunto de reglas previamente establecidas. El fin de lo anterior es llegar a culminar una meta común impuesta por la empresa.

Podemos ver al Workflow como un conjunto de métodos y tecnologías que nos ofrece las facilidades para modelar y gestionar los diversos procesos que ocurren dentro de una empresa. El Workflow 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.

A.1.2. Historia

Se podría decir que la tecnología de Workflow se basa sobre la asunción de que algunas cosas son realizadas más efectivamente por las computadoras que por las personas. Los humanos somos buenos para tomar decisiones, innovar, identificar hechos inesperados. Pero usualmente no somos eficientes en actividades tales como: buscar un documento entre cientos; tener presentes los vencimientos de las tareas que se tienen que realizar dentro de ciertos plazos; así como también el asegurarse de que el trabajo terminado pase de un lugar a otro respetando la secuencia definida.

Al igual que la evolución de la informática en general, la evolución del Workflow está ligada con el cambio en los objetivos centrales de cada época. Si resumimos la evolución de la informática en las últimas cuatro décadas, podremos ver cómo han cambiado los objetivos a seguir de cada época. En la década de los 60' y 70' el gran objetivo era resolver grandes cantidades de cálculo de manera eficiente. En los 80' se buscaba mejorar el manejo y administración de las bases de datos y en los 90' surge la necesidad de entender y poder manejar eficientemente el Workflow, de manera de poder sacarle el mayor provecho posible. Si miramos la actuación del Workflow dentro de estas tres etapas, podremos identificar lo que sería un Workflow Manual en la primera etapa, el Workflow Automatizado dentro de la segunda, y lo que ofrece el Workflow en la actualidad.

En el primer caso podemos ver que antes de que la informática se integrara al trabajo cotidiano, éste era realizado manualmente combinando toda la información en

distintas carpetas. En este ambiente era bastante difícil determinar el estado de una determinada carpeta, así como también el hecho de determinar el proceso a seguir. Se manejaban grandes cantidades de documentos en forma manual, con los consiguientes errores humanos que traían aparejados dichos manejos. Por esto podemos identificar un Workflow Manual inmerso en las tareas cotidianas de esta época. Surge la necesidad de remplazar las actividades manuales por actividades automáticas. Es decir, se busca tener un mayor control y coordinación sobre toda la información que se maneja para llevar a cabo las tareas de las empresas.

Cuando entramos en la década del 80' podemos apreciar la existencia de diversos sistemas de información, donde se maneja y administra toda la información necesaria para llevar a cabo la producción de las empresas. Se ha logrado automatizar ciertas tareas, que antes se realizaban manualmente. Por esto podemos hablar de un Workflow Automatizado. Durante la década del 90' se busca mejorar el flujo de la información, el desafío que se plantea es obtener la información rápida y eficientemente. Surgen las necesidades de incrementar la eficiencia, optimizar la productividad, acortar los tiempos de procesos, tener un control sobre estos, así como también de reducir los costos y mejorar la gestión. Todo esto como consecuencia del incremento de la competitividad y de la exigencia de mejores productos, dentro de un mercado que avanza a gran velocidad.

Finalmente llegamos a la actualidad, en donde nos encontramos con el objetivo de resolver eficientemente el Workflow. Actualmente existe una proliferación de diversos mecanismos de intercambio de información. Los mismos facilitan el manejo del flujo de la información en general. Las metas son similares a las de épocas anteriores, pero el punto de partida, las asunciones y el impacto son distintos. Dentro de la evolución actual del Workflow como tecnología identificamos la evolución y creación de ciertos productos que acompañan al Workflow. Dichos productos son [Coleman97]:

- **Procesamiento de imágenes:** En este caso se captura en forma de imagen electrónica (por ejemplo mediante un escáner) la información o documento que se desea, para luego ser pasada entre los diferentes participantes con distintos propósitos, durante la realización de un proceso.
- **Administración de documentos:** Esta tecnología está relacionada con la administración del ciclo de vida de los documentos. Esta incluye facilidades para guardar en un depósito común aquellos documentos que se comparten, así como también las facilidades para el acceso o modificación de los mismos mediante un conjunto predefinido de reglas.
- **Correo Electrónico y Directorios:** El Correo Electrónico provee las facilidades para distribuir información entre individuos de una organización, o entre distintas organizaciones. El sistema de directorios no sólo provee una forma de identificar a los participantes dentro de un conjunto de direcciones de correo electrónico, nos 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 Workflow guardan la información, reglas, roles, y otros elementos sobre un servidor de Bases de Datos Relacionales, ejecutando la aplicación de Workflow sobre una interfaz gráfica para los usuarios. Estas aplicaciones típicamente incluyen componentes gráficos para el ingreso de los datos.
- **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 Workflow, basado en el valor de los campos de un formulario. Algunos productos para implementar aplicaciones de Workflow proveen constructores de formularios, o se integran a constructores de terceros.

A.1.3. Justificación

Es evidente que el contar con un sistema de Workflow 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 Workflow.

A continuación daremos algunas razones por las cuales las organizaciones podrían considerar adoptar una solución de Workflow:

- ✓ Eficiencia en los procesos y estandarización de los mismos. Esto conduce a una reducción de costos dentro de una empresa.
- ✓ La estandarización de los procesos lleva a tener un mayor conocimiento de los mismos, lo que a su vez conduce a obtener una mejor calidad de estos.
- ✓ Control de los Procesos (Process Management). Utilizando la tecnología de Workflow es posible monitorear el estado actual de las tareas así como también observar cómo evolucionan los planes de trabajo realizados. Permite ver cuáles son los embotellamientos dentro del sistema, es decir aquellas tareas o decisiones que están requiriendo de tiempo no planificado y se tornan en tareas o decisiones críticas.
- ✓ Asignación de tareas a la gente. La asignación de tareas se realiza mediante la definición de roles dentro de la empresa, eliminando la tediosa tarea de asignar los trabajos caso por caso.
- ✓ Recursos disponibles. Se asegura que los recursos de información (aplicaciones y datos) van a estar disponibles para los trabajadores cuando ellos los requieran.
- ✓ Diseño de procesos. Se fomenta a pensar los procesos de una manera distinta a la tradicional forma jerárquica que se utiliza para diseñarlos en la actualidad.

Hay además muchos aspectos operacionales por los cuales es deseable contar con una tecnología de Workflow ya que cosas como la secuencia de tareas, quienes la realizan, mecanismos de control y monitoreo, son implementadas por software de Workflow.

El Workflow pues permite automatizar diferentes aspectos del flujo de la información: Rutear los trabajos en la secuencia correcta, proveer acceso a datos y documentos, y manejar ciertos aspectos de la ejecución de un proceso.

La diversidad de procesos que puede haber en una organización nos lleva a pensar en la existencia de diferentes tipos de software de Workflow. El Workflow entonces, da a una empresa la posibilidad de automatizar sus procesos, reducir costos, y mejorar servicios. Parece ser obvio que son grandes beneficios. Organizaciones que no hayan evaluado esta tecnología podrían encontrarse con desventajas en un futuro.

A.2. Definiciones


A continuación se presentan definiciones más formales sobre los Procesos de Empresa (Business Process) y de Workflow. Estas definiciones son las más representativas dentro de las que se pueden encontrar al investigar sobre el tema.


A.2.1. Procesos de Negocio (Business Process)

“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.” WPMC (Workflow Management Coalition)

Si observamos cualquier empresa de la actualidad, podemos apreciar que la misma está constituida por varias partes. Son estas partes quienes tienen que relacionarse fluidamente para que se cumpla el cometido de la empresa. La definición anterior nos plantea este contexto, en el cual para que todo funcione correctamente, se deben definir previamente la forma en que se comunicarán dichas partes. Actualmente no existe un buen grado de formalización de los procesos de empresa. Un área con gran potencial es rediseñar procesos existentes, eliminar redundancias, identificar embotellamientos y entender porque es que se hace lo que se hace.

A.2.2. Workflow

 *“La automatización de un Proceso de Empresa, total o parcial, en la cual documentos, información o tareas son pasadas de un participante a otro a los efectos de su procesamiento, de acuerdo a un conjunto de reglas establecidas.”* WPMC (Workflow Management Coalition)

 *“La automatización computarizada de un Proceso de Empresa, total o parcialmente.”* WPMC (Workflow Management Coalition)

✚ *“Es la automatización de los procesos que usamos todos los días para llevar a cabo nuestro negocio. Una aplicación de Workflow hace automática la secuencia de acciones, actividades, o tareas usadas para ejecutar el proceso, incluyendo el seguimiento del estado de cada instancia del proceso, así como también las herramientas para manejar el proceso mismo.” Marshak, 1994*

De las definiciones anteriores podemos extraer que los Procesos de Empresa son un punto crítico dentro de lo que es el Workflow y que el objetivo central de éste es automatizar dichos procesos. No lo podemos ver como algo totalmente librado al azar, debe existir una estructura que le dé la lógica de coordinación entre los participantes. Es necesario que existan dos o más individuos para poder hablar de Workflow, y además, estos individuos deben cooperar para alcanzar una meta común.

A.3. Workflow y Reingeniería

A.3.1. Workflow como herramienta de Reingeniería

Existió en cierto momento el concepto de que no podía haber un proceso de Reingeniería si no se usaba Workflow y viceversa. Los especialistas no tardaron en determinar que son dos soluciones independientes. Pero entonces: ¿Qué potencialidad tiene la reingeniería del negocio si además se utiliza Workflow? La respuesta a esta interrogante es inmediata si conocemos algunos principios que la Reingeniería propone:

- ✓ Combinación de tareas desarrollándose en el momento adecuado y donde tienen más sentido.
- ✓ Reducción de tiempos, verificaciones y controles.
- ✓ Disminución de niveles jerárquicos. Esto lleva a la ejecución de los procesos en el orden natural.
- ✓ Las tareas se conviertan en procesos.

Por su parte, el Workflow nos ofrece:

- ✓ Integración entre personas, actividades, programas y datos.
- ✓ Optimización de recursos humanos y técnicos, alineándolos con la estrategia del negocio.
- ✓ Eliminación de partes innecesarias en la secuencia de los procesos y la automatización de dicha secuencia.

Se podrían seguir enumerando elementos, pero la idea es simplemente mostrar que el Workflow es estratégico en cualquier proceso de reingeniería.

Muchas empresas han pensado seriamente en rediseñar, reorganizar e implementar en la práctica una nueva empresa. Algunos autores llaman a esta etapa *Business Process Reengineering (BPR)*. En ese momento se hacen presentes las aplicaciones de Workflow como la posibilidad de hacer factible la eficiencia en áreas de la empresa que aún no contaban con esa estrategia.

A.3.2. Requisitos y beneficios de la reingeniería

La reingeniería de procesos (RP) es un ejercicio de administración del detalle: el tipo de actividad que se beneficia con el uso de herramientas automatizadas (Workflow). Aunque las palabras lo digan es importante recalcar lo siguiente: La reingeniería se beneficia con, y no es el, uso de herramientas automatizadas. El Workflow le ofrece a la reingeniería la posibilidad de tener una herramienta de alto poder conceptual para modelar y diseñar los procesos de empresa. Estas herramientas pueden ser muy importantes a la hora de realizar una reingeniería, pero no tiene por qué ser imprescindible.

La elección adecuada de herramientas puede comprender cualquier cosa, desde lápices, papel y pizarras hasta grandes conjuntos totalmente integrados de herramientas para RP y de ingeniería de software ayudada por computadora (CASE Computer-Aided Software Engineering).

Algunos de los beneficios esperados de las herramientas de reingeniería son:

- ✚ Mejora de productividad.
- ✚ Proyectos más rápidos.
- ✚ Más altos niveles de calidad.
- ✚ Eliminación de trabajo aburridor y, por consiguiente, concentración en trabajo que agrega valor.

Ahora bien, con una herramienta de Workflow podríamos obtener estos beneficios. Sin embargo, obsérvese que estos beneficios sólo se obtienen una vez que se ha aprendido a manejar la herramienta. Éste es un punto que sin duda debemos tener en cuenta, los beneficios de la herramienta decrecen notoriamente si no se tiene destreza para su manejo.

Por otro lado, existen una serie de requisitos que las herramientas de reingeniería deben cumplir:

- ✚ Ser utilizables por las personas de negocios.
- ✚ Generar un rendimiento sobre la inversión.
- ✚ Intensificar la claridad de visión. Este punto es muy importante para nosotros, ya que lograr este objetivo implicaría tener un front-end con los usuarios que permita ver con claridad la realidad planteada.
- ✚ Imponer consistencia en el diseño.
- ✚ Dar refinamiento de arriba abajo, desde las metas corporativas hasta la operación del sistema.

A.4. Workflow: Clasificación

A.4.1. Introducción

Para muchos el Workflow es un área difícil de entender, por este motivo se han dividido los diferentes tipos de aplicaciones de Workflow en categorías.

La diversidad de business process existentes explica porque el mercado de Workflow ha sido dividido en varios segmentos 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.

Por otra parte se estima cuan repetitivo es un proceso. Un proceso es repetitivo si cada instancia del Workflow sigue ciertas reglas, ciertos patrones que son similares para toda instancia del proceso. Por el contrario un proceso no es repetitivo si cada instancia es relativamente única.

Teniendo en cuenta las consideraciones anteriores, actualmente encontramos tres tipos de Workflow:

- De Producción.
- De Colaboración.
- Administrativo.

En los siguientes puntos se describen dichos tipos de Workflow.

A.4.2. Workflow de Producción

Frecuentemente este tipo de Workflow es llamado Workflow de Transacciones. Esto se debe a que en este tipo, la *transacción* en una *base de datos* es considerada la clave de todo proceso considerado.

Este tipo de Workflow es el segmento más grande en el mercado. En general automatizan business process que tienden a ser repetitivos, bien estructurados y con gran manejo de datos.

Un ejemplo de una aplicación de Workflow de este tipo es la realizada para una compañía de seguros:

Cuando el recepcionista de una compañía de seguros recibe un reclamo (por ejemplo para el cobro de un seguro de un auto), el recepcionista ingresa el reclamo a la base de datos, la cual automáticamente dispara un proceso el cual luego de culminado enviará una notificación al cliente del fin del trámite, o en algún caso de los estados intermedios del mismo. Dependiendo de la naturaleza del reclamo, la información es enviada a la persona apropiada dentro de la empresa, la cual determinará si el reclamo es pagado o si es necesario enviar la información un nivel más arriba dentro de la empresa. El proceso disparado por la base de datos a menudo son varios procesos interconectados.

A.4.3. Workflow de Colaboración

Las aplicaciones de Workflow que resuelven business process donde participa gente para lograr una meta común, son llamadas *Workflow de Colaboración*.

Los Workflow de colaboración estructuran o semi estructuran business process donde participa gente, con el objetivo de lograr una meta en común.

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 Workflow mantener la integridad de los documentos.

Actualmente los productos de Workflow no construyen aplicaciones donde la colaboración tome lugar.

A.4.4. Workflow Administrativo

Workflow Administrativo como lo dice su nombre es aquel que involucra procesos de administración en una empresa tales como órdenes de compra, reportes de ventas, etc.

Se emplea Workflow Administrativo si se cumplen ciertas condiciones:

- Hay gran cantidad de procesos de administración dentro de la empresa. Por esto la aplicación de Workflow utilizada debe poder manejar gran cantidad de procesos.
- Una solución de Workflow Administrativo difiere para cada organización, y los cambios son frecuentes. Por esto, la posibilidad de poder hacer cambios de diseño es muy importante.
- 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.

Ejemplo:

Aplicación de manejo de órdenes de compra. Cualquier empleado de la empresa dentro de la empresa podría llenar una orden de compra para un cliente, esto debido a que la aplicación debe proporcionar tanto facilidades de manejo como de cálculos. Si la orden de compra es menor que cierto monto la misma pasa directamente a depósito, allí se hace un control automático de stock y si hay disponible mercadería se hace la entrega de la misma. Si la orden es mayor a un monto dado, entonces la misma es pasada al supervisor para su aprobación y luego sigue su ruta habitual si fue aprobada.

A.4.5. Workflow Ad-hoc

Es muy común ver el Ad hoc Workflow (Workflow caótico) como un tipo de Workflow unido con el Workflow Administrativo. Sin embargo hay quienes consideran que Ad hoc es simplemente una propiedad que puede o no tener una aplicación de Workflow.

¿Qué significa Ad hoc? Ad hoc puede ser interpretado como que algún usuario puede definir o tomar decisiones dentro de la aplicación en cualquier momento, por ejemplo, en una empresa los empleados pueden mandar información a clientes en forma desorganizada y no estructurada, por ej. Vía correo electrónico.

De esta forma, tanto el Workflow de Producción, el de Colaboración, como el de Administración pueden llegar a tener características Ad hoc, lo que hace inútil clasificar los Workflows ad hoc separadamente de los anteriores.

A.5. Modelando Workflow

A.5.1. Introducción

A pesar de la gran variedad de productos de Workflow 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 Workflow, 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 Workflow puedan interactuar entre sí.

A.5.2. Conceptos

Cuando se modela un sistema de Workflow generalmente se identifican y utilizan definiciones de los distintos elementos que se pueden encontrar dentro de dicho sistema. A continuación listamos estos elementos, para luego dar una descripción o definición de cada uno de ellos:

Tareas.

Cada tarea es un conjunto de acciones o actividades 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 Workflow, donde se define por quienes deben ser ejecutadas.

Personas (Usuarios).

Las tareas son realizadas en un orden definido por determinadas personas (o agentes automatizados tomando el rol de las personas) basados sobre las condiciones o reglas del negocio.

Roles.

Cada rol define las distintas competencias potenciales que existen en el sistema. Se definen independientemente de las personas físicas a las cuales se les van a asignar dichos roles. Una persona puede tener más de un rol.

Rutas.

Una ruta define la secuencia de pasos a seguir por los documentos (o información) dentro de un sistema de Workflow. La capacidad de rutear las tareas a usuarios remotos u ocasionales es vital en una aplicación de Workflow. Para asegurar el éxito del flujo de información y decisiones, todos los miembros del equipo deben ser capaces de tomar parte en este proceso.

Se distinguen varios tipos de rutas:

- I. *Rutas Fijas:* En este caso los documentos siguen siempre el mismo camino. Se define de antemano cual es la próxima etapa a seguir.

- II. *Rutas Condicionales*: El camino a seguir depende de la evaluación de condiciones. Estas decisiones se toman en el mismo momento que se pasa por el punto donde hay que evaluar las condiciones.
- III. *Rutas Ad Hoc*: En este caso el usuario elige explícitamente cual es la siguiente etapa a seguir.

Construcción de Rutas:

_AND-Split:

A partir de un lugar fuente, los documentos son distribuidos hacia varios destinos simultáneamente.

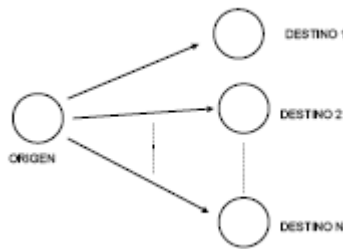


Figura A.1. AND-Split

_AND-Join:

A partir de varios lugares fuentes, los documentos convergen, sincrónicamente, hacia un único destino.

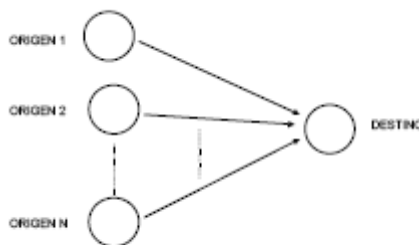


Figura A.2. AND-Join

_OR-Split:

A partir de un lugar origen, los documentos toman un destino entre varios posibles.

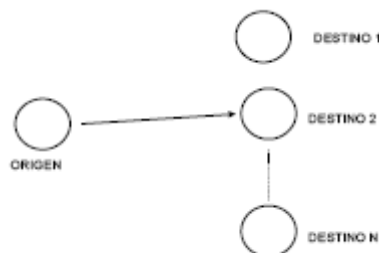


Figura A.3. Or-Split

_ OR-Join:

A partir de uno o más lugares de origen, dentro de varios posibles, convergen hacia un único destino (no se requiere sincronización).

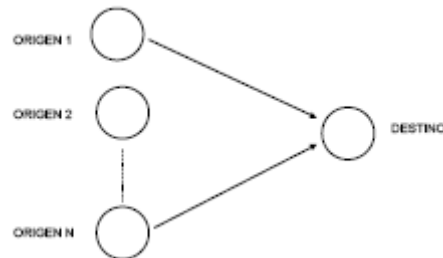


Figura A.4. OR-Join

_ LOOP:

En este caso se forma un circuito cerrado dentro del camino que recorren los documentos.



Figura A.5. LOOP

Reglas de Transición.

Son reglas lógicas que determinan la navegación del documento dentro del sistema.

Expresan que acción se va a tomar dependiendo del valor de expresiones lógicas. La definición de las reglas puede ser muy complicada, con múltiples opciones, variaciones, y excepciones. Un ejemplo sencillo podría ser el siguiente: Un cliente solicita un préstamo por US\$ 1000, entonces la siguiente regla expresa el camino a seguir sobre la base de la solicitud: “Si la cantidad solicitada es mayor que el tope del cliente ENTONCES enviar la solicitud al supervisor del área, SINO, entregar el dinero”. La regla anterior muestra, de manera sencilla, el tipo de reglas que comúnmente se expresan.

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 Workflow encontramos:

_ *Datos de Control:* Son los datos internos manejados por la lógica del sistema de Workflow.

_ *Datos Relevantes:* Son aquellos datos utilizados para determinar el ruteo de las distintas tareas del sistema.

_ *Datos de la Aplicación:* Estos datos son específicos de la aplicación, no son accedidos por la lógica del Workflow.

La noción de documento como recipiente de información que se transmite de una tarea a otra, es muy utilizada. Por esto, cuando nos refiramos a datos manejados por el sistema, los nombraremos por documentos.

Existen ciertas propiedades que se le pueden asociar a un documento, como ser: la definición de los derechos de acceso a los mismos; las vistas definidas sobre ellos; el permitir manejar los accesos concurrentes (o sea, que dos personas o procesos puedan acceder al documento simultáneamente); también se pueden definir formas de relacionar datos provenientes de fuentes externas al documento, como ser, datos de la aplicación o de la Base de Datos.

Eventos.

Un evento es una interrupción que contiene información, el mismo tiene un origen y uno o más destinatarios. La información contenida en el mensaje que se produjo por el evento puede ser implícita o dada por el usuario. Los eventos pueden ser disparados voluntariamente por el usuario; o en forma implícita durante un proceso según el estado de los datos o de decisiones tomadas por el usuario; o en forma automática. Por ejemplo, cuando un gerente de un banco hace una consulta sobre ciertos datos para hacer una auditoria, se dispara un evento que le devuelve la información de dicha consulta.

Plazos (Deadlines).

Podemos ver a los plazos como los tiempos que se le asignan a ciertos elementos.

Ejemplos de plazos pueden ser: el tiempo máximo que se le asigna a una tarea para que sea terminada; el tiempo máximo para recorrer una ruta; terminar una tarea antes de cierta fecha; terminar el recorrido de una ruta antes de cierta fecha; y así podríamos seguir.

A los plazos podemos asignarles eventos, de forma tal de que cuando venza determinado plazo se disparen ciertos eventos asignados por el usuario, o programados para que se disparen automáticamente.

Procesos.

Anteriormente definimos lo que son los procesos de empresas, pero cabe acotar que estos procesos son tan variados y personalizados, como la gente que toma parte en ellos.

Comúnmente los procesos no son “diseñados”, sino que son identificados en la realidad, por el uso diario que se les da. “Nosotros siempre lo hemos hecho así” es una expresión común que se identifica al momento de evaluar estos procesos. Es común que se piense en poner todos los procesos dentro de una aplicación, pero suele ocurrir que sólo algunos de ellos compongan la aplicación final.

Políticas.

Las políticas son una manera formal de expresar sentencias de cómo serán manejados ciertos procesos. Por ejemplo, todas las empresas tienen políticas de licencias vacacionales y beneficios para sus empleados, y podrían definir además como se manejarán los distintos procesos de empresa que la componen.

A continuación se muestra un ejemplo donde se identifican algunos de los elementos explicados en los puntos anteriores:

En la figura A.6 se muestra un diagrama sencillo de un proceso de solicitud de compra de algún producto en una empresa. Esta solicitud ingresa al sistema vía teléfono. En él se pueden identificar las tareas que comprenden el proceso, ruta por la cual fluye la solicitud, reglas de transición entre las tareas, así como también usuarios, roles y eventos.

Las tareas son las que están representadas por rectángulos con una descripción asociada dentro de los rectángulos.

El diagrama está dividido en tres partes, cada una de las partes identifica las tareas realizadas por el rol que corresponde. Los roles que identificamos son: el empleado de atención al público, la sección de ventas y el empacador. Observar que el rol de empacador puede asociarse a una persona en particular y que el rol de sección de ventas no necesariamente identifica a una persona.

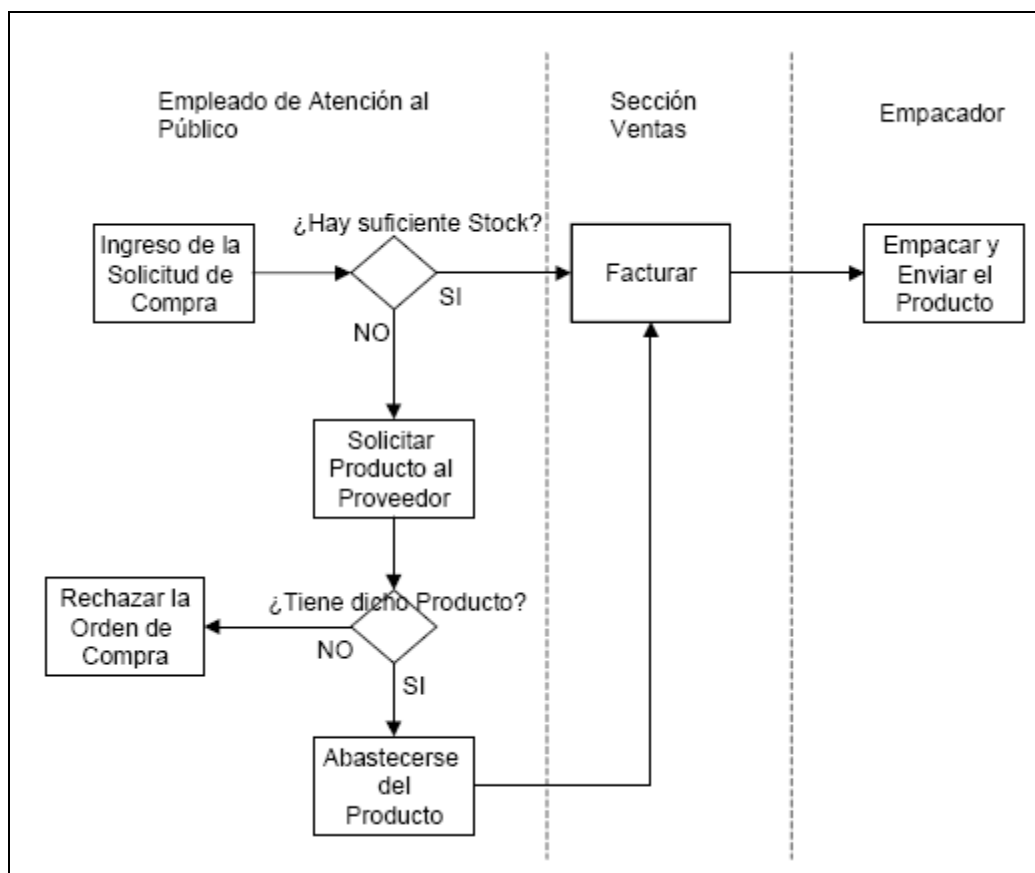


Figura A.6. Diagrama del Ejemplo.

Vemos como el evento asociado al ingreso de una solicitud de compra por teléfono hace que se dispare todo un proceso donde la solicitud de compra va recorriendo ciertas rutas según las condiciones que se van dando. Las reglas de transición se identifican por rombos con una condición asociada. Según el valor de estas condiciones la solicitud de compra toma uno u otro camino. Una de las rutas más sencillas que se pueden identificar es cuando existe producto en stock para el producto de la solicitud de compra, luego de lo cual se pasa a facturar y finalmente se empaqueta y se envía.

A.5.3. Arquitecturas

En un producto de software de Workflow genérico se identifican una serie de componentes e interfaces. La implementación de esta estructura puede ser realizada de varias formas diferentes entre sí. Éste es un punto de desencuentro de los productos existentes.

En los puntos que siguen a continuación explicaremos los diferentes modelos de implementación en forma genérica. Previamente se dará una descripción de las principales componentes de un sistema de Workflow genérico.

A.5.3.1 Componentes

Las principales componentes de un sistema genérico de Workflow son ilustradas en la figura A.7.:

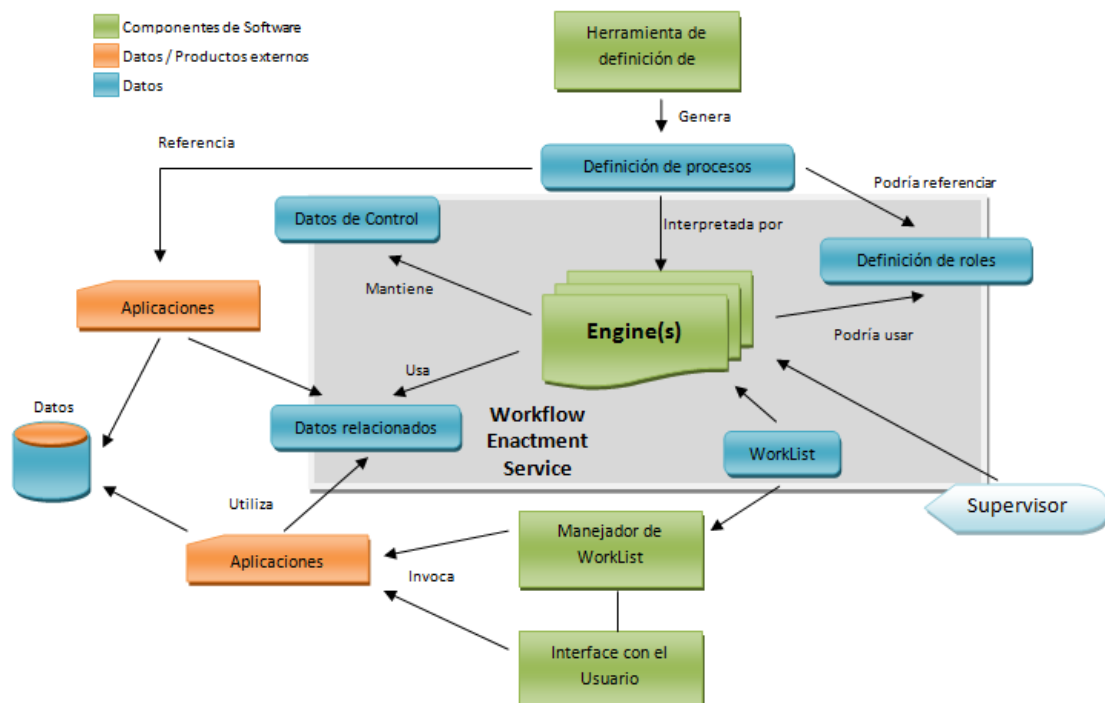


Figura A.7. Modelo de componentes del Workflow.

En este modelo genérico encontramos tres tipos de componentes:

- ✚ *De software:* Proveen soporte para gran cantidad de funciones del sistema de Workflow.
- ✚ *Datos y Definición de procesos:* Usados por los componentes de software.
- ✚ *Aplicaciones externas.*

A continuación describiremos los elementos más importantes mostrados en la figura:

- ✓ **Herramienta de Definición de Procesos.** Forma parte de las componentes de software del Workflow y la podemos ver en el borde superior de la figura. Es utilizada para crear una descripción de los procesos en una forma procesable para una computadora. Esta herramienta podría estar basada en un lenguaje de definición de procesos formal, en un modelo de interacción entre objetos, o simplemente en un conjunto de reglas de ruteo para transferir información entre los participantes.

Esta herramienta puede ser proporcionada como parte de un producto de software orientado a Workflow, o podría simplemente existir por si sola y tener integración con diferentes productos de Workflow.

- ✓ **Definición de Procesos.** Luego de la componente anterior encontramos la Definición de Procesos, que forma parte de los datos del Workflow. Contiene, toda la información necesaria acerca de los procesos, incluye información de comienzo de actividades, condiciones, y reglas de navegación.

Podría tener referencias a la definición de roles, donde se almacena información de la estructura organizacional. Esto quiere decir que en la definición de procesos se puede mencionar que en cierto proceso participa cierto rol, el cual está definido en la definición de roles.

- ✓ **Workflow Enactment Service.** Esta componente interpreta la descripción de procesos y controla las diferentes instancias de los procesos, secuencia de actividades, adiciona ítems (elementos) a la lista de trabajo de los usuarios (Worklist), e invoca aplicaciones necesarias.

Todas estas tareas son hechas por uno o más motores de Workflow (engines), los cuales manejan la ejecución de las distintas instancias de varios procesos.

- ✓ **Worklist (lista de trabajo).** La Worklist forma parte de los datos del Workflow y la podemos apreciar en la parte inferior de la figura. Ya que la interacción con los usuarios es necesaria en algunos casos, el motor de Workflow utiliza una Worklist manejada por un manejador de Worklist para controlar tal interacción. El motor deposita en la Worklist ítems a ser ejecutados para cada usuario. La Worklist puede ser visible o invisible para los usuarios depende del caso, muchas veces se deja que el usuario seleccione ítems y los procese en forma individual.

- ✓ **Manejador de Worklist.** Luego de la componente anterior encontramos el Manejador de la Worklist. Es un componente de software el cual maneja la interacción entre los participantes del Workflow y el Workflow Enactment service, vía la Worklist.

El manejador soporta en general un amplio rango de interacción con otras aplicaciones clientes. En la figura la interface con el usuario es mostrada como una componente separada del manejador de Worklist. En algunos sistemas estas dos componentes están agrupadas como una única entidad funcional.

A.5.3.2 Implementación del Workflow Enactment Service

El Workflow Enactment Software consiste de uno o más Motores (engines) de Workflow, 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 Workflow, 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.

La figura A.8 representa gráficamente lo comentado:

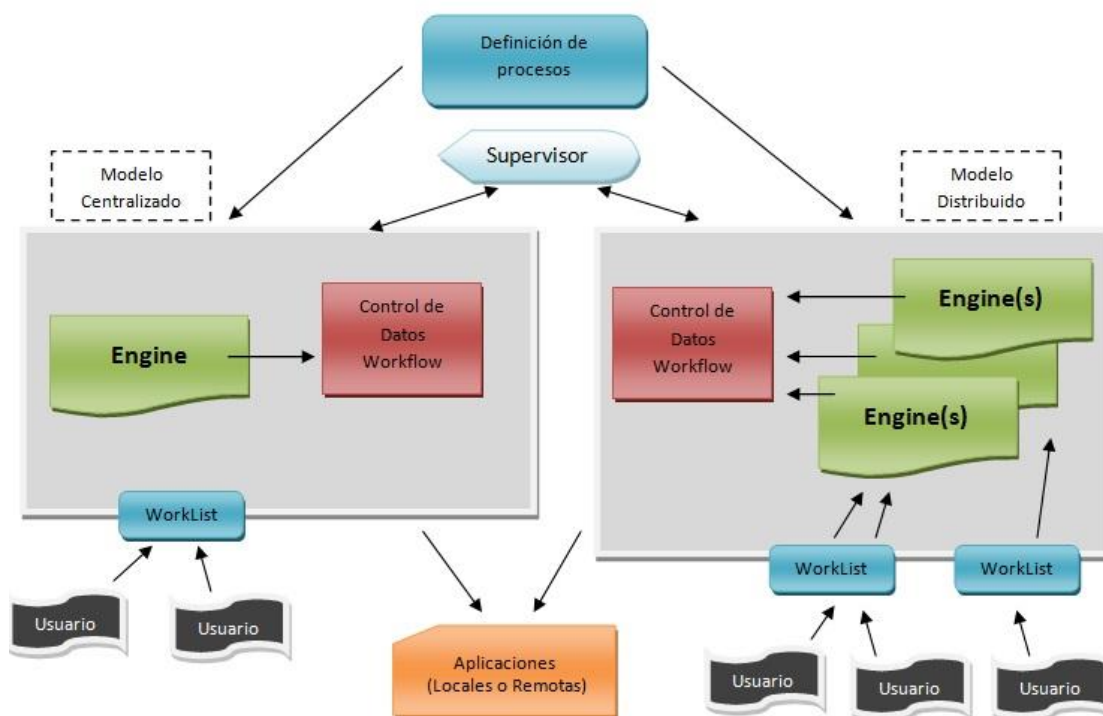


Figura A.8. Implementación del Workflow Enactment Service.

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, 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.

A.5.3.3 Alternativas de aplicaciones clientes de Workflow

En el modelo de Workflow existe interacción entre el manejador de la Worklist y un motor en particular. Recordamos que una Worklist es una cola de tareas asignadas a un usuario en particular (o posiblemente un grupo de usuarios), la asignación es hecha por el Workflow Enactment Service. Hay varias implementaciones para la interacción con la Worklist, dependiendo principalmente del tipo de infraestructura utilizada para soportar la distribución del manejador de la Worklist.

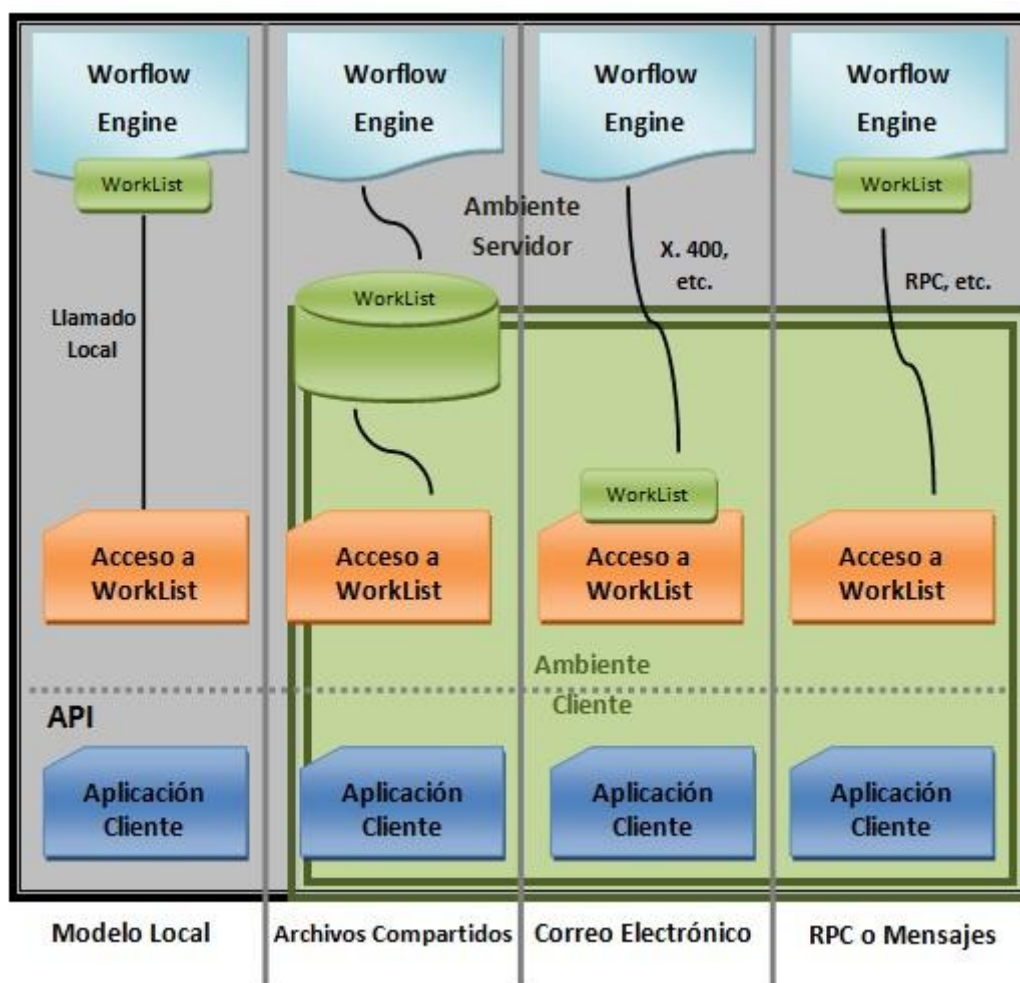


Figura A.9. Aplicaciones Cliente del Workflow.

Hay cuatro posibilidades que son mostradas en el diagrama de la figura A.9, una implementa un manejador de Worklist en forma centralizada y las otros tres en forma distribuida.

Los cuatro escenarios son:

- I. **Modelo local:** La comunicación entre el motor y el manejador de la Worklist es vía una interface local. En este caso la interface con el usuario podría ser vía una terminal local o una Workstation remota.
- II. **Archivos de almacenamiento compartido:** El manejador de la Worklist es implementado en el cliente y la comunicación con el motor es vía archivos compartidos.
- III. **Modelo de correo electrónico:** La comunicación es vía correo electrónico.
- IV. **Llamado a procedimientos remotos (RPC) o pasaje de mensajes:** En este escenario la Worklist podría estar físicamente en el motor o en el manejador de la Worklist dependiendo de las características particulares de la implementación.

A.6. Modelo de la Workflow Management Coalition (WFMC)

A.6.1. Introducción

La WFMC es una agrupación compuesta por compañías, vendedores, organizaciones de usuarios, y consultores. El objetivo de esta agrupación es ofrecer una forma de “diálogo” común a todos. De esta forma las diferentes herramientas que se implementen en esta área podrán tener cierto nivel de interoperabilidad, es decir, podrán comunicarse entre ellas para poder realizar las distintas tareas involucradas en un sistema de Workflow.

A.6.2. Necesidad de Estandarizar

Se estima que actualmente los distintos productos de Workflow que hay en el mercado sobrepasan los cien. Cada uno de ellos se enfoca sobre distintos aspectos funcionales, como ser, herramientas de diseño visual, en las cuales se ofrecen ciertos diagramas para representar la realidad. Otras se enfocan en la integración de los datos con las aplicaciones.

El desarrollo de estándares para la interoperabilidad de las diversas herramientas, nos permitiría la elección de los mejores productos, según el enfoque que le demos a nuestra aplicación. Por ejemplo, podríamos comprar una herramienta que nos ofrezca un entorno amigable para realizar el análisis y diseño de nuestro problema, y por otro lado comprar una componente que nos resuelva la auditoria de los datos y poder trabajar en forma integrada con las dos componentes.

Una de las estrategias que siguen actualmente las empresas, es rediseñar sus procesos, esta metodología es denominada como Reingeniería de los Procesos de Empresas (Business Process Re-engineering). Esto puede ser causa de cambios organizacionales, legislativos, cambios en los objetivos del negocio, etc. En esta situación, muchas veces es necesario relacionarse con otras organizaciones. Pero para poder hacer esto debe existir la posibilidad de que los productos de un vendedor puedan comunicarse con los de otro, pues es claro que cada empresa u organización comprará los productos que crea conveniente para su caso.

Vemos entonces la necesidad de estandarizar la forma de comunicación entre los distintos componentes de un producto de Workflow. De modo de poder tener flexibilidad a la hora de operar con distintos productos. Esta necesidad se justifica además por las proyecciones que se tienen actualmente, sobre la penetración de la tecnología de Workflow en el mercado en los próximos años. Por esto se debe atacar el problema de potenciales incompatibilidades de antemano, y no cuando existan miles de productos en esta área, cada uno con sus particularidades.

A.6.3. Modelo de referencia de la WPMC

El modelo de referencia de Workflow fue desarrollado desde estructuras genéricas de aplicaciones de Workflow, identificando las interfaces con estas estructuras, de forma de permitir a los productos comunicarse a distintos niveles. Todos los sistemas de Workflow contienen componentes genéricos que interactúan de forma definida. Para poder tener cierto nivel de interoperabilidad entre los diversos productos de Workflow, es necesario definir un conjunto de interfaces y formatos para el intercambio de datos entre dichas componentes.

A.6.3.1. El Modelo de Workflow

En la figura A.10 se muestra las distintas interfaces y componentes que se pueden encontrar en la arquitectura del Workflow.

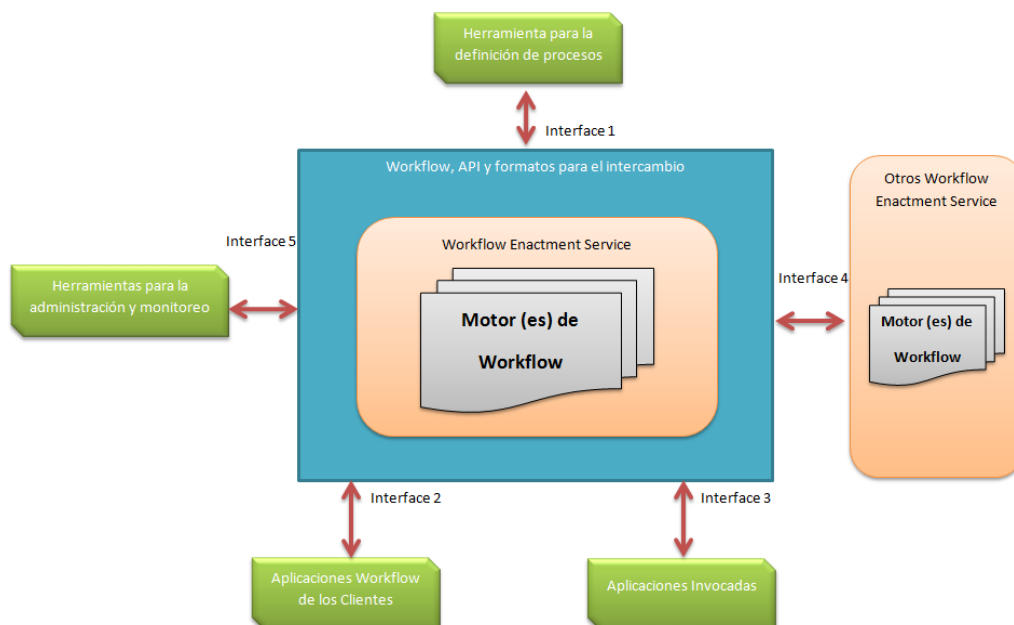


Figura A.10. Interfaces del Modelo de Referencia.

En el modelo adoptado hay una separación entre los procesos y el control de la lógica de las actividades. Esta lógica está dentro de lo que ya definimos como el Workflow Enactment Service. Esta separación permite la integración de las diversas herramientas con una aplicación particular.

La interacción del Enactment Service con los recursos externos se da por una de las dos interfaces siguientes:

- ✓ La interface de las Aplicaciones de los Clientes, a través de la cual el Motor de Workflow interactúa con el manejador de la Worklist, responsable de organizar el trabajo por intermedio de un recurso de usuario. Es responsabilidad del manejador del Worklist elegir y hacer progresar cada elemento de la lista de trabajo (Worklist).
- ✓ La interface de las Aplicaciones Invocadas, la cual le permite al motor de Workflow activar una herramienta para realizar una actividad particular. Esta interface podría ser basada en un servidor, es decir no existe la interacción con el usuario.

Hasta ahora hemos visto al Enactment Service como una entidad lógica, pero físicamente éste podría estar centralizado o funcionalmente distribuido. En un Enactment Service distribuido, distintos motores de Workflow controlan una parte del proceso e interactúan con un subconjunto de usuarios y herramientas relacionadas con las actividades que llevan a cabo el proceso. En este tipo de sistemas se deben usar determinados protocolos y formatos para el intercambio de información entre los distintos motores de Workflow.

A.6.3.2. Motor de Workflow (Workflow Engine)

Es el software que provee el control del ambiente de ejecución de una instancia de Workflow. Típicamente dicho software provee facilidades para:

- ✚ Interpretación de la definición de procesos.
- ✚ Control de las instancias de los procesos: creación, activación, terminación, etc.
- ✚ Navegación entre actividades.
- ✚ Soporte de interacción con el usuario.
- ✚ Pasaje de datos al usuario o a aplicaciones.
- ✚ Invocación de aplicaciones externas.

A.6.3.3. Tipos de Workflow Enactment Services

Podemos encontrar Workflow Enactment Services homogéneos, los cuales están constituidos por uno o más motores de Workflow compatibles. Estos proveen un ambiente de ejecución, con un conjunto definido (específico del producto) de atributos en la definición del proceso. La interacción entre estos motores no está estandarizada, o sea, es específica de los productos.

Se pueden encontrar también, Workflow Enactment Services heterogéneos, que están constituidos de uno o más servicios homogéneos, los cuales siguen un estándar para la interoperabilidad entre los mismos.

Se ofrecen distintos niveles de conformidad en cuanto a la estandarización. La interoperabilidad de los distintos productos depende del nivel de conformidad. Como se dijo anteriormente, tenemos distintos motores de Workflow controlando una parte del proceso e interactuando con otros motores en un dominio de trabajo distinto. Se espera que los siguientes puntos estén entre los niveles de conformidad de los productos para poder soportar la interacción de los diversos motores:

1. Se debe tener un esquema de nominación común a través de motores heterogéneos.
2. Se debe soportar un proceso de definición común para los objetos y atributos, de manera que los diversos motores puedan acceder a ellos.
3. Se debe soportar la transferencia de los datos relevantes del Workflow, a través de los motores.
4. Se debe soportar la transferencia de procesos, sub-procesos o actividades entre los distintos motores de Workflow.
5. Se debe soportar funciones de administración y monitoreo comunes, dentro de un dominio de motores de Workflow.

A.6.3.4. Proceso y estados de transición de las actividades

El Workflow Enactment Service podría ser considerado como una máquina de estados, donde los procesos cambian de estados según eventos externos, o decisiones de control específicas, tomadas internamente por el motor de Workflow.

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

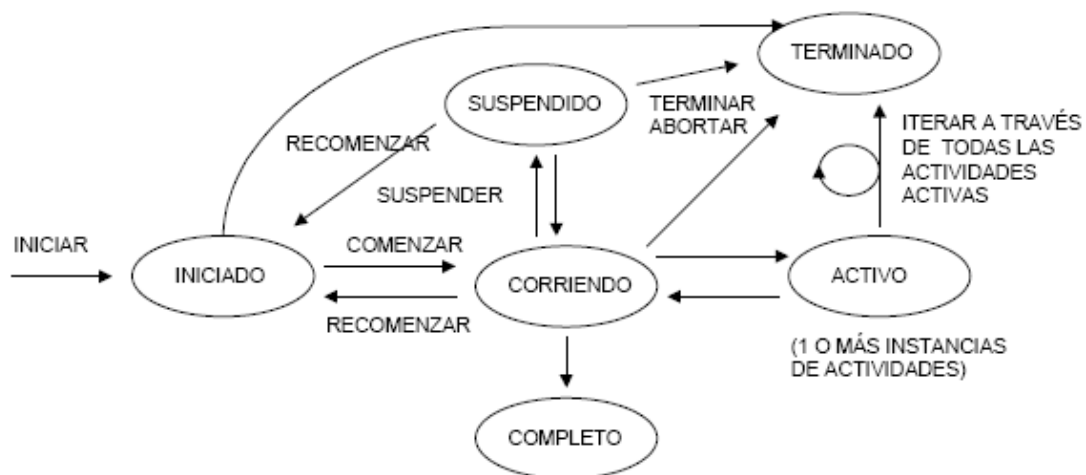


Figura A.11. Estados básicos de un proceso.

La figura A.11 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: Se suspende la ejecución del proceso.

_Completado: El proceso culminó, se realizan las acciones programadas (auditoria) y luego se elimina la instancia del proceso.

_Terminado: No se pudo terminar normalmente la ejecución del proceso.

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:

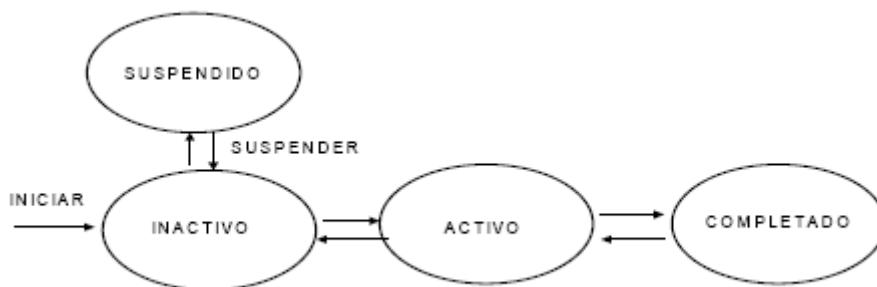


Figura A.12. Estados básicos de una actividad.

_Inactivo: La actividad dentro de la instancia del proceso ha sido creada pero no ha sido activada y no tiene ningún elemento (Workitem) para procesar.

_Activo: Un Workitem ha sido creado y asignado a la instancia para su procesamiento.

_Suspendido: Se suspende la ejecución de la instancia de la actividad. A la misma no se le asigna un Workitem hasta que no vuelve al estado Inactivo.

_Completado: La ejecución de la instancia de la actividad ha sido terminada normalmente.

A.6.3.5. Workflow Application Programming Interface (WAPI)

Las WAPI pueden ser vistas como un conjunto de llamadas API (Application Programming Interface) y funciones de intercambio soportadas por el Workflow Enactment Service. Las APIs son un conjunto de llamadas a funciones de software que permiten a las aplicaciones acceder a funciones de un programa. Las WAPI permiten la interacción del Workflow Enactment Service con otros recursos y aplicaciones.

A.6.3.6. Datos de: Control, Relevantes y de las Aplicaciones de Workflow

El Workflow Enactment Service mantiene el control interno de los datos, identificando el estado de un proceso o de las instancias de las actividades y eventualmente podría soportar algún estado interno más. Estos datos no están 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 de Workflow para mantener el control de las diversas instancias de los procesos.

Los datos relevantes son aquellos que son usados por el motor de Workflow para determinar los estados de transición de una instancia de un proceso.

La manipulación de datos desde una aplicación puede ser requerida por alguna actividad en la definición de un proceso. Estos datos son específicos de la aplicación y no son accesibles por el motor de Workflow.

A.6.3.7. Intercambio de Datos

El intercambio de datos relevantes y datos de las aplicaciones es requerido a través de las WAPI para soportar la interacción de las tres funcionalidades siguientes en tiempo de ejecución:

- ✓ Manejador de la Worklist (interface 2)
- ✓ Aplicaciones Invocadas (interface 3)
- ✓ Intercambio entre los motores de Workflow (interface 4)

El intercambio directo de los datos de las aplicaciones es tipificado por sistemas de Workflow 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 está asociado el formato de los datos. Esto le permite a los sistemas de Workflow heterogéneos proveer la conversión de los datos sobre la base del tipo de datos de los mismos.

En el caso de un sistema de Workflow 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.

A.6.3.8. Definición de procesos (interface 1)

Herramientas de definición de procesos: Hay gran variedad de herramientas utilizadas para el análisis de procesos. 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 de procesos” la cual pueda ser interpretada en runtime por el o los motor(es) de Workflow.

Definición del intercambio de datos: Existe 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 de Workflow, lo deseable es que esta herramienta pueda comunicarse con cualquier motor, esto únicamente sería posible si se establecen ciertas normas de comunicación entre las herramientas de definición de procesos y un motor de Workflow. Por esto la WFMC propone una interface para esta comunicación.

El objetivo de la interface es dar un formato de intercambio y llamadas a APIs (Application Programming Interface), para soportar el intercambio de información de definición de procesos. El intercambio podría ser una completa definición de los procesos o un subconjunto de la misma. En la figura se muestra la composición de la definición de procesos.

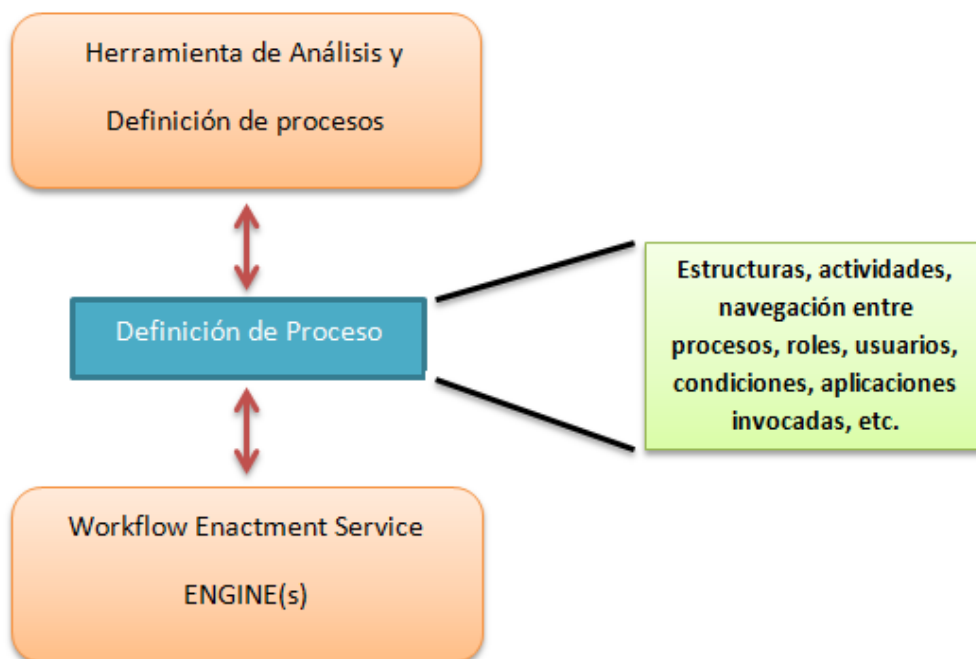


Figura A.13. Definición de procesos.

A.6.3.9. Interface del Workflow con aplicaciones clientes (interface 2)

En el modelo planteado la interacción entre las aplicaciones clientes y el motor de Workflow esta sostenido en gran parte por el concepto de Worklist ya descrito anteriormente.

Parte de la información almacenada en la Worklist es utilizada para trasmitirle al manejador de la Worklist que aplicaciones hay que invocar.

La Worklist podría contener ítems relacionados con diferentes instancias de un proceso o ítems de diferentes procesos. El manejador de la Worklist podría estar interactuando con diferentes motores. La interface entre una aplicación cliente de Workflow y el motor de Workflow debe ser lo suficientemente flexible en los siguientes puntos:

1. Identificadores de procesos y actividades.
2. Estructuras de datos.
3. Diferentes alternativas de comunicación.

El esquema de la figura A.14 muestra las componentes que participan en esta interface.

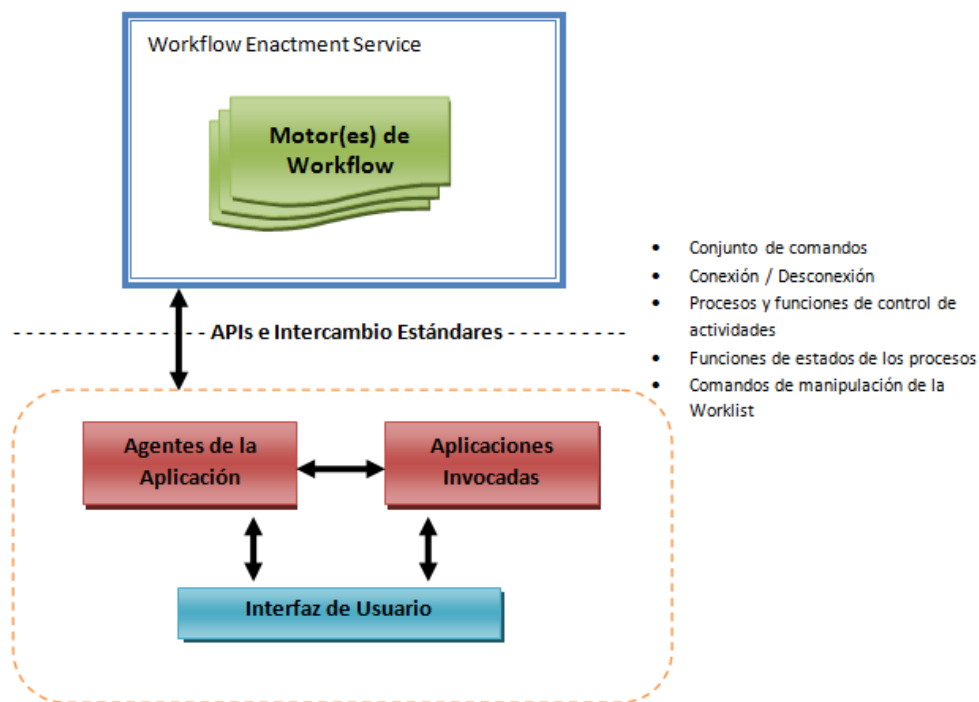


Figura A.14. Interface del Workflow con aplicaciones clientes.

A.6.3.10. Aplicaciones Invocadas (interface 3)

El diagrama de la figura A.15 muestra el alcance de esta interface, la cual está pensada para interactuar con agentes de una aplicación, o con una aplicación entera propiamente dicha.

Dichas aplicaciones deben estar orientadas con el contexto general de un sistema de Workflow, es decir, deben poder interactuar directamente con el motor de Workflow.

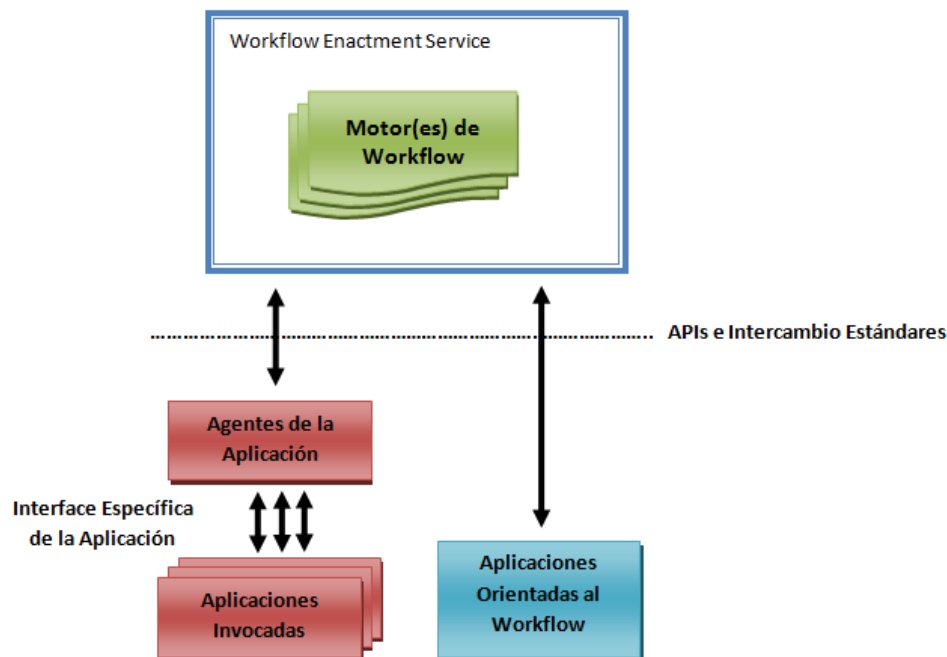


Figura A.15. Aplicaciones Invocadas.

La aplicación invocada es manejada localmente por un motor de Workflow, 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 los datos. La aplicación que se invoca puede ser local al motor de Workflow, 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).

Los detalles semánticos y sintácticos de esta interface se pueden encontrar en el documento elaborado por la WFM, donde se especifica en forma detallada esta interface.

A.6.3.11. Interoperabilidad del Workflow. Workflow Enactment Services Heterogéneos

El trabajo de la WFM se ha enfocado en desarrollar varios escenarios de interoperabilidad.

La idea de estos escenarios es poder operar en diferentes niveles, desde las tareas más simples hasta las aplicaciones de Workflow con un nivel alto de interoperabilidad, con un intercambio completo de la definición de procesos, los datos relevantes y una vista común de esto entre los distintos ambientes de Workflow.

Existen cuatro modelos posibles de interoperabilidad, los cuales se describen a continuación. En las ilustraciones se usan cuadrados para representar las tareas o actividades; y con distintos sombreados para denotar las tareas coordinadas por Workflow Enactment Services individuales.

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.

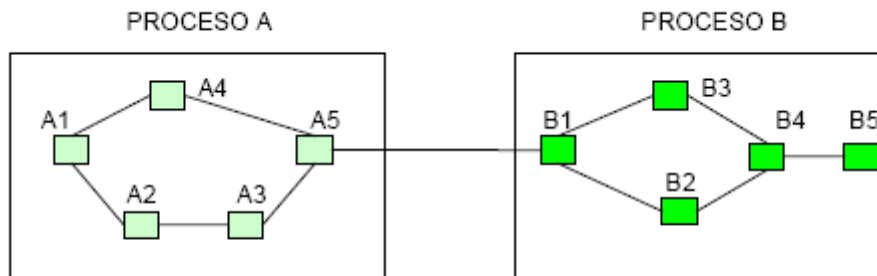


Figura A.16. Modelo Encadenado

Este modelo soporta la transferencia de un único elemento de trabajo (una instancia del proceso o actividad) entre los dos ambientes de Workflow, 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 de Workflow particular, ser completamente encapsulado como una única tarea, dentro de un proceso (superior) ejecutado en un dominio diferente. Existe una relación jerárquica entre el proceso 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, o no, ser permitida por cada producto particular.

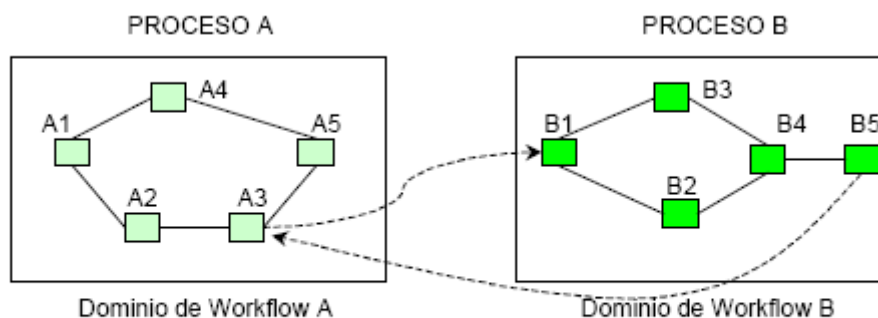


Figura A.17. Modelo de Sub-procesos anidados

En el diagrama se muestra un dominio de Workflow A, el cual tiene una actividad A3 que contiene completamente al proceso B del dominio B. Cuando el proceso B finaliza se le devuelve el control al proceso A.

Escenario 3: Sincronización en Paralelo

Este modelo le permite a dos procesos operar independientemente, posiblemente a través de Workflow Enactment Services 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.

En el diagrama siguiente la sincronización es mostrada entre la actividad A3 del proceso A y la actividad B4 del proceso B.

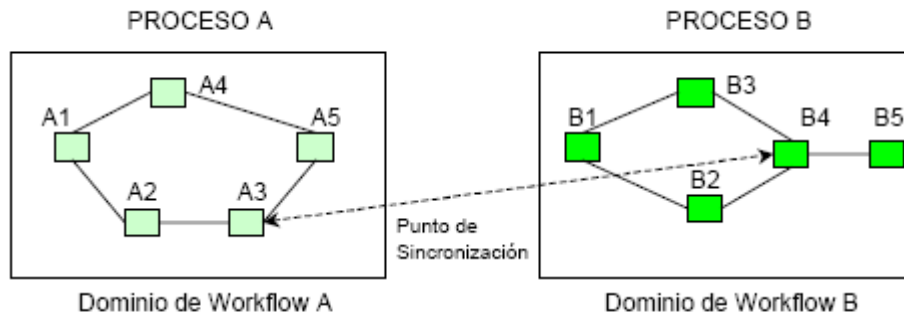


Figura A.18. Modelo de sincronización paralela

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 Workflow entre las diversas instancias de los procesos.

Escenario 4: Conexión No Discreta (Peer-to-Peer)

Este modelo permite un ambiente completamente mezclado. El diagrama indica la composición de un proceso C, el cual incluye actividades que pueden ser ejecutadas por múltiples motores de Workflow, formando un dominio compartido. Las actividades C1, C2 y C5 podrían ser manejadas por el servidor A y el resto por el B.

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 de Workflow cuando es necesario.

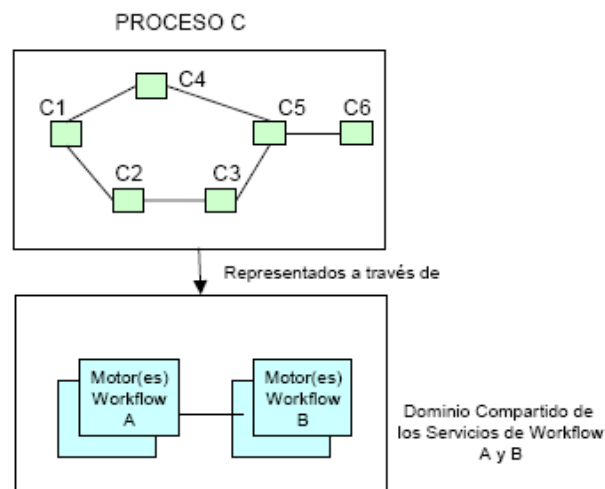


Figura A.19. Modelo PEER-TO-PEER.

Este escenario requiere que ambos motores de Workflow 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 Workflow también deben ser pasados entre varios motores.

A.6.3.12. Funciones de Interoperabilidad WAPI (interface 4)

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.
- ✓ Soporte en tiempo de ejecución para el intercambio de varios tipos de información de control y para la transferencia de los datos relevantes del Workflow y/o de las aplicaciones, entre los distintos Enactment Services.

La figura A.20 muestra la forma general del intercambio y control de flujo entre sistemas de Workflow heterogéneos.

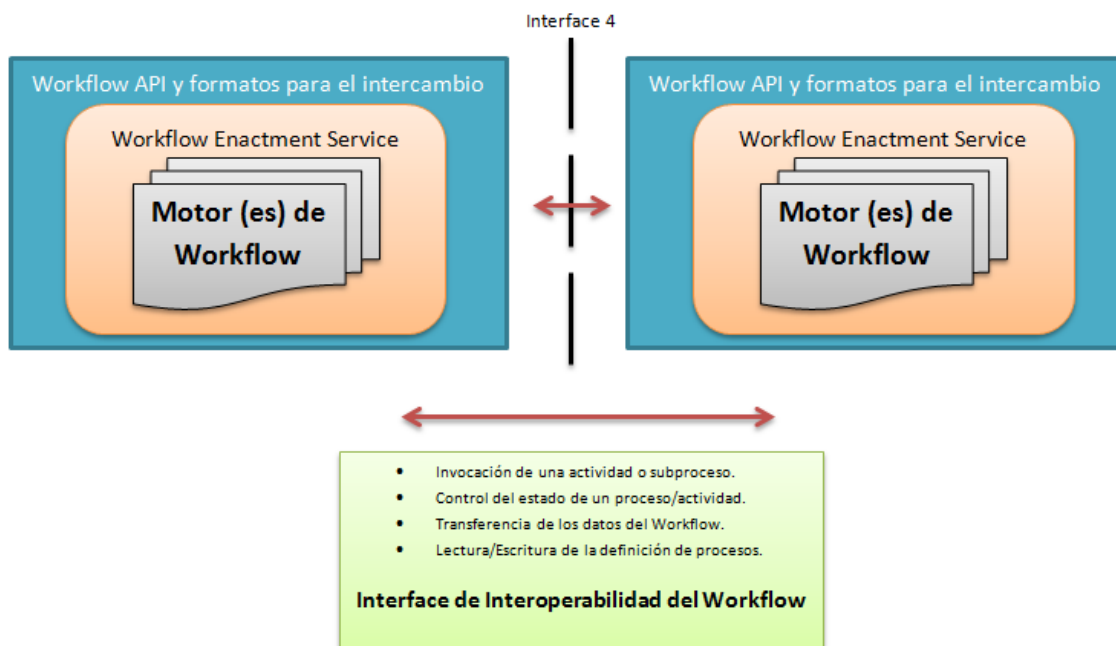


Figura A.20. Funciones de Interoperabilidad WAPI.

Uso de la definición de procesos a través de múltiples dominios:

Ambos Enactment Services pueden interpretar una definición de proceso común, por ejemplo generado con una misma herramienta. Esto permite a ambos ambientes compartir una única vista de la definición de los procesos y sus atributos. Esto permite, potencialmente, a cada motor de Workflow transferir la ejecución de actividades o subprocesos a un conjunto heterogéneo de motores de Workflow dentro del mismo contexto.

Este método es aplicable en el escenario 4 descrito anteriormente, donde distintos sistemas cooperan al mismo nivel, aunque puede también ser empleado en escenarios más simples.

En el caso en que no sea posible tener una vista común, un método alternativo podría ser exportar los detalles de un subconjunto de la definición de procesos, como parte del intercambio que se hace en tiempo de ejecución.

Ahora, si los dos casos anteriores no son factibles, la interoperabilidad se restringe al método de Gateway. En el cual (típicamente un subconjunto de) nombres de objetos y atributos son mapeados entre los dos ambientes por medio de una aplicación que interactúa mediante Gateway. En el caso más simple, los dos Enactment Services por separado usan su propio formato para la definición de procesos y existe algún tipo de mapeo entre los dos, manejado por un Gateway. Este método podría encajar en escenarios simples del tipo 1 y 2 o algunos ejemplos triviales del escenario 3.

Control de Interacciones en Tiempo de Ejecución:

En tiempo de ejecución, los llamados a las WAPI son usados para transferir el control entre los motores de Workflow para representar sub-procesos o actividades individuales sobre un motor específico. En el caso en que ambos motores soporten un nivel común de llamados a las WAPI y tengan una vista común de los objetos de la definición de procesos, la transferencia puede ser realizada directamente entre los motores de Workflow.

Pero en el caso en que no se pueda tener lo anterior, los llamados a las WAPI pueden ser usados para construir una función de Gateway que permita la interacción entre distintos motores, mapeando los diferentes objetos y vistas de datos entre ellos. Esto se ilustra en el diagrama de la figura A.21:

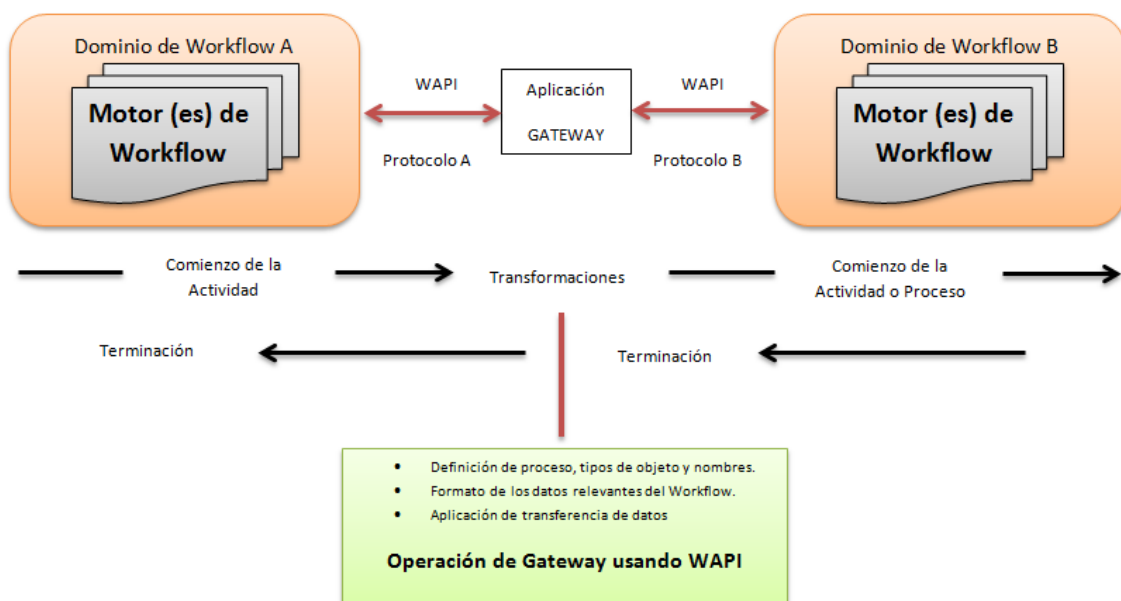


Figura A.21. Control de Interacciones en Tiempo de Ejecución.

El diagrama ilustra los principales principios de una operación Gateway. Dependiendo del escenario particular de interacción, una actividad individual de un dominio A puede ser mapeada como una única actividad o como un nuevo proceso/subproceso en el segundo dominio B.

Una vez que las actividades comienzan a representarse sobre distintos (subordinado) motores, la interacción de las aplicaciones Workflow de los clientes con los motores originales (por ejemplo para consultar los estados de las actividades) puede necesitar “referenciar” al motor subordinado. Algunas operaciones pueden necesitar ser encadenadas a través de distintos motores. Son necesarios eventos de notificación del estado de las actividades y su terminación, para que el usuario tenga el conocimiento de cómo se desarrollan las actividades.

A.6.3.13. Interface para la Administración y monitoreo (interface 5)

El propósito de esta interface es permitir una vista completa del estado del Workflow, además de poder realizar auditorías sobre los datos del sistema. El diagrama de más abajo ilustra como una aplicación de administración independiente, interactúa con los distintos dominios del Workflow. Es posible implementar otros posibles escenarios, como por ejemplo tener la aplicación de administración y monitoreo dentro del propio Enactment Services.

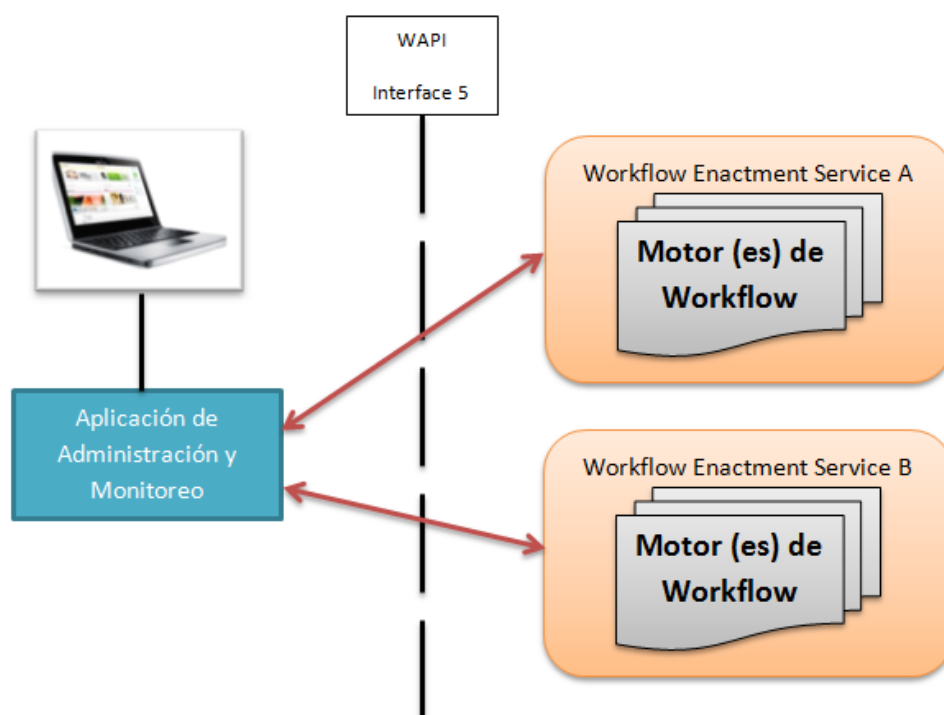


Figura A.22. Interface para la Administración y monitoreo.

A.7. Estructura WAPIs, protocolos y su conformidad

A.7.1. WAPIs - Descripción general funcional de la API

El WAPIs está previsto como un conjunto de llamadas a la API y los relacionados formatos de intercambio que pueden ser agrupados según sea necesario para dar soporte a cada una de las cinco interfaces funcionales mencionadas. Según la siguiente agrupación, entre las operaciones ya identificadas a través de estas 5 interfaces, se incluyen:

1. Llamadas API

- Establecimiento de sesión.
- Operaciones sobre definiciones Workflow y sus objetos.
- Funciones de control de procesos.
- Funciones supervisoras del control de procesos.
- Funciones de estado de los procesos.
- Funciones de gestión de actividad.
- Operaciones de manipulación de datos.
- Funciones de manejo de: Worklist; Workitem.
- Operaciones de gestión de Usuarios
- Operaciones de gestión de Roles.
- Operaciones de gestión de Auditorías.
- Operaciones de control de recursos.

2. Funciones de intercambio de datos: Se espera que los formatos de intercambio cubran:

- Transferencia de un proceso de definición.
- Transferencia de datos relevantes de un Workflow.

3. Estructura y nombramiento de una llamada API:

Las llamadas API, en primer lugar serán definidas, en términos de sus operaciones lógicas, los tipos de datos sobre los cuales ellas operan (llamadas a parámetros) y las estructuras de datos referenciadas por los tipos de datos.

A.7.2. Protocolo soporte de la WAPIs

Las llamadas WAPI serán capaces de funcionar en dos diferentes escenarios según los tipos de interconexión:

1. Cuando una interfaz WAPIs se presenta como una función de promulgación de servicio Workflow (por ejemplo, como rutinas incorporadas en una aplicación cliente o aplicación agente), las asignaciones específicas se pueden utilizar para codificar la llamada y los parámetros asociados a los protocolos utilizados para comunicarse con los motores Workflow.

2. Cuando un interworking directo entre distintos productos se presenta (por ejemplo la interoperabilidad entre los diferentes motores Workflow), un

protocolo común será necesario. Este protocolo requerirá un mapeo de llamadas WAPIs en una o, probablemente, varios protocolos interworking.

Los detalles de uso del protocolo dentro del WAPIs son para un estudio más detenido, sin embargo es de esperar que los mapeos WAPIs mencionados sean desarrollados por importantes entornos de comunicaciones. El soporte inicial para la integración de aplicaciones cliente e interoperabilidad de los Workflows pueden conseguirse utilizando el criterio 1 (con protocolos específicos); sin embargo, este enfoque tiene algunas limitaciones inherentes como el desarrollo de protocolos apropiados usando especificaciones, que es un claro requisito en el mediano plazo.

A.7.3. Principios de la conformidad: ¿Qué quiere decir conformidad?

Conformidad se definirá en contra de cada área funcional correspondiente a cada una de las 5 interfaces, de modo que los proveedores de productos puedan ofrecer una interfaz para las interworking en una o más áreas, pero no tengan que implementar las 5 funciones para lograr interoperabilidad.

Para cada interfaz se espera que conformidad se clasifique en varios niveles, proporcionando un nivel mínimo de interoperabilidad en el nivel 1. En el caso de las funciones de la interfaz 4, esto será especialmente importante debido al alto potencial de complejidad de la interoperabilidad en los servicios Workflow. Los productos que logren conformidad en un nivel determinado, se espera que inter funcionen con productos de conformidad en cualquier nivel inferior o igual al suyo.

Así, la conformidad deberá ser examinada por separado en términos de la API y el uso de sus protocolos. Es probable que algún tipo de matriz deba ser construida indicando las funciones de la API en un nivel específico y el protocolo compatible para lograr el interfuncionamiento con otros productos.

A.7.4. Interoperabilidad Clasificaciones y Niveles de Conformidad

Los niveles de conformidad serán desarrollados para ayudar en la clasificación de productos interoperables.

El alcance potencial de la interoperabilidad de los Workflows y la integración de aplicaciones es muy amplio, y por otro lado, desarrollar una completa gama de APIs y formatos de intercambio para abarcar todos los posibles escenarios de interoperabilidad es una tarea importante. Por estas razones, se considera esencial que un conjunto de escenarios de interoperabilidad sean desarrollados, desde lo simple a lo complejo, de manera que las interfaces de los escenarios más sencillos puedan ser desarrolladas con anterioridad. Esto permitirá que algunos de los beneficios de los sistemas interoperables sean descubiertos e implementados en el corto plazo, en las interfaces de menor complejidad, dejando un mayor desarrollo para desarrollar en las interfaces más complejas. Diversos niveles de la conformidad se pueden definir para agrupar particulares APIs, formatos de intercambio y los protocolos de soporte necesarios para satisfacer los escenarios específicos de interoperabilidad.

El resto de esta sección consta de una clasificación de los niveles de conformidad.

A.7.4.1. Definición de herramientas - Incorporación de software Workflow

Objetivo: permitir la elección por separado de los productos para herramientas de desarrollo (modelación, definición, etc.) y los servicios Workflow en tiempo de ejecución o la prestación de servicios para permitir el almacenamiento y la recuperación en una definición de proceso.

Interoperabilidad: basada en la interfaz de importación/exportación de la definición de proceso. La definición de proceso se exporta por la herramienta de definición y se importa por la promulgación de software del Workflow.

Conformidad con los niveles: sobre un mínimo conjunto de objetos de definiciones de procesos con extensiones opcionales para atender definiciones de procesos más sofisticadas. Formatos de intercambio de archivos y detalles de las llamadas a la API a ser discutidas.

A.7.4.2. Aplicación cliente interoperando con la promulgación de servicios de Workflows

Objetivo: (1) Permitir la construcción de un manejador común de la Worklist para proporcionar una gestión de la Worklist por uno o varios sistemas Workflow. Esto permite la prestación de diferentes servicios de Workflow dando la apariencia para el usuario final de un único servicio; (2) Dar soporte a una simple interacción entre dos servicios de Workflow controlados desde el entorno de un escritorio en común.

Interoperabilidad: basada en el apoyo a las llamadas WAPIs y formatos de intercambio de la interfaz 2.

Conformidad con los niveles: de apoyo a diversos grados de sofisticación en la aplicación cliente. Los detalles han de examinarse.

Protocolo de opciones: a determinar.

A.7.4.3. Herramienta de Integración y Aplicaciones

Objetivo: (1) Permitir que las aplicaciones o herramientas sean habilitadas por los Workflow de manera estandarizada (por ejemplo, para interactuar con un motor de Workflow a través de la actividad de las funciones de control, etc.); (2) Permitir el desarrollo de agentes de aplicación estandarizados.

Interoperabilidad: basada en el soporte para el subconjunto de llamadas WAPIs que manejan las invocaciones a las aplicaciones y el acceso a los datos relevantes de los Workflows.

Conformidad niveles: a tratar.

A.7.4.4. Interoperabilidad de servicios Workflow

Objetivo: (1) Apoyar el desarrollo de aplicaciones automatizadas de procesos utilizando diferentes productos de software Workflow; (2) Posibilitar a los ya

existentes Workflows el intercambio de aplicaciones Workflow o de solicitud los datos pertinentes.

Interoperabilidad: basada en el apoyo a las llamadas WAPIs y formatos de intercambio, ya sea utilizando la puerta de enlace o la actividad directa de interfaces.

Conformidad con los niveles de apoyo y protocolo: a discutir.

A.7.4.5. Administración de un Workflow común

Objetivo: Dar soporte a la gestión común, la administración y funciones de auditoría a través de varios productos de gestión de Workflow.

Interoperabilidad: basada en el apoyo a las llamadas WAPIs de la interfaz 5, para permitir que la administración y funciones de monitoreo sean soportadas por una común aplicación de gestión.

Conformidad con los niveles y el uso del protocolo: por definir.

ANEXO B: Tecnologías de los Web Services

Este anexo extiende las definiciones, características, tecnologías y arquitecturas expuestas en el capítulo 4 en relación a los Web Services.

La composición del anexo es la siguiente. La sección 1 define la Web Service y los contextualiza en el mundo de la computación dando ventajas y desventajas de los mismos. La sección 2 realiza una descripción de cada tecnología (protocolos, estándares, etc.) que tengan relación con los Web Services. Por su parte, la sección 3 introduce y desarrolla los conceptos necesarios para establecer la correspondencia entre los Web Services y la computación distribuida. Finalmente la sección 4 da los aspectos arquitecturales de los Web Services.

B.1. Introducción a los Web Services

B.1.1. ¿Qué es un Web Service?

El término Web Services se utiliza muy a menudo hoy en día, aunque no siempre con el mismo significado. Sin embargo, los conceptos y tecnologías subyacentes son en gran medida independientes de la forma en que puedan ser interpretadas. Las definiciones existentes para el concepto de Web Services, van desde la muy genérica y con todo incluido a la muy específica y restrictiva.

A menudo, un Web Service es visto como una solicitud de acceso a otras aplicaciones a través de la Web, pero esta es una definición muy abierta, en virtud de la cual, cualquier cosa que tenga una dirección URL (Uniform Resource Locator, es decir localizador uniforme de recursos) es un Web Service. Puede incluir, por ejemplo un script CGI, o también puede referirse a un programa accesible a través de la Web.

Entonces, empecemos por definir Web Services como un estándar de comunicación entre procesos y/o componentes, diseñado para ser multiplataforma y multilenguaje, es decir, no importa en qué lenguaje esté programado un Web Service como ser Visual Basic, C# o java, o en qué plataforma esté corriendo, ya sea Windows, UNIX o Linux éstos serán accesibles y utilizables por otras aplicaciones desarrolladas en otras plataformas o lenguajes de programación. Es decir, diversas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los Web Services para intercambiar datos en redes de ordenadores como Internet. Esta interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y, esencialmente, W3C son los comités responsables que se encargan de la arquitectura y la reglamentación de los Web Services.

Antiguamente se utilizaban otros estándares como DCOM (Distributed Component Object Model) introducido por Microsoft e implementado por otras plataformas, y CORBA (Common Object Request Broker Architecture) introducido por el OMG (Object Management Group) e implementado en distintas plataformas, incluido Windows. Estos estándares tenían bastantes problemas de configuración, especialmente en entornos en que se encontraban firewalls de por medio en los cuales era imposible (debido a estándares de seguridad de muchas compañías) habilitar ciertos puertos de comunicación para que estos componentes funcionaran. Es por eso que la preferencia por utilizar el puerto 80 de HTTP, que normalmente se encuentra habilitado en la mayoría de los servidores y firewalls debido al uso de navegadores y servidores Web, no traería mayores complicaciones el uso de una tecnología que utilice este protocolo y puerto de TCP/IP. La gran ventaja que trae el protocolo HTTP es su esquema de mensajes especialmente diseñado y optimizado para ser utilizado en redes como Internet, a diferencia de las viejas tecnologías como DCOM o CORBA que necesitaban un tipo de red más estable y local (LAN). Es por ello que HTTP es el protocolo preferido para el transporte de mensajes de los Web Services. HTTP se tratará más detalladamente en la Sección 2.1.1.

Visto esto, los Web Services pueden pensarse fundamentalmente como un cambio de las reglas del comercio Web. Conectan unas aplicaciones a otras a través de puntos distantes del mundo, transportando grandes cantidades de data más

eficientemente y más barato que nunca antes. El resultado es una más rápida, mejor y más productiva comunicación tanto para los negociantes como los consumidores.

Se centran en una interacción orientada a software más que una interacción orientada a humanos con data textual y gráfica. Las aplicaciones basadas en Internet necesitan buscar, acceder y actuar automáticamente con otras aplicaciones basadas en Internet. Por ello, los Web Services:

- a) Mejoran el uso de Internet al hacer posible la comunicación programa-a-programa (o aplicación-a-aplicación).
- b) Son aplicaciones Extensible Markup Language (XML – Sección 2.1.2) traducidos a programas, objetos, o bases de datos, o a funciones de negocios.
- c) Los estándares de Web Services: definen el formato del mensaje, especifican la interfaz a través de la cual un mensaje se envía, describen las convenciones para estructurar los contenidos de un mensaje dentro y fuera de los programas que llevan a cabo el servicio, y definen los mecanismos para publicar y descubrir interfaces Web Services.
- d) Pueden correr en los clientes desktop y handheld acceder aplicaciones de Internet como sistemas de reservaciones y sistemas de rastreo de órdenes.
- e) Pueden usarse para la integración business-to-business (B2B), conectando aplicaciones que son ejecutadas por varias organizaciones en la misma cadena del suministro (Sección 3.3).
- f) Pueden resolver el amplio problema de la Enterprise Application Integration (EAI), conectando desde múltiples aplicaciones de una sola organización hasta las múltiples aplicaciones tanto dentro y como fuera del firewall (Sección 3.2).

En todos los casos, las tecnologías de Web Services proporcionan el pegamento estándar que conecta diversos pedazos de software. Ahora revisemos otras definiciones con el objetivo de completar el panorama de los Web Services:

- ✓ *Según Gartner, "Web Services son componentes de software débilmente acoplados que se despachan sobre tecnologías estándar de Internet". Son aplicaciones comerciales modulares que se auto describen, que muestran su lógica de negocio como servicios sobre la Internet por medio de interfaces programables y que usan los protocolos de la Internet para proporcionar un forma de encontrarlos, de suscribirse a los mismos y de invocarlos.*
- ✓ *Una definición más precisa es provista por el consorcio UDDI (Universal Description, Discovery and Integration), donde los Web Services son caracterizados como aplicaciones de negocios auto-contenidas y modulares que han abierto, con una orientación a Internet, las interfaces basadas en estándares. Esta definición es más detallada, haciendo hincapié en la necesidad de cumplir con estándares de Internet. Además, se requiere que el servicio sea abierto, que básicamente significa que se ha publicado una*

interfaz que puede ser invocada a través de Internet. A pesar de esta aclaración, la definición no es aún lo suficientemente precisa. En este sentido, no queda claro que se entiende por un sistema modular, auto-contenido.




B.1.2. Web Services por la W3C

Un paso más en el perfeccionamiento de la definición de Web Services es el proporcionado por la World Wide Web Consortium (W3C, que como ya vimos es un organismo que se encarga de desarrollar gran parte de los estándares de internet): *“Un Web Service es una aplicación software identificada mediante una URI (Uniform Resource Identifier), cuyas interfaces (y uso) son capaces de ser definidas, descritas y descubiertas mediante artefactos XML, y soportar interacciones directas con otras aplicaciones software usando mensajes basados en XML y protocolos basados en Internet”.*

La definición que la W3C da, alude a la forma en que los Web Services deben trabajar. Subraya que los Web Services deben ser capaces de ser "definidos, descritos y descubiertos", lo que aclara el hecho de ser "accesibles" y hace más concreto el concepto de "interfaces basadas en estándares orientados a Internet". Asimismo, dispone que los Web Services deben ser "servicios" similares a los convencionales en el middleware. No sólo deben estar en funcionamiento, sino que deben describirse y, en algún sentido, publicitarse de modo que sea posible la comunicación con los clientes que interactúan con ellos. Esta interpretación hace que pongamos mucho énfasis en principio en la necesidad de comprender "middleware" como el primer paso hacia la comprensión de los Web Services. Sección 3.1. La W3C también afirma que XML es parte de la "solución". En efecto, el XML es tan popular y ampliamente utilizado hoy en día que, así como HTTP y los servidores Web, se puede considerar como parte de la tecnología web. No cabe duda de que el XML es (y lo seguirá siendo) el formato de datos más utilizado para muchas interacciones en la Web.

Pero al analizar detenidamente esta definición, además de ser un tanto complicada, uno llega a la conclusión de que es tan genérica que millones de cosas pueden ser un Web Service. Sin embargo, cuando los desarrolladores hablamos de Web Services nos estamos refiriendo a tecnologías muy concretas, al menos la gran mayoría de las veces.

Por ello, una definición alternativa podría ser la siguiente: Un Web Service es un componente software que se basa en las siguientes tecnologías:

-  Un formato que describa la interfaz del componente (sus métodos y atributos) basado en XML. Por lo general este formato es el WSDL (Web Service Description Language).
-  Un protocolo de aplicación basado en mensajes y que permite que una aplicación interactúe (use, instancia, llame, ejecute) al web service. Por lo general este protocolo es SOAP (Simple Object Access Protocol).
-  Un protocolo de transporte que se encargue de transportar los mensajes por internet. Por lo general este protocolo de transporte es HTTP (Hiper-Text

Transport Protocol) que es exactamente el mismo que usamos para navegar por la Web.

Ahora bien, antes de ver con precisión algunas de las definiciones mencionadas y sus connotaciones, tengamos en cuenta que aún existen otras definiciones más específicas. Por ejemplo, en el diccionario técnico en línea Webopedia, un Web Service se define como "una manera de integrar las aplicaciones basadas en Internet utilizando XML, SOAP, WSDL, UDDI y las normas abiertas de protocolo de Internet a través de una columna vertebral. XML se utiliza para etiquetar los datos, SOAP se utiliza para transferir los datos, WSDL se utiliza para describir los servicios disponibles, y UDDI se utiliza para la inclusión de qué servicios están disponibles". En este tipo de definiciones se mencionan las normas específicas que podrían utilizarse para la realización de, e interacción con, un Web Service. Estas son las principales normas de hoy en día para los Web Services. En efecto, muchas aplicaciones que son "puestas a disposición de otras aplicaciones" las llevan a cabo a través de SOAP, WSDL, UDDI, y otros estándares Web. Sin embargo, estas normas no constituyen la esencia de la tecnología de Web Services: los problemas que están en los Web Services son los mismos independientemente de las normas utilizadas.

Entonces, los Web Services no son por tanto aplicaciones con una interfaz gráfica con la que las personas puedan interaccionar, sino que son software accesible en internet (o en redes privadas que usen tecnologías internet) por otras aplicaciones. De esta forma podemos desarrollar aplicaciones que hagan uso de otras aplicaciones que estén disponibles en internet interaccionando con ellas.

Un típico ejemplo podría ser un Web Service al que se le pudiese preguntar por una empresa y que nos retornase en tiempo real el valor al que están cotizando las acciones de dicha compañía. De esta forma cualquier aplicación (ya sea web o de escritorio) que quiera mostrar esta información sólo tendría que solicitarla a través de internet al Web Service cuando la necesitase.

Otro ejemplo de Web Service podría ser uno que al pasarle el nombre de una ciudad, nos devolviese la temperatura, humedad, y otras condiciones climatológicas de la misma.

Es importante entonces remarcar que los Web Services no son la solución mágica universal, sino una tecnología apropiada para resolver ciertos problemas. Básicamente los Web Services permiten que diferentes aplicaciones, realizadas con diferentes tecnologías, y ejecutándose en toda una variedad de entornos, puedan comunicarse e integrarse, lo cual es muy importante, y ¿por qué? ... tomemos un poco de perspectiva.

B.1.3. Un poco de perspectiva

El coste de desarrollar software siempre ha sido altísimo, y la diversidad de las plataformas una realidad desde el inicio de la informática. De hecho conforme más complejas fueron las aplicaciones que las empresas demandaban, más caras era desarrollarlas. Para enfrentarse a esta situación se han seguido diferentes líneas todas ellas encaminadas a reutilizar las aplicaciones ya desarrolladas.

Una de estas propuestas fue (y es) la estandarización de lenguajes de programación de tal forma que si cualquiera de nosotros escribía un programa en C, solo necesitaríamos un compilador de C en la plataforma específica en la que quisiéramos ejecutar la aplicación. Esto en la realidad ha sido (y es) difícil de hacer funcionar, porque en seguida surgieron pequeñas diferencias y extensiones de C que hacían difícil ‘transportar’ una aplicación entre diferentes plataformas.

Otra posibilidad ha sido la que ha desarrollado Sun con Java. Se programa para una plataforma, pero para una plataforma virtual, en este caso para la máquina virtual Java, y en cada plataforma real se implementa una máquina virtual de Java, que será la encargada de ejecutar las aplicaciones escritas en Java. Las implementaciones son específicas de cada plataforma pero al ser todas las máquinas virtuales exactamente la misma, los programas escritos en Java deben poder ejecutarse sin ningún problema en todas las plataformas que tengan una máquina virtual de Java. Esta apuesta ha demostrado ser muy útil y funcionar bastante bien.

Sin embargo, ¿qué pasa si tenemos varias aplicaciones ya desarrolladas en lenguajes propietarios o en plataformas específicas y queremos que interaccionen entre ellas? El coste de elegir a posteriori un único lenguaje o plataforma y migrarlo al mismo es descabellado en la mayoría de las situaciones, y es aquí donde los Web Services, así como otras tecnologías, pueden sernos de una grandísima utilidad.

Con los Web Services podemos reutilizar desarrollos ya utilizados sin importar la plataforma en la que funcionan o el lenguaje en el que están escritos. Los Web Services se constituyen en una capa adicional a estas aplicaciones de tal forma que pueden interaccionar entre ellas usando para comunicarse tecnologías estándares que han sido desarrolladas en el contexto de internet.

B.1.4. ¿Para qué sirven los Web Services?

El desarrollo y la programación de sistemas orientado a objetos o componentes nos ha llevado a lo largo del tiempo a tener la necesidad de reutilizarlos en diferentes proyectos. Ya sean componentes desarrollados por nosotros o componentes desarrollados por terceras partes. Hasta la existencia de los Web Services esta reutilización nos limitaba a un lenguaje de programación o a una plataforma en particular. Por lo tanto, el uso de los Web Services nos facilitará la reutilización de funciones de una aplicación en distintas plataformas o lenguajes ya sea para un uso personal en distintos proyectos, para comercializarlos o adquirir prestaciones de terceros. De la misma forma que anteriormente incluíamos en nuestras aplicaciones referencias a otras librerías como ser DLLs o componentes ActiveX, ahora podremos referenciar funciones que se estarán ejecutando en otra computadora o servidor sin importarnos en qué están programados ni en qué plataforma están corriendo.

Uno de los ejemplos más comunes del uso de los Web Services se encuentra en los sitios web de comercio electrónico, los cuales hacen uso de un Web Service para validar los datos de las tarjetas de crédito de sus clientes. Normalmente este Web Service es provisto por algún banco o entidad financiera que actúa como intermediario entre el comercio y las tarjetas de crédito. Otro ejemplo podría ser que necesitamos usar el corrector ortográfico del Microsoft Word desde un sitio web que creamos en ASP.NET. Ahora bien, esto es algo que podemos hacer a través de los Visual Studio

2005, pero para ello necesitaremos tener instalado el Microsoft Word en el servidor Web. Supongamos que por alguna razón no se nos permite instalar el Microsoft Word en el servidor Web, pero disponemos de un servidor de aplicaciones en el que tenemos control total y allí podemos instalar el MS Word. Para poder utilizar el MS Word que está instalado en otro servidor desde nuestra aplicación web podríamos crear un Web Services en el servidor de aplicaciones, el cual expone un Web Método público que se encargue de ejecutar el corrector ortográfico de MS Word. Teniendo esto podremos utilizar esta funcionalidad desde nuestra aplicación Web a través de un Web Service sin haber instalado MS Word en el servidor Web. Para poder reutilizar bien los componentes y objetos desarrollados era necesario un lenguaje de programación orientado a objetos. Ahí nace el lenguaje c# de la mano del .NET framework, que desde sus inicios se focalizó en proveer una herramienta como el Visual Studio, capaz de crear y consumir Web Services de la forma más rápida y sencilla tornando transparentes para el desarrollador protocolos y tipos de mensajes XML como WSDL y SOAP, los que describiremos a continuación.

B.1.5. Características generales de los Web Services

B.1.5.1. Ventajas de los Web Services

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los Web Services fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP, los Web Services pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos. Las especificaciones son gestionadas por una organización abierta, la W3C, por tanto no hay secretismos por intereses particulares de fabricantes concretos y se garantiza la plena interoperabilidad entre aplicaciones.

B.1.5.2. Inconvenientes de los Web Services

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.

- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

B.1.6. Razones para crear un Web Service

La principal razón para usar Web Services es que se basan en HTTP sobre TCP (Transmission Control Protocol) en el puerto 80. Dado que las organizaciones protegen sus redes mediante firewalls -que filtran y bloquean gran parte del tráfico de Internet-, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los Web Services utilizan este puerto, por la simple razón de que no resultan bloqueados.

Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran ad hoc y poco conocidas, tales como EDI (Electronic Data Interchange), RPC (Remote Procedure Call), u otras APIs.

Una tercera razón por la que los Web Services son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el Web Service y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más utilizada.

Se espera que para los próximos años mejoren la calidad y cantidad de servicios ofrecidos basados en los nuevos estándares.

B.2. Conceptos Generales (Definiciones)

B.2.1. Protocolos, Estándares y Lenguajes utilizados por los Web Services

B.2.1.1. HTTP

El protocolo de transferencia de hipertexto HTTP (Hyper Text Transfer Protocol) es el protocolo usado en cada transacción de la Web (WWW). HTTP fue desarrollado por el consorcio W3C y la IETF, colaboración que culminó en 1999 con la publicación de una serie de RFC, 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 dan 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.
- ✓ Ten 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>

1. Se abre una conexión al host www.example.com, puerto 80 que es el puerto por defecto para HTTP.

2. Se envía un mensaje en el estilo siguiente:

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

Figura B.1. Mensaje HTTP GET

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 tu Host</h1>
(Contenido)
.
.
</body>
</html>
```

Figura B.2. Respuesta del Servidor

B.2.1.2. XML

XML (Extensible Markup Language) es un lenguaje utilizado para definir formatos de documentos o mensajes. Éstos están compuestos por Tags (palabras entre caracteres < y >). Estos tipos de documentos han sido aceptados y adoptados por la mayoría de los fabricantes de software para proveer extensibilidad de los datos debido a que no está atado a ninguna plataforma o lenguaje de programación. Podremos ver que el XML es muy similar al HTML, pero sin embargo el XML es más estricto ya que por cada tag abierto deberemos tener un Tag que lo cierre. El HTML, sin embargo, no es tan estricto debido a que los navegadores de hoy en día, como el Internet Explorer, tienen la inteligencia de adivinar si un tag no fue cerrado correctamente.

De todos modos, si no cerramos un tag en un documento XML éste no podrá ser interpretado por los parsers (clases utilizadas para el manejo de documentos XML).

Otra diferencia importante entre los documentos XML y HTML es que los XML son sensibles a las mayúsculas y minúsculas mientras los HTML no lo son, por ejemplo, un Tag <cliente> no será igual a un tag <CLIENTE>.

```

<?xml version='1.0'?>
<schema
xmlns='http://www.w3.org/2001/XMLSchema'>
  <objeto>
    <propiedad atributo="valor">
  </propiedad>

```

Figura B.3. Ejemplo de documento XML.

Aparte de los Tags hay otras restricciones y validaciones que los documentos XML deberán cumplir, eso depende de un archivo de definiciones llamado DTD (Document Type Definitions) o de un archivo de esquemas de XML, los cuales podrán estar asociados con el documento XML. En estos archivos DTDs podremos definir si un elemento (también llamado nodo) deberá al menos aparecer una vez, una o más veces, una a ninguna vez. A su vez, cada elemento podrá tener uno o más atributos, eso depende también del archivo de definiciones o esquema. Estos documentos XML son la base de los documentos WSDL y SOAP, base de los Web Services.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Ventajas del XML

- ✓ Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- ✓ El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan bugs y se acelera el desarrollo de aplicaciones.
- ✓ Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

B.2.1.3. WSDL

WSDL (*Web Services Description Language*) es un documento XML que se utiliza para describir los mensajes SOAP y cómo estos mensajes son intercambiados. Es decir, supongamos que creamos un Web Services y queremos que otras aplicaciones lo utilicen, las otras aplicaciones deberán acceder a un documento WSDL en donde podrán conocer los métodos que expone nuestro Web Service y cómo acceder a ellos, es decir, cuáles son los nombres de los métodos y qué tipos de parámetros espera

cada uno de ellos.

El lenguaje WSDL se desarrolla en el Anexo C.

B.2.1.4. SOAP

SOAP (*Simple Object Access Protocol*) es el protocolo base de los Web Services.

Este protocolo está basado en XML y no se encuentra atado a ninguna plataforma o lenguaje de programación. A su vez, también es el protocolo más aceptado por la mayoría de las plataformas.

Si bien SOAP es un protocolo, éste no es exactamente un protocolo de comunicación entre mensajes como lo es el HTTP, por ejemplo. Básicamente, SOAP son documentos XML y necesitaremos utilizar algún otro protocolo para la transmisión de estos documentos como ser el protocolo HTTP o cualquier otro protocolo de comunicación capaz de transmitir textos. Los mensajes SOAP están compuestos por un tag principal llamado Envelope, que está dividido en una cabecera o Header y en un cuerpo o Body.

```
<?xml version='1.0' encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <miCabecera>miValor</miCabecera>
  </soap:Header>
  <soap:Body>
    <miMetodo>
      <miParametro>miValor</miParametro>
    </miMetodo>
  </soap:body>
</soap:Envelope>
```

Figura B.4. Ejemplo de mensaje SOAP

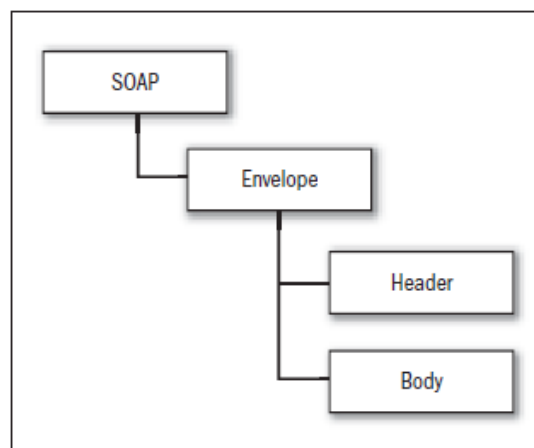


Figura B.5. La figura muestra las relaciones de los diferentes elementos de un documento SOAP.

Dentro del elemento Body estarán los elementos correspondientes al Web Method que queramos invocar y además podrá haber o no un elemento en común llamado Fault, que nos indicará que ha ocurrido un error y la razón de éste.

```

    <?xml version='1.0' encoding="utf-8"?>
    <soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
        <soap:Fault>
          <soap:faultcode>Client.Security</soap:faultcode>
          <soap:faultstring>Acceso denegado.</soap:faultstring>

          <soap:faultactor>http://miwebservice.com</soap:faultactor>
          <soap:detail>
            <miError>no se pudo acceder al archivo
prueba.txt</miError>

```

Figura B.6. Ejemplo del elemento Fault.

Serialización es el proceso de convertir un dato binario a una representación de texto usando caracteres del tipo ASCII, por ejemplo. Debido a que SOAP está basado en XML, éste a su vez no soporta el envío de cualquier carácter ya que algunos caracteres son utilizados para control de los mensajes, como por ejemplo el carácter cero sería utilizado para la finalización de una cadena de caracteres (String).

Si lo que queremos enviar como parámetro es un objeto complejo o un objeto que contiene una imagen, entonces los datos que por su naturaleza se encuentran en binario deberán ser transformados a una cadena de caracteres. Esta técnica es conocida como Encode, hay varias formas realizar una serialización y esta forma estará determinada por el atributo encoding que se especifica en la cabecera XML de un mensaje SOAP. Si vemos en el ejemplo mencionado anteriormente observaremos que éste utiliza el utf-8.

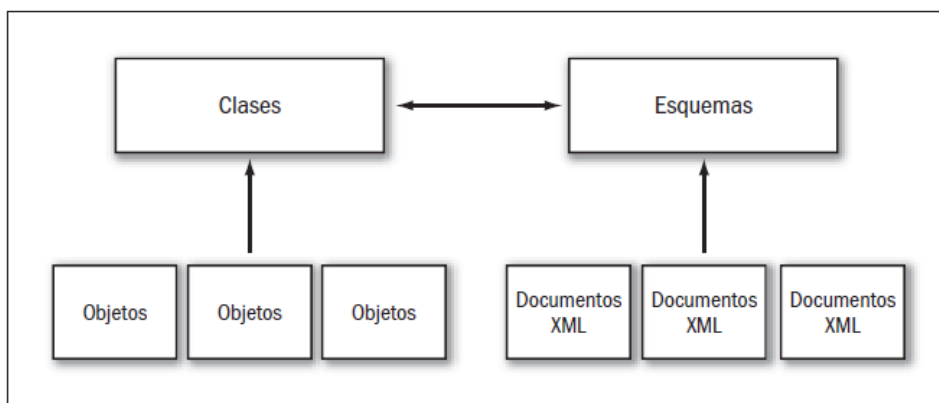


Figura B.7. Equivalencia de las clases y objetos luego de ser serializados donde las clases se convierten en esquemas y los objetos en documentos XML y viceversa.

La misma clase Proxy que nos es generada por el visual Studio para ser utilizada en la aplicación cliente que consumirá nuestro Web Service es la que se encarga de serializar los objetos antes de ser transmitidos por el protocolo http hacia el Web Method, como también de de serializar la respuesta. La figura 6 nos muestra todos los pasos del ciclo de vida de un Web Service.

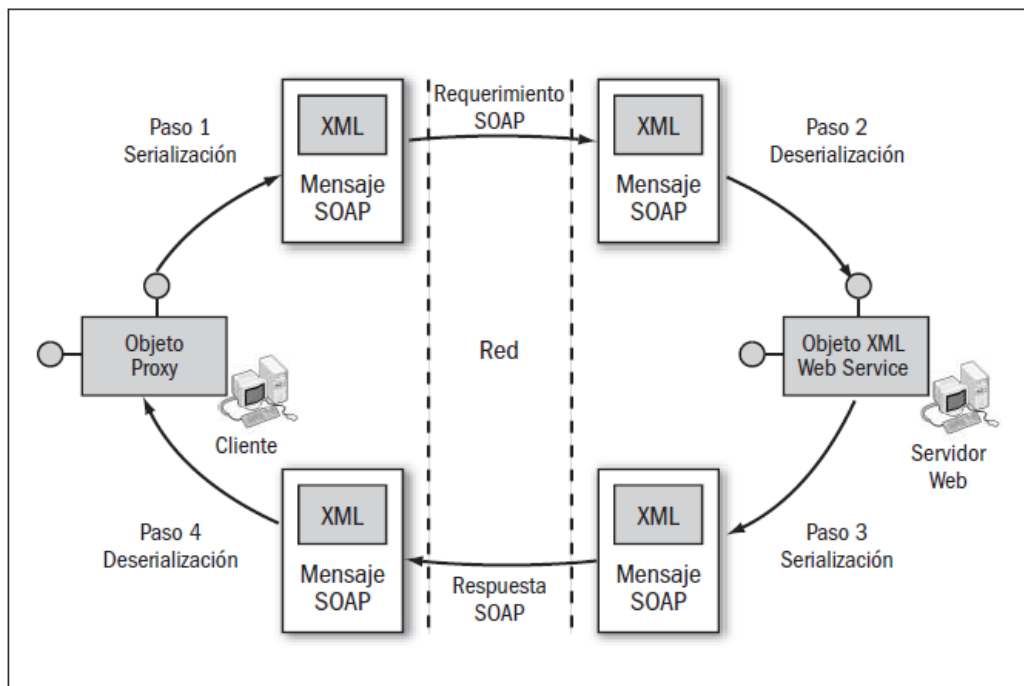


Figura B.8. Ciclo de vida de un Web Service en el que se serializa y se de-serializa en cada uno de los pasos

B.2.1.5. UDDI

UDDI es uno de los estándares básicos de los Web Services cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los Web Services del catálogo de registros. UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen.

El estándar UDDI, al igual que WSDL, se desarrolla en el Anexo C.

B.2.1.6. BPEL

Business Process Execution Language, WS-BPEL (en castellano, Lenguaje de Ejecución de Procesos de Negocio), es un lenguaje estandarizado por OASIS para la composición de Web Services. Está desarrollado a partir de WSDL y XLANG, ambos lenguajes orientados a la descripción de Web Services. Básicamente, consiste en un lenguaje basado en XML diseñado para el control centralizado de la invocación de diferentes Web Services, con cierta lógica de negocio añadida que ayuda a la programación en gran escala.

Propósito. La programación en gran escala generalmente se refiere a desarrollo de software de gran tamaño que involucra grandes procesos de desarrollo, evolución y mantenimiento. Por otro lado, la programación detallada se refiere a la construcción de componentes de software pequeños y autónomos. El desarrollo de BPEL nace de la necesidad de manejar lenguajes distintos entre la programación a gran escala y la programación detallada, ya que en su esencia ambos tipos de desarrollo requieren de distintos grados de comunicación con otros servicios.

El Lenguaje. BPEL es un lenguaje de orquestación, no un lenguaje coreográfico. La diferencia mayor entre ambos es el ámbito. Un modelo de orquestación provee un ámbito específicamente enfocado en la vista de un participante en particular (ej.: un modelo par-a-par). En cambio, un modelo coreográfico abarca todos los participantes y sus interacciones asociadas, dando una vista global del sistema. Las diferencias entre orquestación y coreografía están basadas en analogías: la orquestación describe un control central del comportamiento como un director en una orquesta, mientras que la coreografía trata sobre el control distribuido del comportamiento donde participantes individuales realizan procesos basados en eventos externos, como en una danza coreográfica donde los bailarines reaccionan a los comportamientos de sus pares.

A través de un documento XML BPEL, un analista de negocio es capaz de representar la lógica asociada y los elementos con los que se verá relacionado. Estos elementos serán Web Services y la lógica el proceso BPEL.

Si imaginamos un flujo de negocio determinado, con una entrada A y una salida Z, este se podría componer de muchos procesos internos que se lanzarían dependiendo de valores y respuestas anteriores. BPEL sería el encargado de orquestar todo el proceso ordenando qué proceso ejecutar (Web Service) y en qué momento.

Este lenguaje fue concebido por los grandes de la informática como lo son Oracle, BEA Systems, IBM, SAP y Microsoft entre otros.

Es un lenguaje de alto nivel que lleva el concepto de servicio un paso adelante al proporcionar métodos de definición y soporte para Workflow y procesos de negocio.

El enfoque sobre procesos de negocios modernos más el bagaje de los lenguajes WSDL y XLANG, guiaron a BPEL a adoptar los Web Services como su mecanismo de comunicación externa. Así las facilidades de mensajería BPEL dependen del uso del WSDL para describir los mensajes entrantes y salientes.

Adicionalmente a proveer facilidades para habilitar el envío y recepción de mensajes, el lenguaje de programación BPEL también posibilita:

- ✚ Un mecanismo de correlación de mensajes basado en propiedades.
- ✚ Variables del tipo XML y WSDL.
- ✚ Un modelo de lenguaje extensible de componentes para permitir escribir expresiones y consultas (queries) en múltiples lenguajes: BPEL soporta Xpath 1.0 predeterminadamente.
- ✚ Construcciones de programación estructurada incluyendo "if-then-elseif-else", "while", "sequence" (posibilita la ejecución de comandos en orden) y "flow" (posibilita la ejecución de comandos en paralelo).
- ✚ Un sistema de ámbito (scoping) que permite el encapsulamiento de la lógica con variables locales, manejadores de fallos, manejadores de compensación y manejadores de eventos.

- ✚ Ámbitos serializados para controlar los accesos a las variables.

El Diseño.

- Definir procesos de negocio que interactúan con entidades externas mediante operaciones de un Web Service definidas usando WSDL 1.1 y que se manifiestan a sí mismas como Web Services.
- Definir procesos de negocio utilizando un lenguaje basado en XML. No definir una interpretación gráfica de procesos o proveer de una metodología de diseño en particular.
- Definir una serie de conceptos de orquestación de Web Services que pretendan ser usados por vistas internas o externas de un proceso de negocio.
- Proveer sistemas de control jerárquicos y de estilo gráfico, que permitan que su uso sea lo más fusionado e inconsútil posible. Esto reduciría la fragmentación del espacio del modelado de procesos.
- Proveer funciones de manipulación simple de datos, requeridas para definir datos de procesos y flujos de control.
- Soportar un método de identificación de instancias de procesos que permita la definición de identificadores de instancias a nivel de mensajes de aplicaciones. Los identificadores de instancias deben ser definidos por socios y pueden cambiar.
- Brindar la posibilidad de la creación y terminación implícitas de instancias de procesos, como un mecanismo básico de ciclo de vida. Operaciones avanzadas de ciclo de vida como por ejemplo "suspender" y "continuar" pueden agregarse en futuras versiones para mejorar el manejo del ciclo de vida.
- Definir un modelo de transacción de largo plazo que se base en técnicas probadas tales como acciones de compensación y ámbito, de tal manera a brindar recuperación a fallos para partes de procesos de negocios de largo plazo.
- Usar Web Services como modelo para la descomposición y ensamblaje de procesos.
- Construir sobre estándares de Web Services (aprobados y propuestos) tanto como sea posible, de manera modular y extensible.

B.2.1.7. SOA

La Arquitectura Orientada a Servicios (en inglés Service Oriented Architecture - SOA-), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio. Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma estándar de exposición e invocación de servicios (en general Web Services, aunque no es exclusivo), lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

SOA define las siguientes capas de software:

- ✓ Aplicaciones básicas - Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad;
- ✓ De exposición de funcionalidades - Donde las funcionalidades de la capa aplicativas son expuestas en forma de servicios (Web Services);

- ✓ De integración de servicios - Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración;
- ✓ De composición de procesos - Que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio;
- ✓ De entrega - donde los servicios son desplegados a los usuarios finales.

De esta forma, SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

La consecuencia inmediata de lo que SOA es y sus características, hacen que hoy en día sea, sin ningún tipo de dudas, la arquitectura de desarrollo de sistemas más de moda. Esto porque básicamente se trata de un cambio de mentalidad a la hora de desarrollar un sistema y pensar cada uno de sus módulos como un servicio. Esto nos facilitará el uso de los Web Services. Entonces tendremos un conjunto de servicios que se comunicarán unos con otros intercambiando distintos tipos de datos e involucrando más de un servicio, eso depende de la operación que necesitemos realizar. Es necesario destacar que SOA no es un producto o una tecnología sino más bien una moderna forma de diseñar nuestras aplicaciones o sistemas. Para llevar a cabo esta forma de diseño encontraremos una gran variedad de productos y tecnologías disponibles, como ser la tecnología de Web Services y el producto Visual Studio (que nos ayudará a llevar a cabo soluciones basadas en la arquitectura SOA). Si miramos otras compañías que son competencia de Microsoft, como Sun o IBM, veremos que ellas también están realizando una inversión muy interesante en dar a conocer esta arquitectura y ayudar a implementarla cada uno a través de sus productos. Basta sólo con entrar a msdn.microsoft.com o developers.sun.com/channel/ y realizar una búsqueda por *Service Oriented Architecture* y obtendremos un montón de resultados sobre los cuales podría escribirse más de un libro. Los Web Services son, si bien no los únicos, los recursos más utilizados en esta arquitectura. Por eso, aprender a desarrollarlos y entenderlos nos ayudará mucho a pensar en servicios. SOA tiene 4 principios básicos que detallaremos a continuación:

- Límites: los servicios están demarcados por límites específicos y la única forma de comunicarse con ellos será a través de la tecnología o protocolos expuestos por ellos.
- Autonomía: cada servicio deberá comportarse de forma autónoma. Por ejemplo, esto quiere decir que si mi sistema hace uso de un servicio que consulta una base de datos, mi sistema no tendrá por qué saber ni conocer a qué tipo de base de datos se está conectando este servicio ni de qué manera lo hace. Así obtendremos un diseño totalmente desacoplado.
- Contratos: aquí se definen cómo serán utilizados los servicios y de qué manera intercambiarán los datos y mensajes.
- Políticas: cada servicio deberá definir las políticas de su uso, como por ejemplo, que se deberá utilizar el protocolo HTTP o que se requerirá el uso de transacciones para ser utilizado.

Definiciones SOA:

1. Servicio: Una función sin estado (existen servicios asíncronos en los que una solicitud a un servicio crea, por ejemplo, un archivo, y en una segunda solicitud se obtiene ese archivo), auto-contenida, que acepta una o más llamadas y devuelve las correspondientes respuesta mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. La tecnología concreta utilizada para prestar el servicio no es parte de esta definición.
2. Orquestación: Secuenciar los servicios y proveer la lógica adicional para procesar datos. No incluye la presentación de los datos. Coordinación.
3. Sin estado: No mantiene ni depende de condición pre-existente alguna. En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio.
4. Proveedor: La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor.
5. Consumidor: La función que consume el resultado del servicio provisto por un proveedor.

Diseño y desarrollo de SOA. La metodología de modelado y diseño para aplicaciones SOA se conoce como análisis y diseño orientado a servicios. La arquitectura orientada a servicios es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implementación. Para que un proyecto SOA tenga éxito los desarrolladores de software deben orientarse ellos mismos a esta mentalidad de crear servicios comunes que son orquestados por clientes o middleware para implementar los procesos de negocio. El desarrollo de sistemas usando SOA requiere un compromiso con este modelo en términos de planificación, herramientas e infraestructura.

Cuando la mayoría de la gente habla de una arquitectura orientada a servicios están hablando de un juego de servicios residentes en Internet o en una intranet, usando Web Services. Como ya vimos, existen diversos estándares relacionados a los Web Services (XML, HTTP, SOAP, WSDL, UDDI).

Hay que considerar, sin embargo, que un sistema SOA no necesariamente necesita utilizar estos estándares para ser "orientado a servicios" pero es altamente recomendable su uso.

En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones de SOA identifican la utilización de Web Services (empleando SOAP y WSDL) en su implementación, no obstante se puede implementar SOA utilizando cualquier tecnología basada en servicios.

Diferencias con otras arquitecturas. Al contrario de las arquitecturas orientado a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación (p.ej., WSDL). La definición de la interfaz encapsula (oculta) las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo (como Plataforma Java o Microsoft.NET). Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio C Sharp podría ser usado por una aplicación Java. En este sentido, ciertos autores definen SOA como una Súper-Abstracción.

Beneficios. Los beneficios que puede obtener una organización que adopte SOA son:

- Mejora en los tiempos de realización de cambios en procesos.
- Facilidad para evolucionar a modelos de negocios basados en tercerización.
- Facilidad para abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores).
- Poder para reemplazar elementos de la capa aplicativa SOA sin interrupción en el proceso de negocio
- Facilidad para la integración de tecnologías disímiles

B.3. Aspectos arquitecturales de los Web Services

Después de describir lo que son los Web Services, ahora entraremos en más detalles sobre los aspectos arquitecturales que con ellos se deben tener en cuenta a la hora de su definición.

B.3.1. Operaciones de los Web Services

B.3.1.1. Descripción del "Servicio"

Visto que el enfoque de los Web Services se centra en la noción de "servicio", una de las primeras cuestiones que se abordarán en su tecnología es lo que exactamente un servicio es, y cómo se puede describir. La descripción del servicio en un middleware convencional se basa en las interfaces y la interface de definición de lenguajes. En ese contexto, las especificaciones IDL (Un Lenguaje de Especificaciones IDL permite especificar interfaces remotas en un lenguaje neutral) son necesarias para generar automáticamente los stubs y construir los bindings dinámicos. La semántica de las distintas operaciones, el orden en que deben ser invocadas, y otras propiedades de los servicios (posiblemente no funcionales), se suponen conocidas de antemano por el programador desarrollador de los clientes. Esto es razonable, ya que los clientes y los servicios son a menudo desarrollados por el mismo equipo. Además, la plataforma middleware, al mismo tiempo que define, limita muchos aspectos de la descripción del servicio y del proceso de binding implícito, y no necesita ser especificada como parte de la descripción del servicio. En Web Services e interacciones B2B, tal contexto

implícito no está, por lo tanto las descripciones de servicios deben ser más ricas y detalladas, abarcando los aspectos más allá de la simple interfaz de servicio.

La pila de la Figura B.9 ilustra los diferentes aspectos involucrados en la descripción de Web Services. Esencialmente la cifra muestra una pila de lenguajes, donde los elementos en los niveles más altos utilizan las descripciones proporcionadas por los elementos de los niveles inferiores.

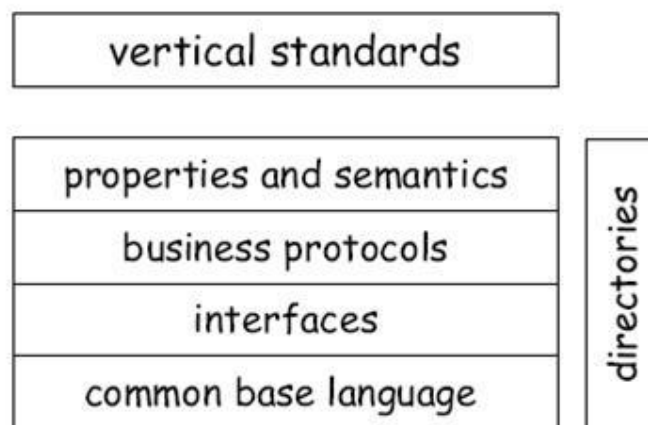


Figura B.9. Descripción del Servicio y Pila de Descubrimiento.

Descripción de cada uno de los niveles:

- ✚ Lenguaje base común. El primer problema que debe abordarse es la definición de un meta-lenguaje común que pueda ser utilizado como base para especificar todos los lenguajes necesarios para describir los diferentes aspectos de un servicio. XML se utiliza para este fin, tanto porque es de una amplia aceptación como porque tiene una sintaxis lo suficientemente flexible como para permitir esta definición de lenguajes de descripción de servicios y protocolos.
- ✚ Interfaces. Los lenguajes de definición de interfaces están en la base de cualquier paradigma orientado a servicios. Puesto que (como se ha mencionado anteriormente) el contexto implícito en Web Services está perdido, su descripción debe ser más completa. En consecuencia, a modo de ejemplo, es necesario especificar la dirección (URI) del servicio y el protocolo de transporte (por ejemplo, HTTP) a utilizar cuando se invoca el servicio. Así, es posible construir un cliente que llame las operaciones que ofrece un Web Service. La propuesta de IDL en este ámbito es el lenguaje de descripción de Web Services (WSDL, Sección 2.1.3).
- ✚ Protocolos de negocios. Un Web Service a menudo ofrece una serie de operaciones que los clientes deben invocar en un cierto orden para lograr sus objetivos. Estos intercambios (ordenados) entre los clientes y los Web Services son llamados conversaciones. Los prestadores de servicios normalmente quieren imponer normas que rigen la conversación, así indicando como las conversaciones son válidas y de común entendimiento. Este conjunto de

normas se especifica como parte del llamado “protocolo de negocios” soportado por el servicio (donde la palabra “negocio” se utiliza para diferenciarlo de un protocolo de comunicación). Los protocolos de negocio son ejemplos de por qué la interfaz de simple descripción no es suficiente en los Web Services. De hecho, para describir un servicio completo, es necesario especificar no sólo su interfaz, sino también los protocolos de negocios que soporta el servicio. En este sentido, existen varias propuestas para estandarizar los lenguajes para la definición de estos protocolos de negocios. Ejemplos de ello son los Lenguajes de Conversación de los Web Services (WSCL) y los Lenguajes de Ejecución de Procesos de Negocios para Web Services (BPEL). Esto es, no obstante, una zona con muy poco desarrollo en términos de estandarización que no atañe a este trabajo.

✚ Propiedades y semánticas. La mayoría de las plataformas de middleware convencionales no incluyen nada sobre interfaces funcionales en la descripción de un servicio. De nuevo, esto se debe a que el contexto del sistema permite a los diseñadores inferir otra información necesaria para unirse o ligarse a un servicio, y porque los servicios están estrechamente unidos, acoplados unos con otros. En consecuencia, los Web Services proporcionan capas de información adicionales para facilitar el binding, donde la descripción de servicio es todo lo que los clientes tienen a su disposición para decidir si utilizar un servicio o no. Por ejemplo, esto puede incluir propiedades no funcionales tales como el costo o la calidad de un servicio, o una descripción textual del servicio, como lo son la política de retorno de mensaje. Esta es la información que es fundamental para utilizar el servicio, pero no es parte de lo que tradicionalmente entendemos como la interfaz del servicio. En los Web Services, dicha información puede estar conectada a la descripción de un servicio mediante el uso del UDDI (Sección 2.1.6). Entonces, esta especificación describe cómo organizar la información acerca de un Web Service y cómo construir los depósitos donde esa información deba ser registrada.

✚ Vertical Standards. Todas las capas explicadas hasta el momento son de carácter genérico. Ellas no estandarizan ni el contenido de los servicios ni las semánticas de los mismos (por ejemplo, el significado de un determinado parámetro o el efecto de una determinada operación). En cambio las Vertical Standards definen interfaces específicas, protocolos, propiedades, y las semánticas que los servicios ofrecen en determinados ámbitos de aplicación que deben soportar. Por ejemplo, la red Rosetta describe los intercambios comerciales en el mundo IT, normalizando todos los aspectos descritos anteriormente. Estos estándares complementan las capas anteriores adaptándolas a aplicaciones concretas, facilitando aún más el uso de herramientas estándares para la correcta conducción de los intercambios. Específicamente, permiten el desarrollo de aplicaciones de cliente que pueden interactuar de manera significativa con cualquier Web Service que les sea compatible.

B.3.1.2. Descubrimiento del “Servicio”

Una vez que los servicios han sido adecuadamente descritos, estas descripciones deben ser puestas a disposición de los clientes interesados en utilizarlas. A tal efecto, las descripciones de servicios se almacenan en un directorio de servicios (representado por la columna vertical en la Figura 5.8). Estas guías permiten a los diseñadores de servicios el registro de nuevos servicios y permitir que los usuarios de los servicios busquen y localicen servicios. El descubrimiento del servicio se puede hacer tanto en tiempo de diseño (navegando por el directorio e identificando la mayoría de los servicios pertinentes) como en tiempo de ejecución (utilizando las técnicas de binding dinámico). Estos directorios pueden ser alojados y gestionados por una entidad de confianza (enfoque centralizado) de cada empresa, o bien cada empresa puede alojar y gestionar un servicio de directorio (enfoque peer-to-peer). En ambos casos, las APIs y protocolos son necesarios para los clientes para que puedan interactuar con el directorio de servicios y así el intercambio de información entre ellos sea posible. La ya mencionada especificación UDDI define APIs estandarizadas para la publicación y descubrimiento de información en el directorio de servicios. También se describe cómo esos directorios deberían funcionar.

B.3.1.3. Interacciones del “Servicio”

Tanto la descripción del servicio como el descubrimiento del mismo están relacionados con binding estático y dinámico. Una vez que el problema del binding ha sido abordado, un conjunto de abstracciones y herramientas que permitan las interacciones entre los Web Services es de importante necesidad. En los Web Services, estas abstracciones toman la forma de una serie de normas que abordan diferentes aspectos de la interacción en diferentes niveles. La Figura B.10 resume estos diferentes aspectos presentándolos como una pila de protocolos, ya que, como veremos, cada aspecto se caracteriza por uno o más protocolos definidos en la parte superior de las capas inferiores inmediatas. A diferencia de las verticals standards discutidas en la sección anterior, estos protocolos son útiles para cualquier Web Service y, por tanto, ejecutado por el middleware de los Web Services. Como tales, son transparentes (en su mayor parte) a los desarrolladores, de la misma forma que las interacciones entre dos objetos CORBA son ocultos a los programadores.

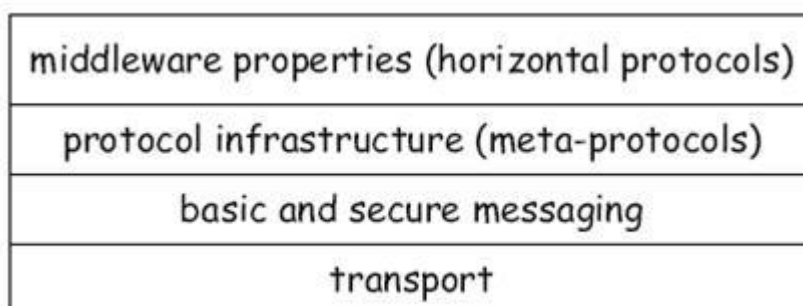


Figura B.10. Pila de interacción de servicios.

Los diferentes tipos de protocolos que componen la pila de las interacciones de servicios son:

- ✚ Transporte. Desde el punto de vista de los Web Services, la red de comunicación está oculta detrás de un protocolo de transporte. Los Web Services pueden usar una amplia gama de protocolos de transporte, aunque ampliamente el más común es HTTP (Sección 2.1.1).
- ✚ Mensajes. Una vez que un protocolo de transporte esté en su lugar, tiene que haber una manera estándar para el formato y el empaquetado de la información a ser intercambiada. En los Web Services, este papel es desempeñado por el Simple Object Access Protocol (SOAP, Sección 2.1.4). SOAP no detalla qué propiedades se asocian con el intercambio (por ejemplo, si se trata de transacción o cifrado). SOAP simplemente especifica un mensaje genérico (en forma de plantilla) para añadir a la parte superior de la aplicación de datos. Existen algunas especificaciones adicionales con el fin de estandarizar la forma de utilizar SOAP para la aplicaciones particulares. Por ejemplo, WS-Security (Sección 2.1.7) describe cómo implementar el intercambio seguro de SOAP.
- ✚ Protocolos de infraestructura (meta-protocolos). Hemos hecho hincapié en que los Web Services se caracterizan no sólo por una interfaz, sino también por los protocolos las empresas deben cumplir. Si bien los protocolos de negocios son aplicaciones específicas, mucho del software necesario para soportar tales protocolos puede ser implementado como componentes de la infraestructura general. Por ejemplo, la infraestructura puede mantener el estado de la comunicación entre un cliente y un servicio, los mensajes asociados a dicha conversación, o verificar que un mensaje de intercambio se produce en conformidad con las normas definidas por los protocolos. Parte de la tarea de la infraestructura, también está en la ejecución de los meta-protocolos. Los meta-protocolos son aquellos que tienen por objeto facilitar y coordinar la ejecución de los protocolos de negocios. Por ejemplo, antes de que pueda darse comienzo a la interacción, los clientes y los servicios necesitan llegar a un acuerdo sobre qué protocolo/s debe/n ser ejecutado/s, quien debe coordinar la ejecución del protocolo, y cómo la ejecución debe llevar identificadores de protocolos en los mensajes para indicar que un cierto intercambio de mensajes se está produciendo. WS-Coordinación (Sección 2.1.8) es una especificación que trata de normalizar los protocolos de estos meta-protocolos y la forma en que WSDL y SOAP deberían utilizarse en estas transmisiones.
- ✚ Propiedades del middleware (protocolos horizontales). Idealmente, el middleware de Web Services debe proporcionar las mismas propiedades que los middleware convencionales (por ejemplo, la fiabilidad), ya que estas son útiles en este contexto. Sabiendo que los Web Services y su infraestructura son por naturaleza distribuidos, las propiedades del middleware que van más allá de comunicaciones básicas, se logran por medio de la estandarización de protocolos peer-to-peer, y se los denomina protocolos horizontales, ya que son generalmente aplicables a muchos Web Services. Entre estas propiedades, por

ejemplo, la fiabilidad y las transacciones requieren la ejecución de protocolos entre la interacción de las entidades participantes (por ejemplo dos PCs). Al igual que los protocolos de negocios, estos protocolos horizontales, puede ser apoyado por los meta-protocolos descritos anteriormente. Sin embargo, es probable que estos protocolos sean ocultos a los desarrolladores de los Web Services y los usuarios, y de esta manera ser totalmente administrados por la infraestructura.

B.3.1.4. La combinación de Web Services: Composición

Un Web Service puede ser implementado a través y por medio de otros Web Services, posiblemente proporcionados por diferentes negocios. Por ejemplo, un distribuidor de computadoras personales puede ofrecer un Web Service que permite a los clientes solicitar cita para obtener los ordenadores. Sin embargo, la aplicación de la operación de respuesta puede requerir la invocación de varios Web Services, incluidos, por ejemplo, los proporcionados por los fabricantes de las PCs o por los transportistas de los mismos. Un Web Service implementado por la invocación de otros Web Services es llamado un servicio compuesto, mientras que un Web Service implementado por el acceso a un sistema local se llama servicio básico. Observe que si un Web Service es compuesto o básico, esto es irrelevante desde el punto de vista de los clientes, ya que es sólo una cuestión de aplicación. De hecho, todos los Web Services se pueden describir, descubrir, e invocar de la misma forma.

Mientras el número de Web Services disponibles siga creciendo y el entorno empresarial siga exigiendo que las nuevas aplicaciones se vayan desarrollando de acuerdo a horarios muy ajustados, tanto la oportunidad como la necesidad de que nuevos servicios compuestos sean desarrollados, como parte de los Web Services de middleware, es extremadamente alta. Estas tecnologías se parecen a las de los sistemas de Workflow, y tienen el potencial de permitir el rápido desarrollo de complejos servicios básicos, en general, para simplificar el mantenimiento y la evolución de los servicios complejos (compuestos). En términos de estandarización, esta área es todavía muy nueva, a pesar de que BPEL parece estar surgiendo como el principal lenguaje de servicios compuestos (Ver Sección 2.1.9).

B.3.2. Los Web Services y la computación Distribuida

B.3.2.1. ¿Qué es el middleware?

El middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible y facilitan la interacción entre aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware.

El middleware es una solución arquitectural al problema de integrar una colección de servidores y aplicaciones bajo una misma interfaz que, por ejemplo, del

lado cliente está implementado por el Sistema Operativo subyacente, el cual posee las librerías que implementan todas las funcionalidades para la comunicación a través de la red.

El middleware trabaja ofreciendo abstracciones que ocultan algunas de las complejidades sobre la construcción de aplicaciones distribuidas. Así, en lugar de que el programador tenga que lidiar con cada aspecto de una aplicación distribuida, es el middleware el que se hará cargo de algunos de ellos. Detrás de estas abstracciones de programación existe una compleja infraestructura de software que las implementa. Hoy en día, la tendencia es que esta infraestructura vaya creciendo en complejidad, a medida que los productos provean más y mejores abstracciones de programación sofisticadas. Todo ello hace que las plataformas middleware sean sistemas de software muy complejos.

Entonces, concluimos en que hay dos grandes aspectos al hablar de middleware, que a veces no son tan claros, pero que deberían serlo. Por un lado, el middleware provee abstracciones de programación para el diseño de aplicaciones distribuidas. Estas abstracciones van desde los modelos de comunicación más sofisticados (tales como llamadas a procedimientos remotos), a transacciones colas para interacciones asíncronas. Por el otro lado, el middleware implementa las funcionalidades que estas abstracciones ofrecen y proveen. Esto puede incluir soporte transaccional, primitivas de comunicación especializadas, nombres y directorios de servicios, persistencia y muchas otras cosas más.

Sin embargo, en la práctica, las plataformas middleware difieren mucho en términos de las abstracciones de programación que proveen de la infraestructura que las implementa.

B.3.2.2. ¿Qué es una plataforma EAI?

Enterprise Application Integration (EAI) o Integración de Aplicaciones de Empresa se define como el uso de software y principios de arquitectura de sistemas para integrar un conjunto de aplicaciones. EAI puede ser usado con diferentes fines:

- Integración de datos (información): asegurando que la información en varios sistemas es consistente. Esto también se conoce como EII (Enterprise Information Integration).
- Integración de procesos: enlace de los procesos de negocios entre diferentes aplicaciones.
- Independencia de proveedor: extrayendo las políticas o reglas del negocio de las aplicaciones e implementándolas en un sistema EAI, de forma que cualquiera de las aplicaciones usadas pueda ser cambiada sin que dichas reglas de negocio deban ser re implementadas.
- Facade común: Un sistema EAI puede actuar como el front-end de un cúmulo de aplicaciones, proporcionando una interfaz de acceso única y consistente a esas aplicaciones y aislando a los usuarios sobre la interacción con distintas aplicaciones.

Los conceptos de middleware y EAI no son completamente distintos, pero sí muy relacionados. En la sección anterior queda en evidencia que cuando dos sistemas son muy distintos en naturaleza y funcionalidad, el hecho de usar middleware convencional para integrarlos se convierte en una tarea extremadamente engorrosa, e incluso en algunos casos simplemente inviable. En consecuencia, EAI puede ser visto o considerado como un paso adelante en la evolución del middleware, extendiendo sus capacidades y características para enfrentar la integración de aplicaciones. Estas extensiones involucran algunos cambios significativos en el modo en que el middleware debe ser utilizado.

Por un lado al middleware le concierne la integración de aplicaciones. En este sentido, el termino plataformas EAI se usa para referirse a específicos sistemas de software que facilitan la integración de aplicaciones heterogéneas. Actualmente, la elección de estas plataformas EAI son mensajes brokers y sistemas de gestión de Workflows.

Ventajas.

1. Acceso a la información en tiempo real entre los sistemas.
2. Permite encadenar los procesos de negocio y ayuda a incrementar la eficiencia organizacional.
3. Mantiene la integridad de la información entre varios sistemas.

Desventajas.

1. Costos de desarrollo muy altos, especialmente para pequeños y medianos negocios (small and mid-sized businesses: SMBs).
2. Las implementaciones de EAI consumen mucho tiempo y requieren muchos recursos.
3. Requieren una gran cantidad de diseño frontal, el cual muchos gerentes son incapaces de visualizar o en el cual no muchos desean invertir. La mayoría de los proyectos de EAI usualmente comienzan como esfuerzos de integración punto a punto, y muy rápidamente se vuelven inmanejables en la medida que el número de aplicaciones aumenta.

B.3.2.3. Web Services y EAI

Servidores de integración como SeeBeyond, Tibco, Vitria, y webMethods ofrecen soluciones que incluyen adaptadores empaquetados para interactuar con sistemas específicos (podríamos considerarlo bibliotecas de 'drivers'), junto con herramientas que simplifican el proceso de construir adaptadores específicos para otros sistemas que tenga la empresa.

El EAI lo que hace es interconectar a todos esos sistemas, de tal forma que cualquier aplicación conectada al EAI (es decir que tenga un adaptador) puede interactuar con cualquier otra aplicación conectada al EAI.

Podríamos ver al EAI como un traductor que permite que dos sistemas que hablan en idiomas distintos sean capaces de entenderse.

Conforme ha ido pasando el tiempo, los constructores de sistemas EAI han ido añadiendo funciones avanzadas como: el mapeo de datos, la transformación y traducción de datos, la coordinación de transacciones, la gestión de comunicaciones y la gestión de procesos de negocio. Estas capacidades son críticas para las empresas de cierto tamaño en sus proyectos de integración de sistemas.

Estos productos tienen también su parte menos brillante como el coste, la complejidad, y el uso de arquitecturas propietarias.

Desde la perspectiva de la integración de aplicaciones, los Web Services (de los cuales ya hemos dicho que este es uno de sus escenarios típicos) ofrecen ventajas sobre los sistemas EAI en términos de uso de estándares, simplicidad, y bajo coste, ya que ofrecen una solución rápida y ajustada en coste para resolver problemas de interoperatividad e integración, usando infraestructuras ya existentes y reutilizando la tecnología de componentes.

Eso sí, tenemos que también ser conscientes que actualmente los Web Services no son capaces de ofrecer todo el conjunto de características avanzadas de muchos sistemas EAI, como por ejemplo, gestión de la capa de comunicación, coordinación de transacciones, seguridad, etc., aunque son cuestiones sobre las que se está trabajando.

Los Web Services por tanto son atractivos para soluciones de este tipo en el ámbito de la integración de baja y media complejidad.

Hoy en día, los Web Services y sus tecnologías se están desarrollando con un uso específico en mente: la de ser los puntos de entrada al sistema de información local. Por lo tanto, el uso primario de un Web Service es la de exponer (a través de la interfaz de Web Services) la funcionalidad de los sistemas internos y hacerla accesible y descubierta a través de la Web de una manera controlada. La idea de los Web Services es, por tanto, análoga a sofisticadas envolturas que encapsulan una o más aplicaciones al proporcionar una única interfaz Web y su acceso (Figura 5.6). Por supuesto que esto es una simplificación, y la realidad es un poco más compleja, pero esto ayuda a aclarar la interpretación sobre como los Web Services se utilizan.

Los componentes de envolver y ocultar la heterogeneidad es la clave de la aplicación que permite la integración. Desde la perspectiva de los clientes, los envoltorios son los componentes que deben integrarse, ya que son lo que la aplicación de integración puede ver del sistema subyacente.

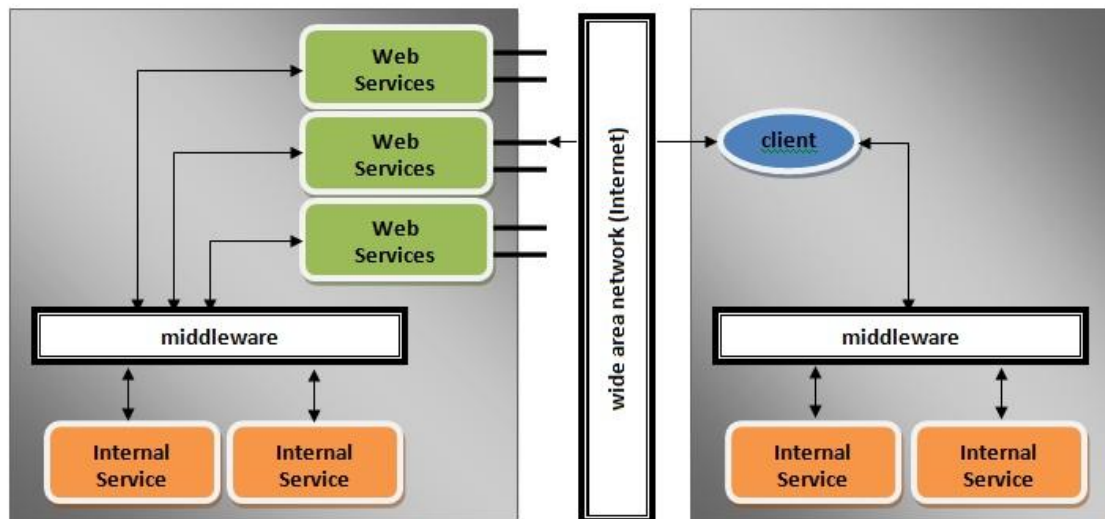


Figura B.11. Los Web Services proveen un punto de entrada para el acceso a servicios locales.

Teniendo componentes heterogéneos se reducen considerablemente las dificultades de integración. Esto también es válido para los Web Services, que son, en efecto, wrappers homogéneos, en el sentido de que interactúan a través de estándares web. Como tales, constituyen la base sobre la que podemos construir el soporte de middleware de integración de aplicaciones Web, permitiendo a los diseñadores evitar los problemas generados por la falta de estandarizaciones típicas de los enfoques anteriores.

Este es un aspecto interesante de los Web Services, a pesar del hecho de que la mayoría de la literatura sobre el tema está fuertemente apuntada hacia las aplicaciones B2B y que la integración de B2B es lo que genera en esencia la necesidad por los Web Services. De hecho, los Web Services no tienen que ser si o si accesibles a través de Internet. Es perfectamente posible desarrollar Web Services disponibles a los clientes que residen en una LAN local (Figura 3.7).

Muchos Web Services se utilizan en el contexto presentado, es decir, dentro de cierta empresa en sentido de la integración de aplicaciones en lugar de intercambios entre empresas. Esta tendencia se consolidará a medida de que los proveedores de software mejoren y amplíen su apoyo a los Web Services. De hecho, si las aplicaciones de software salieran de su caja con una interfaz de Web Services, su integración sería simplificada considerablemente, ya que todos los componentes en este contexto son homogéneos. Sin embargo, el reto y objetivo final de los Web Services es la interacción entre empresas. Los esfuerzos existentes en torno a los Web Services van en esta dirección, aunque esto es claramente un objetivo a largo plazo.

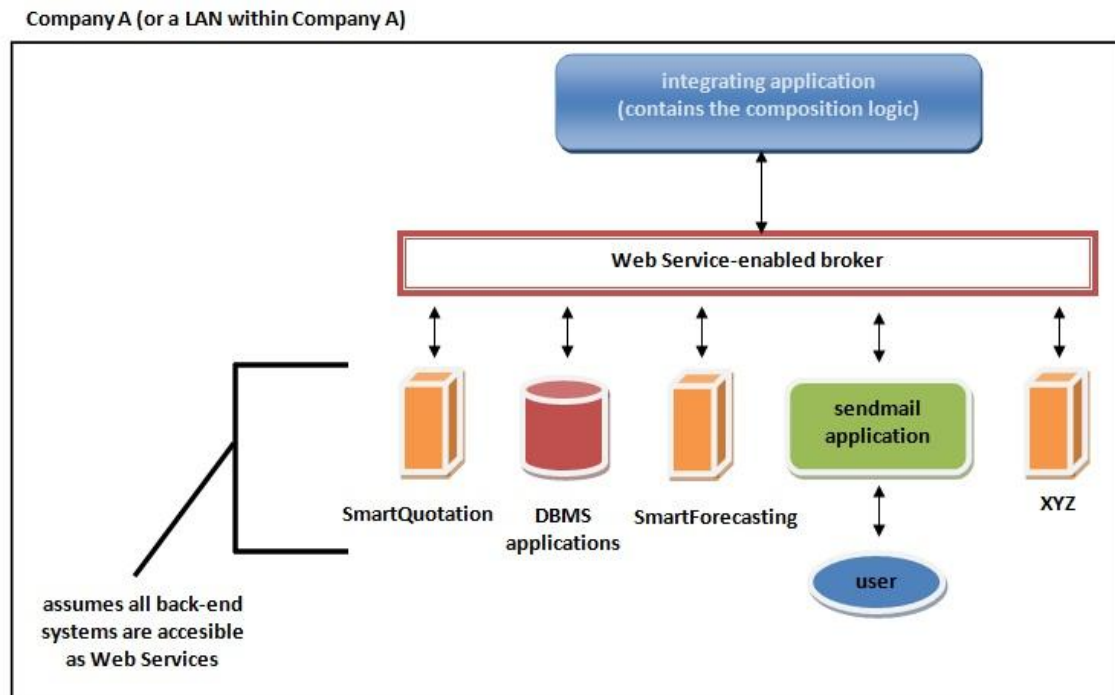


Figura B.12. Los Web Services también pueden ser utilizados sin estar en función de aplicaciones EAI.

B.3.3. Arquitectura global de los Web Services.

Al haber finalizado la descripción de los principales componentes middleware de los Web Services en las secciones anteriores, esto nos permite mostrar ahora cómo se pueden combinar dentro de una arquitectura global.

B.3.3.1. Las dos facetas de arquitecturas de los Web Services

Las secciones anteriores han mostrado que hay dos grandes aspectos que deben considerarse al analizar las arquitecturas de los Web Services, a saber.

El primer aspecto está relacionado con el hecho de que los Web Services son una manera de exponer operaciones internas a fin de que puedan ser invocadas a través de la Web. Una aplicación (y por consecuencia, una implementación) con esa característica requiere que el sistema sea capaz de recibir solicitudes a través de la Web para luego pasarlas al sistema informático subyacente. De este modo, los problemas bajo este aspecto son análogos a los encontrados en los middleware convencionales. En adelante, nos referiremos a este tipo de infraestructura como middleware interno para Web Services. De la misma manera, vamos a utilizar el término arquitectura interna para referirnos a la organización y estructura del recientemente mencionado middleware interno (Figura 5.1).

La otra faceta de las arquitecturas de los Web Services está representada por la infraestructura de middleware cuyo objetivo es integrar diversos Web Services. Esta vez, nos referiremos a este tipo de infraestructura como infraestructura de middleware externo para Web Services (Figura 5.1). De la misma manera, vamos a utilizar la expresión arquitectura externa de arquitectura para referirnos a la

organización y estructura del middleware externo. La arquitectura externa tiene tres componentes principales:

- 1) Brokers centralizados (agentes centrales). Estos agentes son análogos a los componentes centralizados del middleware convencional, quienes se encargan del ruteo de mensajes y de proporcionar las propiedades necesarias en las interacciones (como el loggeo, garantías de transacciones, el nombre y directorio del servicio, y la fiabilidad). Sin embargo, como veremos, en la práctica, el nombre y el directorio del servicio son a menudo los únicos componentes centralizados presentes en las arquitecturas de Web Services.
- 2) Protocolo de infraestructura. El protocolo de infraestructura se refiere al conjunto de componentes que coordinan las interacciones entre los Web Services y, en particular, que implementan los protocolos de tipo peer-to-peer (como los protocolos horizontales y los meta-protocolos discutidos en la sección anterior), cuyo objetivo es proporcionar propiedades del middleware a las configuraciones B2B.
- 3) Composición de infraestructura de servicios. Esto se refiere al conjunto de herramientas que apoyen la definición y ejecución de servicios compuestos.

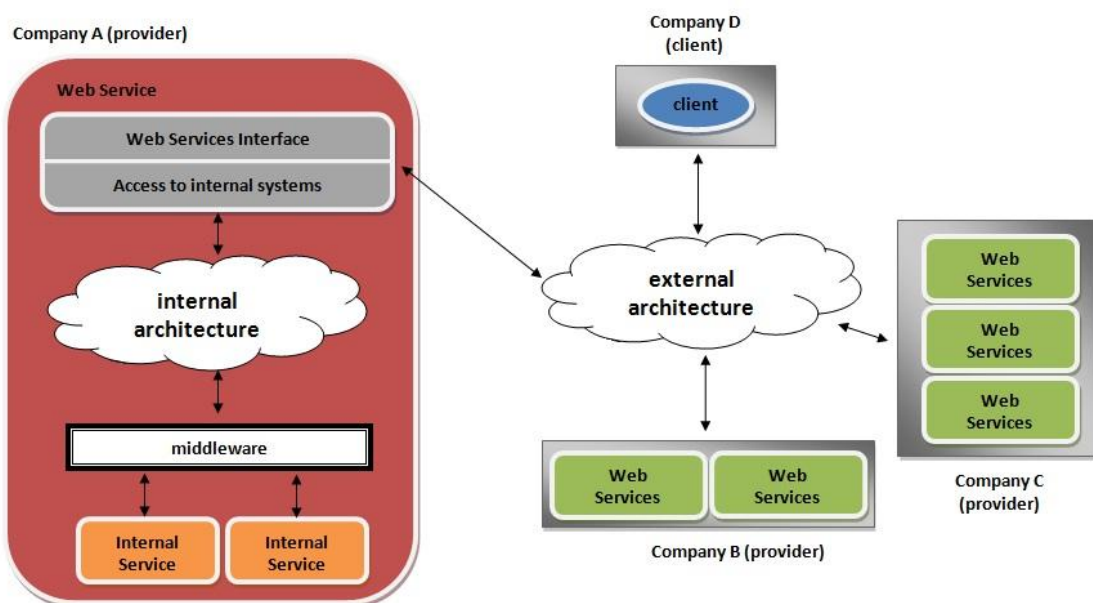


Figura B.13. Los Web Services requieren tanto de una Arquitectura interna como una externa.

Observemos lo siguiente: que un componente forme parte de una arquitectura interna o de una arquitectura externa es independiente, en gran medida, de si el componente es utilizado por el prestador de servicios o por un tercero. De hecho, la misma distinción entre arquitectura interna y externa puede hacerse cuando se utilizan Web Services para EAI, es decir, Web Services dentro de la misma empresa.

Esta distinción entre arquitectura interna y externa es fundamental para comprender bastante de lo que está ocurriendo en torno a los Web Services. Existen tecnologías y productos de Web Services que se ocupan únicamente ellos mismos de la arquitectura interna de un Web Service. Por otro lado, existen también tecnologías y productos que sólo abordan la arquitectura externa de un Web Service. Sin embargo, los esfuerzos de estandarizaciones en este sentido giran principalmente en torno a la arquitectura externa. En la práctica, tanto la arquitectura interna como su par externa, deben trabajar juntas para que un Web Service pueda hacer que su funcionalidad sea accesible a, y para, todos los clientes.

B.3.3.2. Arquitectura Interna de un Web Service

La forma más fácil de entender la arquitectura interna de los Web Services es verlos como solo un nivel en la parte superior de los demás niveles en la arquitectura de una empresa.

En general, el middleware convencional se utiliza para crear arquitecturas de varios niveles. En estas arquitecturas, los programas o aplicaciones individuales se ocultan detrás de las abstracciones del servicio, las cuales, a su vez, son combinadas con el objetivo de formar más programas o aplicaciones pero de un orden más alto que las anteriores. Esto se puede llevar a cabo gracias a la disponibilidad de las funcionalidades proporcionadas por los middleware subyacentes. Estos programas resultantes de orden más alto pueden ser ocultos detrás de las abstracciones de los nuevos servicios pudiendo ser utilizados como bloques de construcción para los nuevos servicios. Dado que la composición de las abstracciones de los servicios pueden repetirse libremente, el resultado final es un sistema de varios niveles en los que los servicios que se ejecutan, son implementados por encima de otros servicios y programas básicos. La correspondiente arquitectura se muestra en la Figura 5.2.

Cuando instancias múltiples de middleware son apiladas una encima de otra, el middleware utilizado en cada nivel no tiene por qué ser el mismo. Lo importante es contar con abstracciones de servicio compatibles o hacer los servicios compatibles usando wrappers. El middleware simplemente actúa como el nexo necesario para que todos los componentes de un determinado nivel puedan interactuar unos con otros para formar los servicios finales, y que estos puedan ser utilizados por los clientes o por niveles más altos jerárquicamente hablando. Aunque no es estrictamente necesario, por lo general los componentes básicos de cada instancia de los middlewares, residen en una LAN, lugar donde también se ejecutará la aplicación resultante.

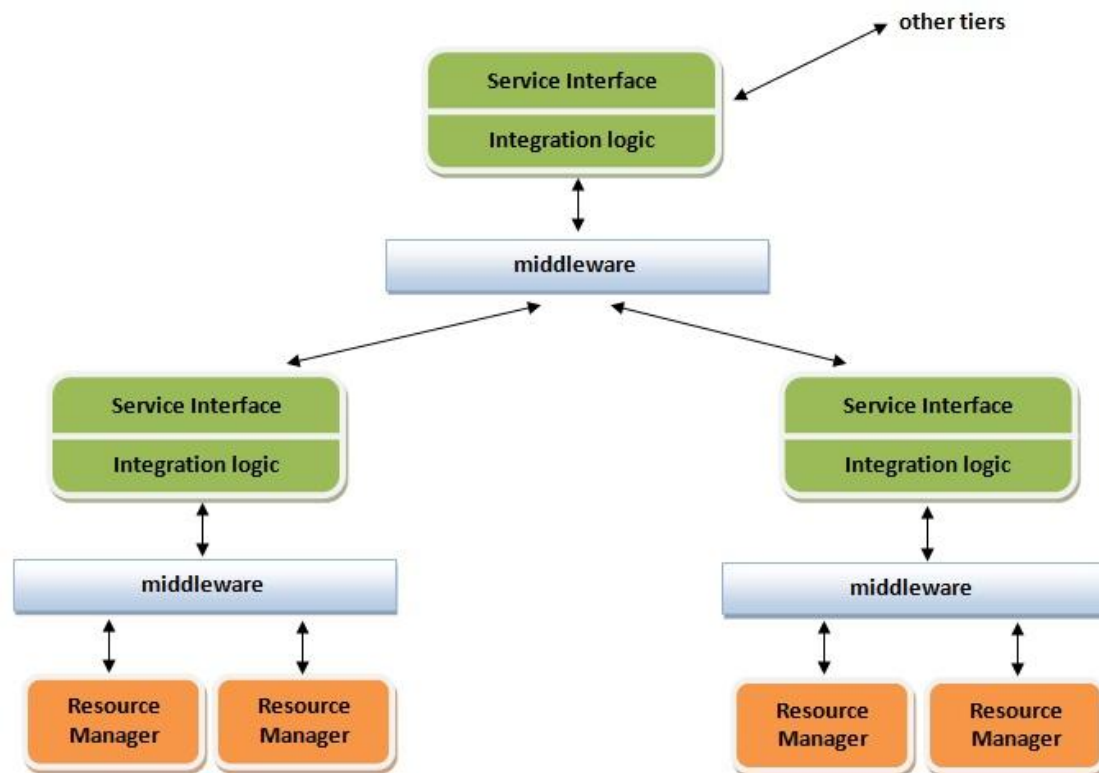


Figura B.14. Middleware convencional como una plataforma integradora de aplicaciones y programas básicos.

Los Web Services, o más precisamente, las tecnologías de los Web Services, juegan el mismo papel que los middleware convencionales, pero a una escala diferente. Las bases de la composición de servicios (abstracciones de servicios) son de naturaleza muy similar a las utilizadas en los middleware convencionales, de modo que la implementación de aplicaciones de un Web Service, requiere esencialmente un nivel (extra, si se quiere) por encima de los demás niveles para así permitir el acceso al servicio a través de los protocolos de estandarización. La Figura 5.3 muestra un ejemplo típico de este tipo de arquitectura interna. Tenga en cuenta que la figura enfatiza el hecho de que la aplicación no se produce en la capa de Web Services sino dentro de los middleware convencionales.

Como se observó anteriormente, los Web Services son sólo wrappers (envoltorios). Ellos solo invocan servicios internos que implementan cual sea la aplicación lógica que se necesite y, a continuación, recogen los resultados.

Hoy en día, gran parte del middleware interno para Web Services gira en torno al embalaje y desembalaje (empaquete y desempaquete) de mensajes intercambiados entre diversos Web Services, y la conversión de estos mensajes al formato soportado por el middleware subyacente. Esto es similar a cómo un servidor de aplicaciones mapea datos en páginas HTML y viceversa. Obsérvese, que la presencia de este nivel adicional y la necesidad de convertir mensajes causa lógicamente una sobrecarga en la tramitación de dichos mensajes. Esta es también la razón por la cual también, los Web Services tienden a ser utilizados para la transmisión de operaciones, digamos de grueso calibre, grandes, en donde las sobrecargas causadas por las conversiones son pequeñas en comparación con los tiempos de ejecución que ellas conllevan.

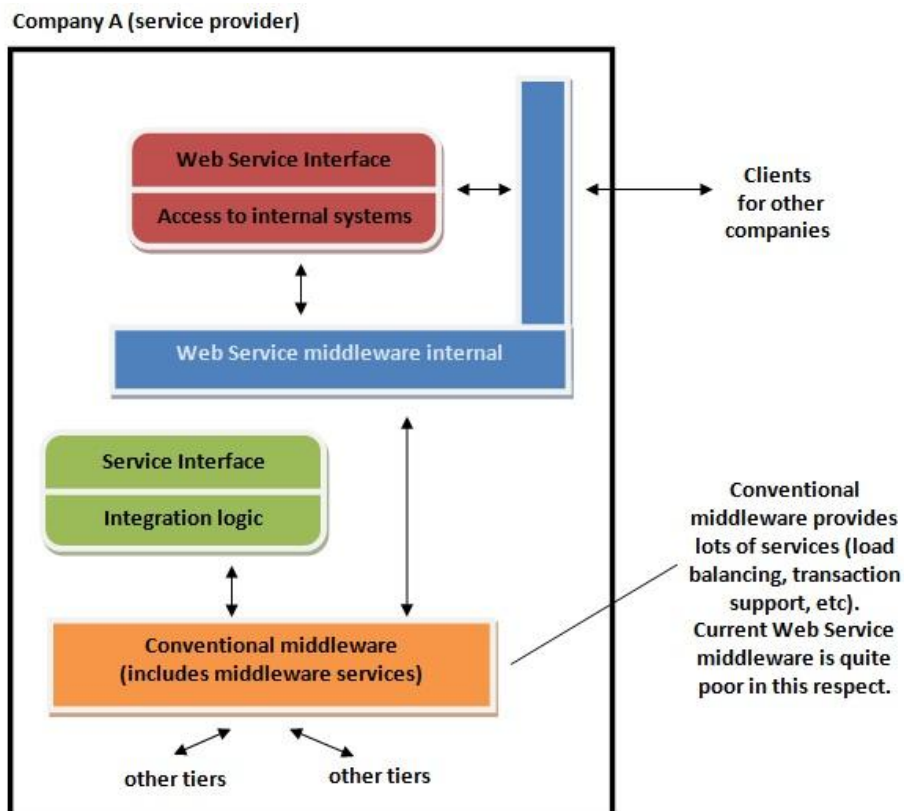


Figura B.15. Arquitectura básica de un conjunto de Web Services.

B.3.3.3. Arquitectura Externa de un Web Service

El uso de plataformas de middleware convencional para implementar las arriba mencionadas arquitecturas internas de Web Services es algo natural. Sin embargo, lo que hemos visto hasta ahora se relaciona con las funcionalidades internas de un Web Service y en cómo se las ve de forma de wrappers, y no a la integración de estos wrappers. Este aspecto, que fue abordado por mensajes brokers y/o sistemas de gestión de Workflow (WfMSs) en el middleware convencional, debe ser el trabajo de los middleware externos.

Ahora bien, como hemos observado anteriormente, no es claro dónde debe residir este middleware al que nos referimos. De todas formas, en los Web Services, las partes pueden residir en distintos lugares, y no hay lugar evidente para localizar el middleware.

Hay dos soluciones a este problema. Una es implementar el middleware como un sistema "peer-to-peer" en el que todos los participantes colaborarán para proporcionar servicios de directorio y nombre. Conceptualmente, se trata de un enfoque muy atractivo, pero no es evidente la manera o forma de proveer el grado de fiabilidad y confianza necesaria en los sistemas industriales grandes y fuertes. La otra solución es introducir intermediarios o brokers que actúen como el middleware necesario. Suponiendo que encontremos algún sitio en algún lugar de la red en el que podamos confiar y del que sepamos es suficientemente confiable, el sitio podría actuar como el nombre y servidor de directorio de Web Services que necesitamos. Ahora

bien, si vemos este tipo de servidores como parte de la infraestructura middleware de los Web Services, se deduce que los participantes y parte del middleware pueden residir en distintos lugares.

Actualmente, sólo hay un tipo de bróker de Web Services que ha sido estandarizado y que se utiliza en la práctica, aunque en un grado muy limitado: solo incluye el nombre y el servidor de directorio. Como resultado, gran parte de la literatura existente sobre la arquitectura de Web Services gira en torno a ese bróker.

La Figura 5.4 muestra la arquitectura externa de Web Services tal y como es entendida hoy en día respecto a los componentes centralizados. Esta representación hace hincapié en que las abstracciones y las infraestructuras del descubrimiento de los Web Services son parte del middleware externo.

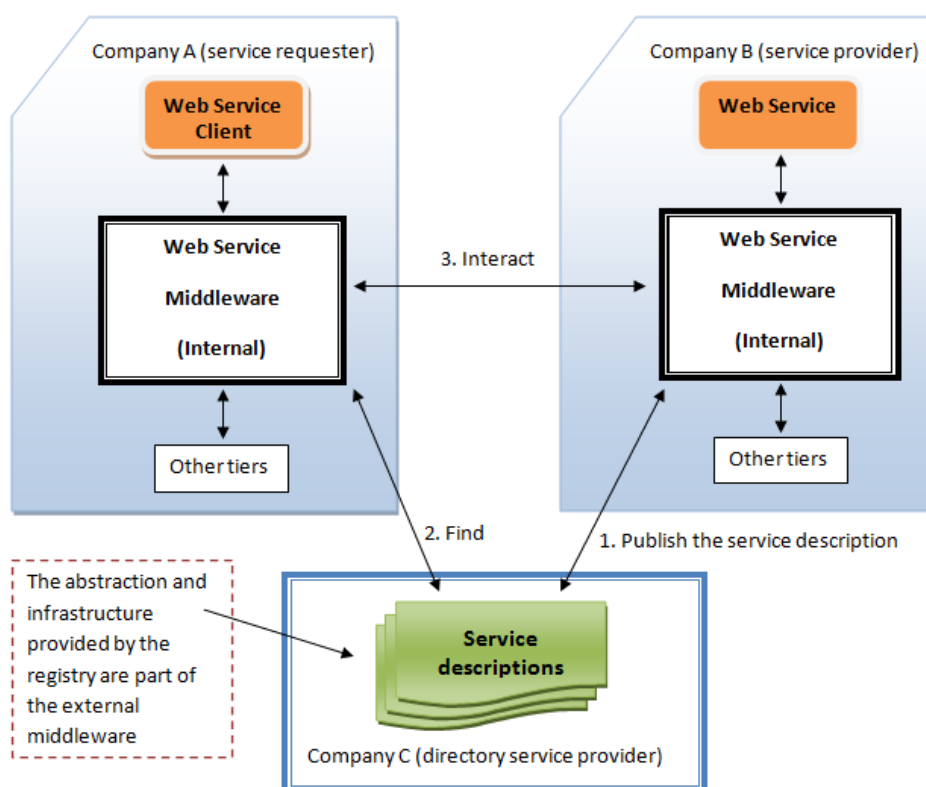


Figura B.16 Arquitectura Externa de un Servicio WB

La figura describe cómo tiene lugar el descubrimiento del servicio en un entorno de Web Services. Haciendo caso omiso o ignorando los detalles de los protocolos y la sintaxis subyacente de los intercambios correspondientes. En otras palabras, el procedimiento que se muestra es un mecanismo normal de descubrimiento de servicios. Más aun, este mecanismo podría aplicarse no sólo a los Web Services, sino también para casi cualquier tipo de middleware, incluyendo las primeras versiones de RPC. La idea es para que los proveedores de servicios creen Web Services y puedan definir una interfaz para la invocación de los mismos. Además, el proveedor de servicios también tiene que generar las descripciones de servicios para esos servicios. Finalmente, el proveedor de servicios deberá hacer conocer sus servicios al mundo mediante la publicación de las correspondientes descripciones de servicios en un registro de servicios. La información que se incluye con la descripción

del servicio es de vital importancia, ya que es utilizada por el registro de servicios no solo para catalogar cada servicio, sino también para buscarlo y encontrarlo cuando las solicitudes de servicio de los solicitantes (nuevos clientes) llegan. Cuando una nueva solicitud trata de encontrar un determinado servicio, consulta el registro de servicios. Este registro envía respuestas con las descripciones de servicios correspondientes en donde indica como localizar el servicio y la forma en que se lo debe invocar. Así, el solicitante luego podrá comunicarse con el proveedor del servicio mediante la invocación del servicio que este último provee. El directorio en sí mismo es muy probablemente un Web Service, cuyas direcciones e interfaz se suponen conocidas a priori por el solicitante en cuestión.

En función de todo esto, lo esencialmente importante en la Figura 5.4 no es tanto el mecanismo que describe, sino el hecho de que representa el descubrimiento del servicio como un único componente de Web Services.

Consideremos de nuevo el ejemplo de gestión de transacciones. Podríamos utilizar exactamente el mismo enfoque que middleware convencional y así tener un broker centralizado de intermediario para los Web Services, de manera que el nombre y el servidor de directorio funcionen. De esta forma, la Figura 5.4 se ampliaría mediante la adición de una nueva parte correspondiente al hipotético broker, cuya implementación se asemeja a la de los agentes en el middleware convencional. Tal solución es técnicamente viable, pero abre muchas cuestiones que son muy difíciles de resolver en la práctica. Por un lado, se requiere una forma estándar en el funcionamiento de las transacciones aceptadas por todos a fin de que la semántica de las transacciones no sea violada. Ahora bien, como estas semánticas en cada punto final de las transacciones están dadas por las plataformas de middleware, esto equivale a estandarizar las interacciones de las transacciones por medio de las herramientas de middleware. Hay esfuerzos en curso para hacer precisamente eso, pero acaban de comenzar, y llevaría bastante tiempo antes de que las plataformas de middleware sigan una interfaz de transacciones común.

Una restricción mucho más importante es que este enfoque asume que todos los participantes confíen en el broker. Salvo en configuraciones muy restringidas, esto es altamente improbable. Existe una gran diferencia entre consultar un registro de servicio proporcionado por una empresa externa y brindar un completo seguimiento de cada transacción de un Web Service de una empresa. Las empresas normalmente guardan esta información haciendo irrazonable suponer que se basarán en algunos sistemas externos para realizar seguimientos de sus transacciones. Incluso la propia funcionalidad básica del registro de servicios que está disponible en una arquitectura centralizada no es muy utilizada hoy en día, por consecuencia de la desconfianza. De todas formas, esto puede cambiar en el futuro si se confiara en los brokers que aparecen, ellos podrían jugar un papel similar al desempeñado por las empresas como Yahoo! o Lycos para compras on-line.

Una solución alternativa es aplicar nuestra hipotética transacción como un bróker en un sistema peer-to-peer. La idea aquí es que cada solicitante de servicio tendrá su propio gerente de operación. Cuando el solicitante invoque Web Services en forma normal de transacción, su propio gestor de la transacción se hace responsable de orquestar la ejecución de transacciones a fin de que las semánticas transaccionales se conserven, o mejor dicho, se preserven. Al igual que en la solución centralizada, esto también requiere de una interacción transaccional estandarizada. Sin embargo, la

funcionalidad proporcionada por esta solución es quizás un subconjunto de la solución proporcionada por los sistemas de middleware convencionales.

Argumentos similares se pueden utilizar para otros componentes que normalmente se centralizan en cualquier arquitectura externa como la vista. Este es el motivo por el que las propiedades de los middleware son, en general, provistas a través de protocolos peer-to-peer y a través de una infraestructura que apoya la ejecución de dichos protocolos. Esta infraestructura es parte de la arquitectura externa, pero normalmente es de propiedad y control del solicitante y del proveedor del servicio, no por un tercero (Figura 5.5).

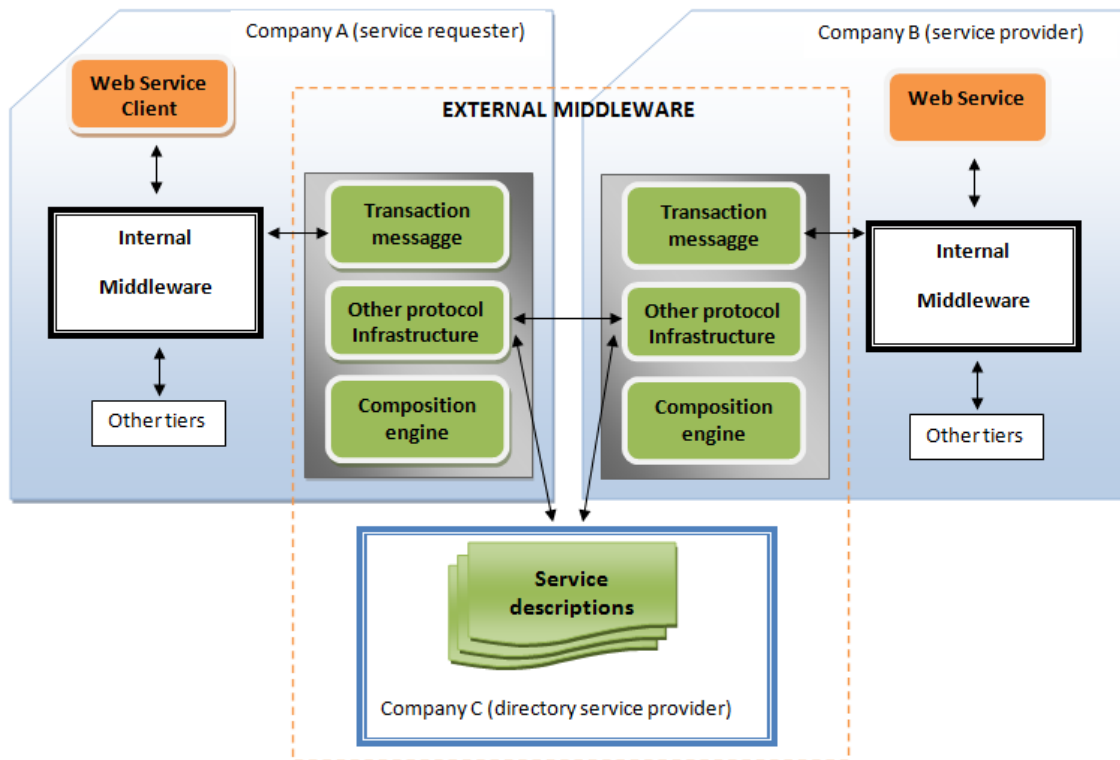


Figura B.17. Arquitectura Externa de un Web Service basada en un protocolo peer-to-peer y en composición de servicios.

Las herramientas de composición de servicios son otro ingrediente de las arquitecturas de Web Services externos, que pueden ganar más importancia a medida que las tecnologías sigan creciendo. Es importante considerar la composición como parte de una arquitectura externa, ya que se trata de la integración de diversos Web Services. Técnicamente, la infraestructura de la composición de servicios puede ser centralizada. Sin embargo, desde el punto de vista de la puesta en práctica, esta infraestructura es a menudo de propiedad registrada y confidencial, lo que hace que probablemente sea finalmente desarrollada por un proveedor de servicios, y no por un tercero (Figura 5.5).

ANEXO C: WSDL y UDDI

Este anexo se especializa en las dos tecnologías de los Web Services que se han tomado como conceptos básicos para desarrollar la herramienta especificada en el capítulo 5.

La composición del anexo es la siguiente: En la sección 1 se desarrollan los conceptos referentes a WSDL (Web Services Description Language) partiendo de XML, lenguaje de marcas padre de WSDL. Y en la sección 2 se complementa la información provista en el capítulo 4 respecto del protocolo UDDI (Universal Description, Discovery and Integration).

C.1. Web Service Description Language

C.1.1. XML: el lenguaje de los Web Services

Si bien WSDL (Web Service Description Language) es en realidad el lenguaje de especificación utilizado para definir los Web Services, esencialmente se trata de un documento XML que es utilizado para describir los mensajes SOAP y cómo estos mensajes son intercambiados. Entonces, en este apartado el lector conocerá por así decirlo, el lenguaje sobre el cual se soportan los Web Services, XML. Se evocará en sus orígenes, sus características principales y porque es el lenguaje escogido para desarrollar Web Services.

C.1.1.1. Vista general de XML

A continuación se presenta un ejemplo concreto para comprender este poderoso lenguaje:

El lenguaje HTML permite insertar menús, tablas, imágenes o bases de datos en los documentos, pero no permite al usuario que maneje esos elementos como mejor le convenga con la poderosa ayuda del ordenador. Esa es la principal novedad que XML aporta.

Con HTML se pueden hacer accesos a información comparativa en diferentes negocios por ejemplo, pero nada más. Con XML el usuario podrá ordenar los datos o actualizarlos en tiempo real o realizar un pedido.

La información que manejan las empresas es uno de sus principales activos. Pero lo normal es que esa información esté fragmentada, en diferentes departamentos, ordenadores conectados o no, etc. El reto ahora está en interrelacionar toda esa información para rendir todo su potencial y ponerlo a trabajar para aumentar los beneficios o reducir los costes. Para realizar esto se necesita un estándar de almacenamiento estructurado que es lo que nos ofrece XML. Así, se puede definir a XML como un lenguaje de Marcas que soporte precisamente ese almacenamiento estructurado, pero entonces ¿qué es exactamente un lenguaje de marcas?

C.1.1.2. Lenguajes de Marcas

En los años 60, IBM intentó resolver sus problemas asociados al tratamiento de documentos en diferentes plataformas a través de GML (Generalized markup Language). El principal problema era que cada aplicación utilizaba sus propias marcas para describir los diferentes elementos. Las marcas son códigos que indican a un programa cómo debe tratar su contenido y así, si se desea que un texto aparezca con un formato determinado, dicho texto debe ir delimitado por la correspondiente marca que indique como debe ser mostrado en pantalla o impreso. Y lo mismo ocurre con todas las demás características de cualquier texto.

Entonces, conociendo este sistema y también conociendo a la perfección el sistema de marcas de cada aplicación sería posible pasar información de un sistema a otro sin necesidad de perder el formato indicado. La forma que IBM creó para solventar esto se basaba en tratar las marcas como texto accesible desde cualquier

sistema, texto plano, código ASCII. Y la norma se denominó GML (General Modeling Language).

Más tarde GML pasó a manos de ISO y se convirtió en SGML (ISO 8879), Standart Generalized Markup Language. Esta norma es la que se aplica desde entonces a todos los lenguajes de marcas, cuyos ejemplos más conocidos son el HTML y el RTF.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación aunque se definan igualmente como "lenguajes". Son sistemas complejos de descripción de información, normalmente documentos, que si se ajustan a SGML, se pueden controlar desde cualquier editor ASCII. Las marcas más utilizadas suelen describirse por textos descriptivos encerrados entre signos de "menor" (<) y "mayor" (>), siendo lo más usual que existan una marca de principio y otra de final.








Se puede decir que existen tres utilidades básicas de los lenguajes de marcas: los que sirven principalmente para describir su contenido, los que sirven más que nada para definir su formato y los que realizan las dos funciones indistintamente. Las aplicaciones de bases de datos son buenas referencias del primer sistema, los programas de tratamiento de textos son ejemplos típicos del segundo tipo, y aunque no lo parezca, el HTML es la muestra más conocida del tercer modelo.

C.1.1.3. Significado e Historia de XML

XML es el estándar de Extensible Markup Language. XML no es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

XML fue creado al amparo del Word Wide Web Consortium (W3C), organismo que vela por el desarrollo de WWW partiendo de las amplias especificaciones de SGML. Su desarrollo se comenzó en 1996 y la primera versión salió a la luz el 10 de febrero de 1998. La primera definición que apareció fue: Sistema para definir validar y compartir formatos de documentos en la web. Durante el año 1998 XML tuvo un crecimiento exponencial, y con ello me refiero a sus apariciones en medios de comunicación, menciones en páginas web, soporte software, etc.

Respecto a sus objetivos, estos son:

-  XML debe ser directamente utilizable sobre Internet.
-  XML debe soportar una amplia variedad de aplicaciones.
-  XML debe ser compatible con SGML.
-  Debe ser fácil la escritura de programas que procesen documentos XML.
-  El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
-  Los documentos XML deben ser legibles por humanos y razonablemente claros.
-  El diseño de XML debe ser preparado rápidamente.

- ✚ El diseño de XML debe ser formal y conciso.
- ✚ Los documentos XML deben ser fácilmente creables.
- ✚ La concisión en las marcas XML es de mínima importancia.

Por otra parte, se pueden enumerar sus principales características:

- 1) Es una arquitectura abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/intranet por medio de un validador de documentos (parser).
- 2) Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta del HTML.
- 3) Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.
- 4) Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.
- 5) Gestión y manipulación de los datos desde el propio cliente web.
- 6) Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible.
- 7) Se desarrollarán de manera extensible las búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conllevará que los clientes web puedan ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.
- 8) Se permitirá un comportamiento más estable y actualizable de las aplicaciones web, incluyendo enlaces bidireccionales y almacenados de forma externa (El famoso epígrafe "404 file not found" desaparecerá).
- 9) El concepto de "hipertexto" se desarrollará ampliamente (permitirá denominación independiente de la ubicación, enlaces bidireccionales, enlaces que pueden especificarse y gestionarse desde fuera del documento, hiperenlaces múltiples, enlaces agrupados, atributos para los enlaces, etc. Creado a través del Lenguaje de enlaces extensible (XLL).
- 10) Exportabilidad a otros formatos de publicación (papel, web, CD-ROM, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

C.1.1.4. Estructura del XML

El metalenguaje XML consta de cuatro especificaciones (el propio XML sienta las bases sintácticas y el alcance de su implementación) que se enumeran en la tabla C.1:

1	<p>DTD (Document Type Definition) Definición del tipo de documento. Es, en general, un archivo/s que encierra una definición formal de un tipo de documento y, a la vez, especifica la estructura lógica de cada documento. Define tanto los elementos de una página como sus atributos. El DTD del XML es opcional. En tareas sencillas no es necesario construir una DTD, entonces se trataría de un documento “bien formado”(wellformed) y si lleva DTD será un documento “validado” (valid).</p>
2	<p>XSL (eXtensible Stylesheet Language) Define o implementa el lenguaje de estilo de los documentos escritos para XML. Desde el verano de 1997 varias empresas informáticas como Arbortext, Microsoft e Inso vienen trabajando en una propuesta de XSL (antes llamado “xml-style”) que presentaron a W3C. Permite modificar el aspecto de un documento. Se puede lograr múltiple columnas, texto girado, orden de visualización de los datos de una tabla, múltiples tipos de letra con amplia variedad en los tamaños. Este estándar está basado en el lenguaje de semántica y especificación de estilo de documento (DSSSL, Document Style Semantics and Specification Language, ISO/IEC 10179) y, por otro lado, se considera más potente que las hojas de estilo en cascada (CSS, Cascading Style Sheets), usado en un principio con el lenguaje DHTML. “Se espera que el CSS sea usado para visualizar simples estructuras de documentos XML (actualmente se ha conseguido mayor integración en XML con el protocolo CSS2 (Cascading Style Sheets, level 2) ofreciendo nuevas formas de composición y una más rápida visualización) y, por otra parte, XSL pueda ser utilizado donde se requiera más potencia de diseño como documentos XML que encierran datos estructurados (tablas, organigramas, etc.)(2)”.</p>
3	<p>XLL (eXtensible Linking Language) Define el modo de enlace entre diferentes enlaces. Se considera que es un subconjunto de HyTime (Hipermedia/Timed-based structuring Language o Lenguaje de estructuración hipermedia/basado en el tiempo, ISO 10744) y sigue algunas especificaciones del TEI (Text Encoding Initiative o Iniciativa de codificación de texto). Desde marzo de 1998 el W3C trabajo en los enlaces y direccionamientos del XML. Provisionalmente se le renombró como Xlink y a partir de junio se le denomina XLL.</p> <p>Este lenguaje de enlaces extensible tiene dos importantes componentes: Xlink y el Xpointer. Va más allá de los enlaces simples que sólo soporta el HTML. Se podrá implementar con enlaces extendidos. Jon Bosak establece los siguientes mecanismos hipertextuales que soportará esta especificación:</p> <ul style="list-style-type: none">○ Denominación independiente de la ubicación.○ Enlaces que pueden especificarse y gestionarse desde fuera del

	<p>documento a los que se apliquen (Esto permitirá crear en un entorno intranet/extranet un banco de datos de enlaces en los que se puede gestionar y actualizar automáticamente. No habrá más errores del tipo “404 Not Found”).</p> <ul style="list-style-type: none"> ○ Hiperenlaces múltiples (anillos, múltiples ventanas, etc.). ○ Enlaces agrupados (múltiples orígenes). ○ Transclusión (el documento destino al que apunta el enlace aparece como parte integrante del documento rigen del enlace). ○ Se pueden aplicar atributos a los enlaces (tipos de enlaces).
4	<p>XUA (XML User Agent): Estandarización de navegadores XML. Todavía está en proceso de creación de borradores de trabajo. Se aplicará a los navegadores para que compartan todas las especificaciones XML.</p>

Tabla C.1. Especificaciones de XML.

C.1.2. XML y los Web Services

Finalmente ahora que ya se conoce algo más sobre XML, queda responderse ¿porqué XML es utilizado en los Web Services?:

- ✚ Es un estándar abierto es decir que es reconocido mundialmente ya que muchas compañías tecnológicas integran en sus software compatibilidad con dicho lenguaje. Esto quiere decir que la gran mayoría de software de escritorio de sistema operativo, aplicaciones móviles permiten la compatibilidad con XML esto lo hace muy potente a la hora de permite la comunicación entre distintas plataformas de software y hardware (y si bien recordamos este es el sentido final de los Web Services).
- ✚ Simplicidad de sintaxis esto quiere decir que es muy fácil de escribir código en XML y la representación de los datos es casi entendible por cualquier ser humano. Esto lo hace muy flexible a la hora de querer manipular datos de cualquier especie, bastara con contar con cualquier editor de texto y aprende unas cuantas intrusiones básicas y ya está en condiciones de escribir código XML el cual será soportado o entendido por cualquier aplicación que pueda leer documentos XML. El hecho de que XML sea tan fácil de codificar y de entender lo hace el lenguaje ideal para utilizarlo en los Web Services.
- ✚ Independencia del protocolo de Transporte, el hecho de que XML es un lenguaje de Marcado de Texto, no necesita de ningún protocolo de transporte especial, solo necesita de un protocolo que pueda trasferir texto o documentos simples. Esto nos trae a la memoria que en mercado existen muchos protocolos con estas característica como lo son los más conocidos en HTTP y SMTP por nombrar algunos. Volviendo al tema de los Web Services una de las características de estos es la independencia del protocolo de transporte.

C.1.3. WSDL y la documentación de Web Services

El esquema XML por sí solo no puede describir totalmente un Web Service. Supongamos que se ha creado un Web Service Calculadora. Este Web Service expone los métodos sumar y restar. Ambos métodos aceptan dos enteros y devuelven un único entero con el resultado; sumar devuelve la suma de los dos enteros y restar devuelve su diferencia.

En un esfuerzo para describir cómo interacciona un cliente con el Web Service se define un esquema para los mensajes que se intercambiarán entre el cliente y el servidor. El esquema contiene una definición de un tipo de complejo para los mensajes de petición y repuesta para los métodos sumar y restar. Recuerde que el objetivo último es que los desarrolladores no tengan que investigar en las definiciones del esquema intentando descifrar cómo interaccionar con el Web Service. En lugar de ello se quiere describir el servicio de forma que una herramienta pueda descifrarlo y crear un proxy por el cliente.

Además de la información que proporciona el esquema, ¿Qué más necesita conocer el cliente para invocar los métodos que expone el Web Service Calculadora? Como el cuerpo de un mensaje de SOAP puede contener cualquier cosa que no invalide el XML los mensajes de SOAP se pueden combinar para disponer de una amplia variedad de patrones de intercambio de mensajes. Los patrones de intercambio de mensajes para el Web Service Calculadora son bastante inmediatos pero una asociación formal entre los mensajes de petición Sumar y Restar y sus mensajes de respuesta asociados eliminarían cualquier posible ambigüedad.

Una descripción formal de los patrones de mensaje resulta aún más importante en el caso de Web Service más complejos. Algunos servicios podrían aceptar una petición pero no enviar la respuesta correspondiente devuelta al cliente. Otros podrían solamente enviar mensajes al cliente.

Además, el esquema no contiene información sobre cómo acceder al Web Service. Como SOAP es independiente del protocolo, se intercambiarán los mensajes entre el cliente y el servidor de numerosas formas. ¿Cómo se sabe si hay que enviar un mensaje mediante HTTP, SMTP o cualquier otro protocolo de transporte? Más aún, ¿cómo se sabe la dirección la que hay que enviar el mensaje?

El lenguaje de descripción de Web Services (WSDL, Web Service Description Language) es un dialecto basado en XML sobre el esquema que describe un Web Service. Un documento WSDL proporciona la información necesaria al cliente para interaccionar con el Web Service. WSDL es extensible y se puede utilizar para describir, prácticamente, cualquier servicio de red, incluyendo SOAP sobre HTTP e incluso protocolos que no se basan en XML como DCOM sobre UDP.

Dado que los protocolos de comunicaciones y los formatos de mensajes están estandarizados en la comunidad del Web, cada día aumenta la posibilidad e importancia de describir las comunicaciones de forma estructurada. WSDL afronta esta necesidad definiendo una gramática XML que describe los servicios de red como colecciones de puntos finales de comunicación capaces de intercambiar mensajes. Las definiciones de servicio de WSDL proporcionan documentación para sistemas

distribuidos y sirven como fórmula para automatizar los detalles que toman parte en la comunicación entre aplicaciones.

Los documentos WSDL definen los servicios como colecciones de puntos finales de red o puertos. En WSDL, la definición abstracta de puntos finales y de mensajes se separa de la instalación concreta de red o de los enlaces del formato de datos. Esto permite la reutilización de definiciones abstractas: mensajes, que son descripciones abstractas de los datos que se están intercambiando y tipos de puertos, que son colecciones abstractas de operaciones. Las especificaciones concretas del protocolo y del formato de datos para un tipo de puerto determinado constituyen un enlace reutilizable. Un puerto se define por la asociación de una dirección de red y un enlace reutilizable; una colección de puertos define un servicio. Por esta razón, un documento WSDL utiliza los siguientes elementos en la definición de servicios de red:

- ✚ Types: contenedor de definiciones del tipo de datos que utiliza algún sistema de tipos (por ejemplo XSD).
- ✚ Message: definición abstracta y escrita de los datos que se están comunicando.
- ✚ Operation: descripción abstracta de una acción admitida por el servicio.
- ✚ Port Type: conjunto abstracto de operaciones admitidas por uno o más puntos finales.
- ✚ Binding: especificación del protocolo y del formato de datos para un tipo de puerto determinado.
- ✚ Port: punto final único que se define como la combinación de un enlace y una dirección de red.
- ✚ Service: colección de puntos finales relacionados.

C.1.4. WSDL y la descripción de los Web Services

WSDL [WSDL2.0-0] [WSDL2.0-1] [WSDL2.0-2] es el lenguaje de especificación utilizado para definir los Web Services. Permite proveer información acerca de la ubicación del servicio y su interfaz. Con esta información el usuario sabe cómo interactuar con el servicio. WSDL describe la interfaz de un Web Service 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 Web Service, 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 Web Service y se pretende que otras aplicaciones lo utilicen, éstas deben acceder a un documento WSDL para conocer los métodos que expone el Web Service 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. Por un lado, en la Interfaz Abstracta los dos elementos a definir son:

3) **types**: provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar, sus operaciones los parámetros de entrada o salida de cada operación.

4) **interface**: describe la secuencia de mensajes que el servicio envía o recibe.

Y por el otro, los tres elementos a definir en la implementación concreta son:

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

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

5) **endpoint**: es una URL que se usa para representar los puntos finales de los servicios.

Además, la descripción de un Web Service 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 Web Service, 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 Web Service ofrecido por un proveedor.

C.1.5. 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 Web Service 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 Web Service en términos de los mensajes que este envía y recibe. La *sección concreta* contiene especificaciones del transporte y detalles de formato.

En la primera versión de este lenguaje, WSDL 1.1 [WSDL1.1], el documento que describe un Web Service 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 Web Service.

La nueva versión de este lenguaje, WSDL 2.0 [WSDL2.0-0], hereda la mayoría de los principios arquitecturales de WSDL 1.1, incluyendo las 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 reúso.

El diagrama de la Figura C.1 presenta el Infoset XML que describe un documento WSDL 2.0.

En esta versión los elementos principales que componen el documento que describe el Web Service, 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.

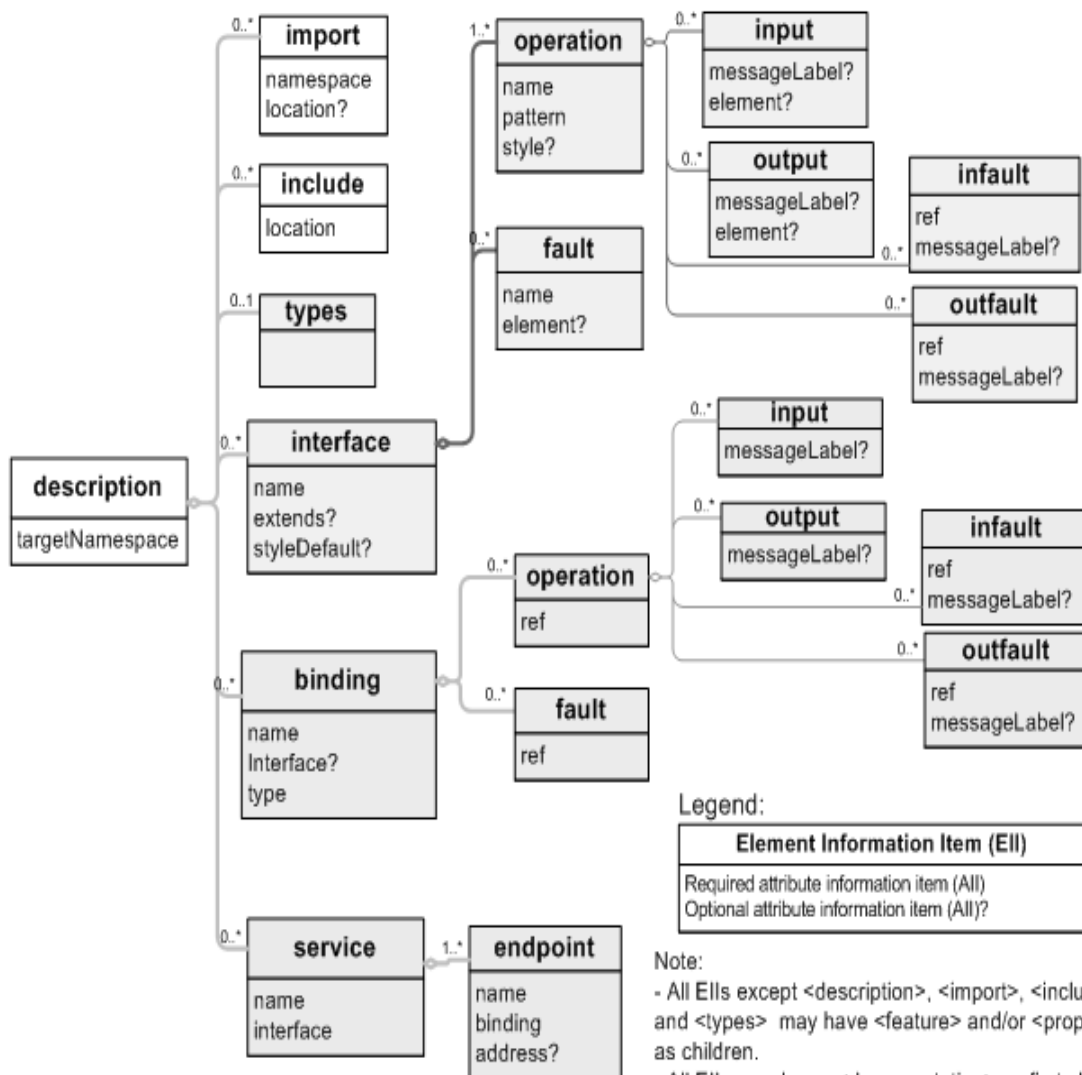


Figura C.1. Estructura del documento WSDL.

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 C.2. Elemento *description* del WSDL.

C.1.5.1. 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 C.3. Elemento *types* del WSDL.

Para definir los tipos de datos del Web Service 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.

C.1.5.2. 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 C.4. Elemento interface del WSDL.

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 los 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.

C.1.5.3. 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 C.5. Elemento *binding* del WSDL.

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 descrita 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.

C.1.5.4. El Elemento Service

El elemento *Service* asocia un binding con el URL donde está realmente el servicio a través de un *endpoint*. Los *endpoints* 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 C.6. Elemento service del WSDL.

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).

C.1.5.5. Elemento message [WSDL 1.1]

El elemento Message proporciona una abstracción común para el paso de mensajes entre el cliente y el servidor. Como puede utilizar múltiples formatos de definición de esquema en documento WSDL es necesario de disponer de un

mecanismo común de identificar los mensajes. El elemento Message proporciona este nivel común de abstracción al que se hará referencia en otras partes del documento WSDL.

Puede Aparecer, y normalmente aparecerán, múltiples elementos Message en un documento WSDL, uno para cada mensaje que se comunica entre el cliente y el servidor. Cada mensaje contiene uno o más elementos "Part" que describen las piezas del contenido del mensaje. Un ejemplo de una parte es el cuerpo de un mensaje de SOAP o un parámetro que forma parte de una cadena de petición, un parámetro codificado en el cuerpo del mensaje de SOAP o todo el cuerpo de un mensaje de SOAP.

C.1.5.6. Elemento portType [WSDL 1.1]

El elemento portType contiene un conjunto de operaciones abstractas que representan los tipos de correspondencia que pueden producirse entre el cliente y el servidor. Para los Web Services de estilo RPC se puede pensar en un portType como una definición de internas en donde cada método se puede definir como una operación.

Un tipo puerto se compone de un conjunto de electos operation que define una determinada acción. Los electos operation se componen de mensajes definidos en el documento WSDL. WSDL define cuatro tipos de operaciones como ilustra la figura C7:

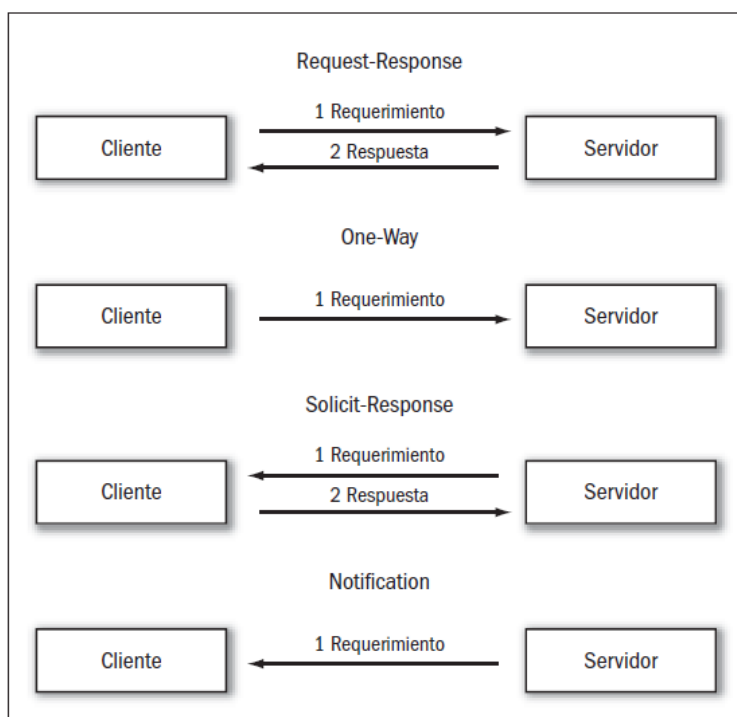


Figura C.7. Posibles operaciones de WSDL.

- Request-response (petición-respuesta). Comunicación del tipo RPC en la que el cliente realiza una petición y el servidor envía la correspondiente respuesta.
- One-way (un-sentido). Comunicación del estilo documento en la que el cliente envía un mensaje pero no recibe una respuesta del servidor indicando el resultado del mensaje procesado.

- Solicit-response (solicitud-respuesta). La contraria a la operación petición-respuesta. El servidor envía una petición y el cliente le envía de vuelta una respuesta.
- Notification (Notificación). La contraria a la operación un-sentido el servidor envía una comunicación del estilo documento al cliente.

C.1.5.7. Elementos de Extensibilidad

Los elementos de extensibilidad se utilizan para representar determinadas tecnologías. Por ejemplo, se puede utilizar los elementos de extensibilidad para especificar el idioma en que se utiliza en el esquema de los elementos types.

El esquema para un determinado conjunto de elementos de extensibilidad se debe definir dentro de distintos espacios de nombres que WSDL. La definición de los propios elementos puede contener un atributo WSDL: required que indique un valor boolean si el atributo required se establece a true en una definición de elementos una asociación que haga referencia a ese conjunto concreto de electos de extensibilidad tiene que incluir dicho elemento.

Lo más habitual es que los elementos de extensibilidad se utilicen para especificar especificación de asociación. La especificación WSDL define conjunto de elementos de extensibilidad para la asociación SOAP, HTTP GET, HTTP POST, MIME. Sin embargo, la especificación sólo define las asociaciones para dos de los cuatro tipos de operaciones. Un sentido y petición repuesta.

C.2. Universal Description Discovery and Integration

C.2.1. Estudio de UDDI

UDDI (Universal Description Discovery and Integration) está comúnmente considerado en la actualidad como la piedra angular de las Arquitecturas orientadas a Servicios (SOA), definiendo un método estándar de publicación y descubrimiento de Web Services. Esta versión del estándar se ha construido desde la visión de UDDI de un “metaservicio” de localización de Web Services, permitiendo consultas robustas ricas en metadatos. Con el acrónimo UDDI nos referimos tanto al protocolo de acceso al registro de Web Services, como al propio registro, puesto que ambas facetas se cubren en la especificación de OASIS.

Inicialmente se consideró que se podría crear un repositorio mundial de Web Services con UDDI (UBR), esta tentativa ha sido un profundo fracaso, al no ponerse de acuerdo la industria informática en la plataforma y gestión del mismo. Sin embargo, UDDI si ha tenido éxito como registro corporativo de Web Services, por ello desde la versión 2 de UDDI se ha puesto el énfasis en la interacción de registros, la integración de estándares y la clasificación flexible de los servicios, centrándose la versión actual en la seguridad y en esa integración de registros, al introducir el concepto de afiliación de registros y una nueva API de notificaciones de cambios (subscription). Actualmente existen registros UDDI estándares e interoperables para los servicios corporativos o internos a una organización; registros afiliados, con los servicios accesibles a una red de organizaciones y registros públicos, con los servicios accesibles públicamente por cualquier usuario de Internet.

El objetivo de UDDI es definir un conjunto de servicios que describan y permitan el descubrimiento de proveedores de servicios (conocida como sección blanca), de los Web Services que proporcionan (conocida como sección amarilla) y de la información técnica que permite el acceso a los mismos (conocida como sección verde). UDDI actúa, como un registro de Web Services, y no como un repositorio, puesto que sólo contiene información y referencias a servicios, no los propios servicios que pueden encontrarse en cualquier otro punto de la red, tratándose de un registro organizado en categorías, como las páginas amarillas telefónicas. UDDI se basa en varios estándares ampliamente difundidos como HTTP, XML, XML Schema (XSD), SOAP y WSDL.

Un registro UDDI está formado por nodos. Un nodo se comporta como un conjunto de Web Services que soportan, al menos, una de las APIs de nodos UDDI. Es decir, es un servidor que soporta la interacción de datos UDDI mediante, al menos, una API. Cada nodo pertenece a un único registro UDDI y conceptualmente permite el acceso y manipulación de toda la información del registro. Los registros UDDI pueden estar formados por un único nodo o por varios de ellos, por ejemplo, podemos tener un registro UDDI formado por dos nodos, uno dedicado a recibir las consultas, es decir, responde a la API inquiry y otro dedicado a la publicación y gestión del registro, respondiendo al resto de APIs. La figura C.8 muestra un posible esquema de esta configuración:

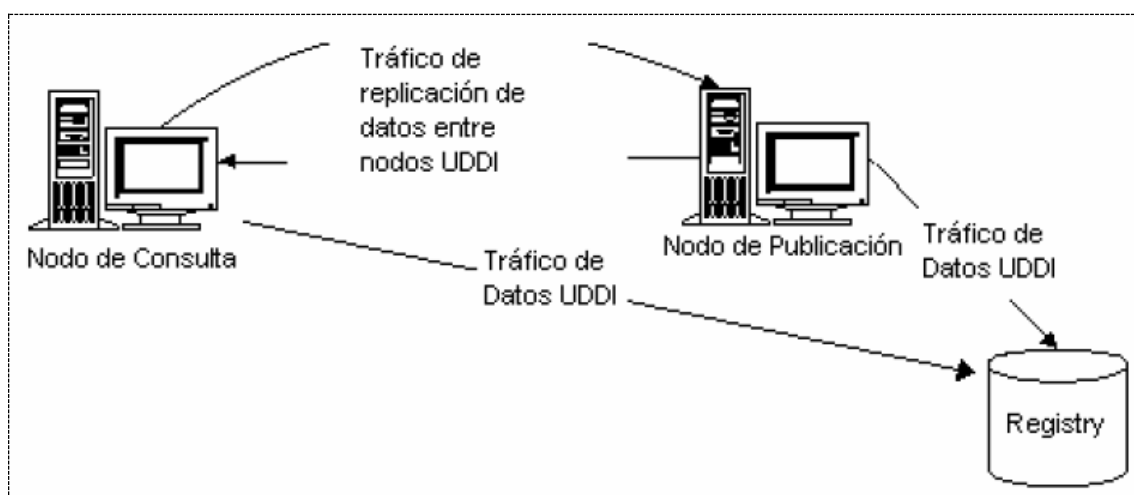


Figura C.8. Esquema del registro UDDI.

Un registro UDDI implementa toda la funcionalidad definida en la especificación y tiene que definir las políticas especificadas por UDDI, pudiendo delegar su implementación en los nodos. Las políticas soportan la flexibilidad de implementaciones que permite UDDI. En la versión 3 se ha incluido una guía de políticas con todas las decisiones sobre las políticas de cada registro UDDI [SpecUDDI3].

A partir de esta versión 3 de la especificación, actualmente la última revisión es la 3.0.2, se permite la afiliación de registros. Se trata de un mecanismo por el que varios registros pueden agruparse permitiendo la copia controlada o compartición de la información que contienen. Los registros que compartan información tienen que

definir un espacio de nombres común para las claves de la información que contienen y políticas compatibles de asignación de dichas claves. Se recomienda dar a las claves formato URI, aprovechando los nombres DNS para crear claves más comprensibles para las personas y mantener su unicidad global, facilitando el intercambio de información entre registros. La afiliación de registros permite crear una jerarquía de registros corporativos similar a la actual jerarquía de DNS.

La arquitectura de UDDI se compone de una serie de estructuras de datos que almacenan la información de las tres secciones mencionadas y un conjunto de APIs que permiten la operación con dichas estructuras de datos.

El modelo de seguridad de UDDI está formado por la colección de políticas del registro y de los nodos que lo forman, las principales áreas que recogen las políticas de seguridad son la gestión de datos, identificación y autenticación de usuarios, confidencialidad de los mensajes e integridad de datos. UDDI permite ahora la firma digital de la información publicada en el registro, mejorando la integridad del mismo. También soporta la creación de categorizaciones más complejas, extendiendo los sistemas de categorización por derivación de su esquema XSD. De una forma parecida se puede extender el modelo de información mediante la derivación de sus esquemas XSD, proporcionando reglas claras para el tratamiento de la información extendida.

C.2.2. Estructuras de Datos UDDI

Las estructuras de datos conforman un modelo de información que almacena de forma persistente en el registro todos los elementos necesarios para su funcionamiento. Estas estructuras de datos se expresan en varios esquemas XML. Cada estructura de datos tiene un identificador único o UDDI Key.

La estructura que almacena la sección blanca es de tipo *businessEntity*, esta estructura describe a un proveedor de Web Services. Para almacenar la sección amarilla se utiliza la estructura tipo *businessService*, que describe una familia de Web Services ofrecidos por el proveedor descrito en el *businessEntity*.

La sección verde se almacena entre las estructuras tipo *bindingTemplate*, que describen la información técnica de acceso a un Web Service concreto y las estructuras tipo *tModel*, que describen modelos técnicos o metadatos reutilizables que pueden representar cualquier concepto como el tipo de Web Service, algún protocolo usado por los servicios o un sistema de categorización.

A continuación se muestra un ejemplo de una entidad *businessEntity*, conteniendo un *businessService* con su *bindingTemplate*:

```
<uddi:businessEntity xmlns:uddi="urn:uddi-org:api_v3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
  xsi:schemaLocation="urn:uddi-org:api_v3 uddi_v3.xsd"
  businessKey="uddi:informatica.us.es">
  <uddi:discoveryURLs>
    <uddi:discoveryURL>
      www.informatica.us.es
    </uddi:discoveryURL>
  </uddi:discoveryURLs>
  <uddi:name xml:lang="ES-es">
```



```

        Escuela Técnica Superior de Ingeniería Informática
    </uddi:name>
    <uddi:name xml:lang="EN-us">
        High Technical Institute of Computer Science
    </uddi:name>
    <uddi:description xml:lang="ES-es">
        Centro de Estudios de la Universidad de Sevilla
    </uddi:description>
    <uddi:description xml:lang="EN-us">
        High Studies Center of the Seville University
    </uddi:description>
    <uddi:contacts>
        <uddi:contact>
            <uddi:description xml:lang="ES-es">
                Catedrático del Departamento de Lenguajes y
                Sistemas Informáticos
            </uddi:description>
            <uddi:personName>
                Miguel Toro Bonilla
            </uddi:personName>
            <uddi:phone>
                954552778
            </uddi:phone>
            <uddi:email>
                mtoro@informatica.us.es
            </uddi:email>
            <uddi:address>
                <uddi:addressLine>
                    Avenida de Reina Mercedes, s/n
                </uddi:addressLine>
                <uddi:addressLine>
                    41012 Sevilla
                </uddi:addressLine>
                <uddi:addressLine>
                    Edificio L4. Despacho F1.76
                </uddi:addressLine>
            </uddi:address>
        </uddi:contact>
    </uddi:contacts>
    <uddi:businessServices>
        <uddi:businessService
            businessKey="uddi:informatica.us.es">
            <uddi:name xml:lang="ES-es">
                Material Docente
            </uddi:name>
            <uddi:description xml:lang="ES-es">
                Servicio que proporciona material
                docente de cualquier asignatura en la
                que se encuentre matriculado un alumno
            </uddi:description>
            <uddi:bindingTemplates>
                <uddi:bindingTemplate>
                    <uddi:accessPoint useType="endPoint">
                        http://www.informatica.us.es/swMatDoc
                    </uddi:accessPoint>
                    <uddi:tModelInstanceDetails>
                        <uddi:tModelInstanceInfo
                            tModelKey="uddi:swMatDoc.informatica.us.es">
                            <uddi:description xml:lang="ES-es">
                                Modelo Técnico del acceso al Sv Web
                                de suministro de Material Docente de la

```

```

        ETSII de la Universidad de Sevilla
    </uddi:description>
    <uddi:instanceDetails>
        <uddi:instanceParms>
            CodAlumno CodAsignaturaOpcional
        </uddi:instanceParms>
    </uddi:instanceDetails>
    </uddi:tModelInstanceInfo>
    </uddi:tModelInstanceDetails>
    </uddi:bindingTemplate>
</uddi:bindingTemplates>
</uddi:businessService>
</uddi:businessServices>
<uddi:identifierBag>
    <uddi:keyedReference
        tModelKey="uddi:tMIdentFiscal.minhac.es"
        keyName="CIF_ETSI"
        keyValue="S-4111012-F"/>
</uddi:identifierBag>
<uddi:categoryBag>
    <uddi:keyedReference
        tModelKey="tM_NAICS.uddi.org"
        keyName="Universidades_Sevilla"
        keyValue="611310"/>
</uddi:categoryBag>
</uddi:businessEntity>

```

Además de estas entidades existen otras de primer nivel como *publisherAssertion* y *subscription*. *publisherAssertion* describe relaciones entre *bussinnessEntity*, mientras que *subscription* describe un mecanismo de notificación y transferencia de cambios sobre entidades.

Existe una relación jerárquica padre-hijo entre *bussinnessEntity*, *bussinnessService* y *bindingTemplate*. Un *bussinnessEntity* contiene información descriptiva del proveedor y de los servicios que ofrece, a través de los *bussinnessService* y de los *bindingTemplate* que dependen de él. El *bussinnessEntity* contiene el nombre del proveedor, una descripción, una URL de referencia, información de contactos y de clasificación de su actividad. Toda la información admite multiplicidad, bien para concretarla en varios idiomas, bien porque la propia información sea múltiple, como los contactos o las actividades del proveedor de servicios. Las entidades de datos y las principales APIs de UDDI, publicación, consulta y seguridad, se recogen en el espacio de nombres urn:uddi-org:api_v3 (UDDI), esquema uddi_v3.xsd.

La actual versión del estándar admite la publicación de entidades de un registro en otro registro UDDI, este mecanismo se conoce como promoción de la entidad y permite compartir información entre diferentes registros. Para que sea posible la promoción de entidades ambos registros deben compartir la política de gestión de claves, puesto que éstas deben ser únicas.

C.2.2.1. BusinessEntity

La estructura *bussinnessEntity* tiene un atributo opcional que es su clave, el *bussinnessKey*, este atributo, de tipo *uddiKey*, identifica unívocamente a la entidad, es opcional porque el publicador puede dejar que sea el propio registro UDDI el que la genere. A la hora de recuperar la estructura del registro, la clave siempre debe

contener un valor. La estructura de la entidad se muestra en la figura C.9. A continuación se describen los componentes de la estructura:

- ✚ **discoveryURLs**, contiene una secuencia no vacía de URLs que apuntan a mecanismos alternativos, basados en ficheros, de descubrimiento de los servicios. Cada URL (discoveryURL) puede tener un atributo useType opcional de tipo cadena que contiene el nombre del formato de los ficheros de descubrimiento. También se puede utilizar para almacenar la URL del sitio del proveedor de servicios. El atributo useType no se define como un atributo habitual de cualquier elemento XML, sino que se define como una extensión (extensión) de una URI de hasta 4096 caracteres de longitud. Por lo que el elemento discoveryURL tiene una estructura formada por el contenido simple (simpleContent) de esa extensión. La URI no es más que una cadena de caracteres. El atributo useType del contacto describe el tipo de contacto en formato texto. Este atributo es común a muchas estructuras de UDDI. Siempre se trata de texto no estructurado con fines descriptivos de la estructura a la que se asocia, es decir, proporciona información del tipo de la misma, a veces la especificación define la semántica de algunos valores reservados, en otras ocasiones sugiere ejemplos de uso, también se pueden definir valores propios, recomendándose en este caso asociarlos con los correspondientes modelos técnicos que aclaren su significado.
- ✚ **name**, contiene el nombre del proveedor de servicios, puede expresarse en múltiples idiomas simultáneamente, especificando el idioma a través del atributo xml:lang, común a otras muchas estructuras de UDDI que soportan múltiples idiomas. Este atributo, al igual que useType aparece en el esquema XML como una extensión, por lo que este elemento tiene una estructura compleja, formada por el contenido simple (simpleContent) de la extensión (extensión) de este atributo, el tipo base que extiende es una cadena de hasta 255 caracteres.
- ✚ **description**, contiene una descripción del proveedor de servicios, opcionalmente multidioma.

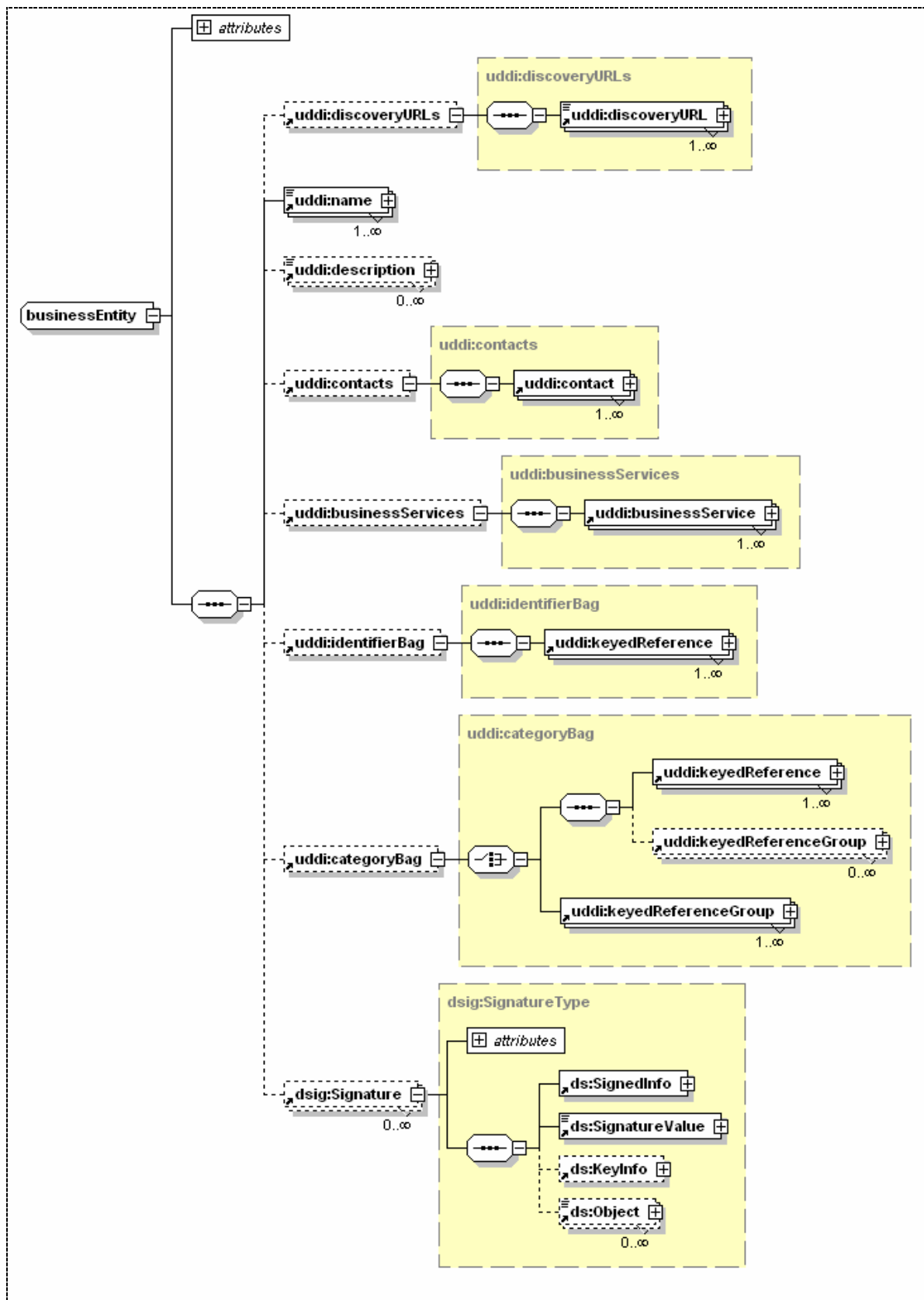


Figura C.9. Estructura bussinessEntity

- ✚ **contacts**, contiene una secuencia no vacía de contactos de personas o puestos de trabajo del proveedor de servicios. Cada contacto contendrá, al menos, el nombre de una persona o puesto, pudiendo contener además descripciones (para indicar la forma de usar el contacto), teléfonos, direcciones de correo electrónico y postales, compuestas estas últimas por varias líneas. La dirección postal puede tener un atributo (tModelKey) que identifique el tModel que especifica su formato y/o el de las líneas que lo forman. La estructura del contacto es ilustrada en la figura C.10:

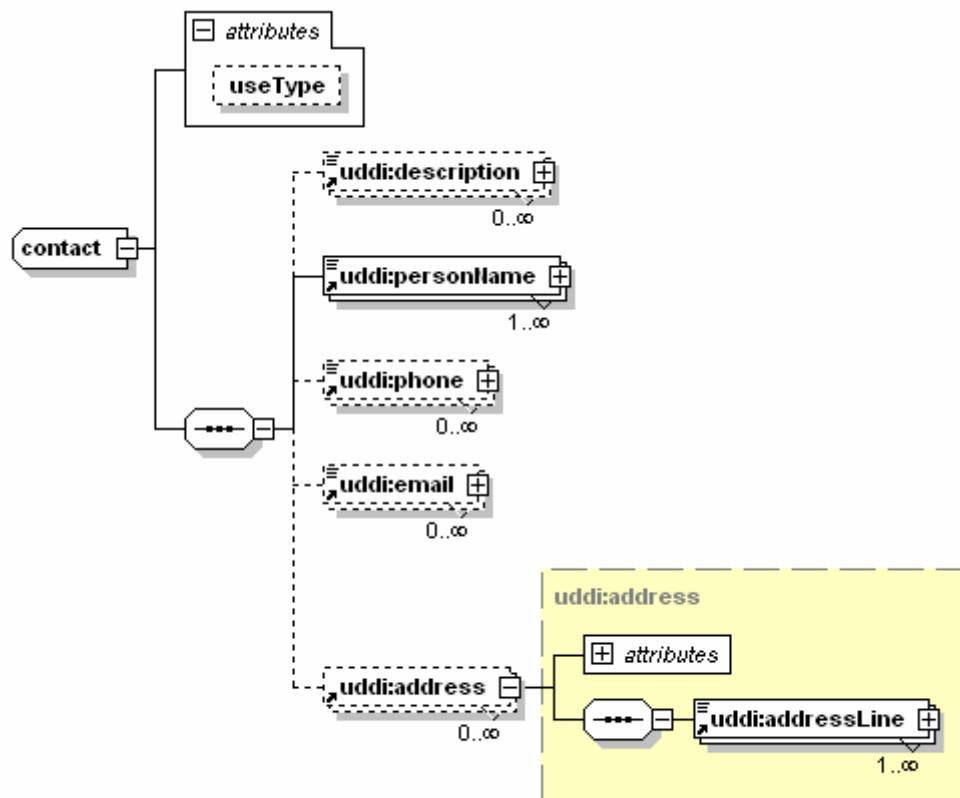


Figura C.10. Estructura contact de bussinessEntity.

- ✚ **identifierBag**, permite identificar a los proveedores de servicios conforme a sistemas de identificación públicos como Bun & Bradstreet D-U-N-S Numbers, identificaciones fiscales como el CIF, etc. Se trata de una secuencia no vacía de estructuras keyedReference, donde cada una representa una identificación. La estructura keyedReference está formada por los atributos tModelKey, keyName (opcional) y keyValue. El primero identifica el modelo técnico seguido para representar un sistema de codificación, en este caso el sistema de identificación, keyValue contiene un valor concreto de la clave dentro del sistema de codificación, en este caso se trata del identificador dentro de ese sistema de identificación asociado al proveedor de servicios. El nombre puede usarse para dar un nombre descriptivo al valor, en este caso un identificador. Esta estructura se utiliza mucho a lo largo de los esquemas XML de UDDI, permitiendo hacer referencia a un valor concreto dentro de cualquier sistema de codificación, ya sea identificadores, categorías empresariales o técnicas, etc. Se trata, por tanto, de una estructura sintáctica con semántica muy variada, igual que el tModel. Puede

verse la estructura de este elemento en la Figura C.11. Por ejemplo, la identificación de SAP AG en D-U-NS es la siguiente:

```
<identifierBag>
  <keyedReference
    tModelKey="uddi:uddi.org:ubr:identifier:dnb.com:d-u-n-s"
    keyName="SAP AG"
    keyValue="31-626-8655" />
</identifierBag>
```

Figura C.11. Estructura de identifierBag de BussinessEntity.

categoryBag, permite categorizar a los proveedores de servicios conforme a sistemas de categorización publicados, como la categorización de productos y servicios UNSPSC o el sistema ISO 3166 que describe el área geográfica de los productos y/o servicios ofrecidos. Puede contener una simple secuencia de referencias codificadas (keyedReference) como el elemento anterior, en esta ocasión haciendo referencia a sistemas de categorización; una secuencia de referencias codificadas junto con referencias codificadas agrupadas (keyedReferenceGroup), que a su vez, no son más que una secuencia de referencias codificadas que se mantienen agrupadas por mantener una relación lógica entre ellas; o sólo una secuencia de referencias codificadas agrupadas. La secuencia de referencias codificadas agrupadas contiene un atributo tModelKey que identifica el modelo técnico que describe la estructura y el significado de las referencias del grupo. Por ejemplo, para categorizar un proveedor de servicios localizado en la latitud 49.6827 N y la longitud 8.2952 O, según el sistema de coordenadas WGS84 (GPS) se utilizaría la estructura de la figura C.12:

```
<keyedReferenceGroup tModelKey="uddi:uddi.org:ubr:categorizationGroup:wgs84">
  <keyedReference
    tModelKey="uddi:uddi.org:ubr:categorizationGroup:wgs84:latitude"
    keyName="WGS 84 Latitude"
    keyValue="+49.682700" />
  <keyedReference
    tModelKey="uddi:uddi.org:ubr:categorizationGroup:wgs84:longitude"
    keyName="WGS 84 Longitude"
    keyValue="+008.295200" />
</keyedReferenceGroup>
```

Figura C.12. Estructura de categoryBag de BussinessEntity.

dsig:Signature, es la estructura que permite almacenar la información de la firma digital XML del proveedor de servicio. Esta firma digital proporciona integridad a la información registrada.

C.2.2.2. *BusinessService*

La estructura *businessService* representa un servicio lógico, conteniendo información descriptiva empresarial del servicio. La información técnica se encuentra en los *bindingTemplate*. El *businessService* tiene dos atributos opcionales, el identificador del proveedor de servicios, *businessKey* y el identificador del servicio, *serviceKey*, que funciona de forma análoga al *businessKey* en el proveedor de servicios, en cuanto a su opcionalidad. La estructura del servicio es mostrado en la figura C.13:

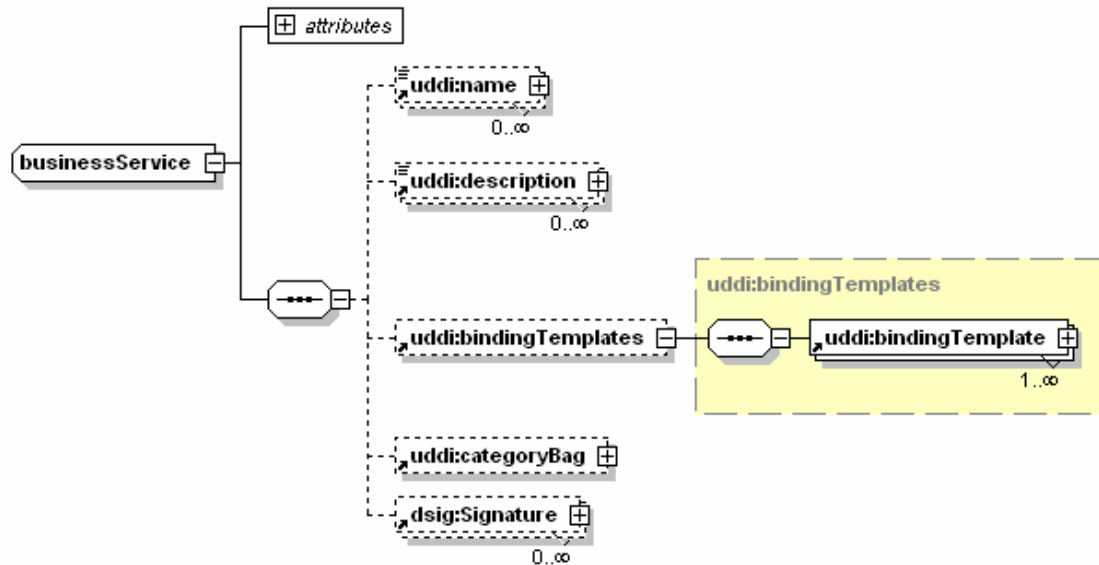


Figura C.13. Estructura de *BusinessService*.

Como puede apreciarse todos los elementos son opcionales, la única estructura no comentada es el *bindingTemplate*.

C.2.2.3. *BindingTemplate*

Contiene las descripciones técnicas de los Web Services. Cada uno describe una instancia de un Web Service ofrecido normalmente a través de una URL, también describe el tipo de Web Service ofrecido utilizando modelos técnicos, parámetros de aplicaciones específicas y diferentes configuraciones. Cada *bindingTemplate* está contenido en un *businessService* y tiene dos atributos opcionales, uno identifica al servicio (*serviceKey*) y otro al propio *bindingTemplate* (*bindingKey*). La estructura *categoryBag* del *bindingTemplate*, contiene las categorizaciones que describen aspectos concretos del mismo, puede utilizarse, por ejemplo, para distinguir servicios en producción de servicios en pruebas. La estructura de *bindingTemplate* es como ilustra la figura C.14:

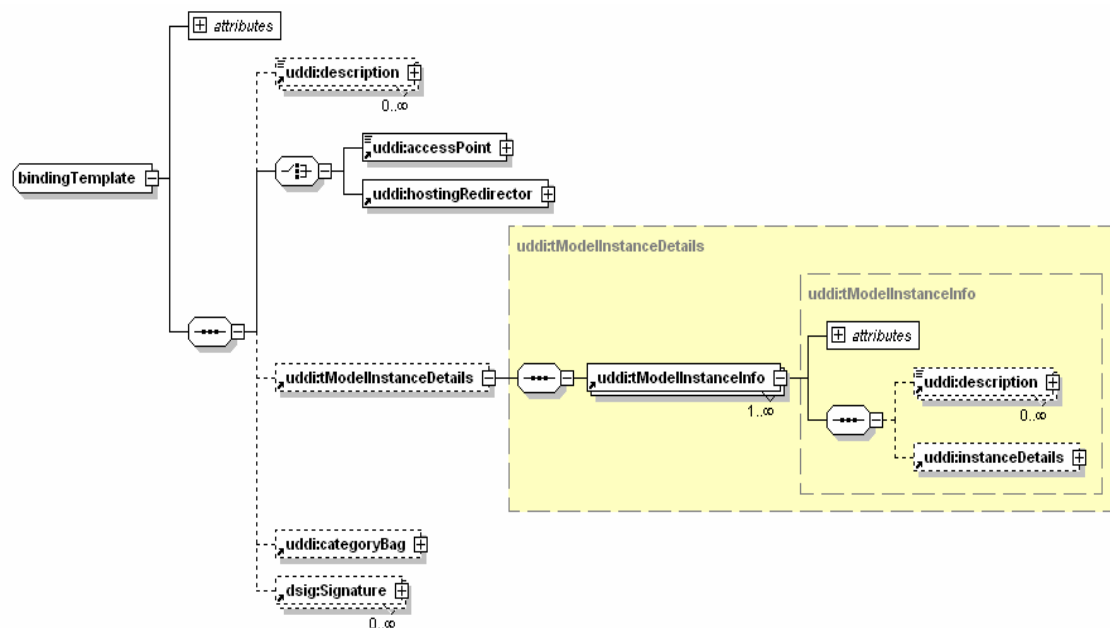


Figura C.14. Estructura de BindingTemplate.

- ✚ **accessPoint**, se trata de una cadena que contiene la URI (normalmente una URL) de invocación al servicio. Su atributo useType describe el tipo de punto de acceso, algunos valores predefinidos son: endPoint, que indica que el punto de invocación final del servicio se encuentra en este elemento; bindingTemplate, que indica que el punto de acceso contiene la bindingKey que apunta a un punto de acceso de otro bindingTemplate; hostingRedirector, que indica que el punto de acceso se determina consultando otro registro UDDI concretado en el punto de acceso; y wsdlDeployment, que indica que el punto de acceso se determina a través de un fichero WSDL remoto cuya localización se encuentra en el punto de acceso.
- ✚ **hostingRedirector**, es un elemento en desuso, actualmente se recomienda usar el elemento anterior.
- ✚ **tModelInstanceDetails**, es una secuencia no vacía de elementos tModelInstanceInfo. Cada tModelInstanceInfo tiene un atributo tModelKey que referencia al tModel que representa una especificación que cumple el servicio. El conjunto de todos los tModelInstanceInfo, cada uno con su atributo tModelKey, forman la “huella técnica” del Web Service, que puede utilizarse para identificar servicios compatibles, puesto que representan el conjunto de especificaciones que cumple el servicio. La estructura tModelInstanceInfo se enseña en la figura C15:
- ✚ **instanceDetails**, puede utilizarse para especificar alguna configuración concreta del tModel u otra información descriptiva requerida como parámetros. Se trata de información específica de la instancia del servicio. El overviewDoc se usa para almacenar referencias a descripciones generales remotas o instrucciones relativas

al tModel y los parámetros de la instancia. La descripción obligatoria, opcionalmente multi-idioma, describe brevemente la forma de usar el tModel. El overviewURL opcional se utiliza para almacenar la URL de un formulario largo o de un documento general que cubra la forma de uso del tModel, se utiliza como un componente más de la descripción del Web Service. El formato recomendado es una URI que admita una operación HTTP GET. La URL tiene un atributo useType que indica el tipo de documento referenciado: texto (text), interfaz WSDL (wsdl Interface), etc. Este atributo está definido como contenido simple del elemento, en forma de extensión del tipo básico URI con una longitud máxima de 4096 caracteres. El elemento opcional instanceParams es una cadena utilizada para almacenar localmente parámetros relativos al uso correcto del tModelInstanceInfo. El formato recomendado es el de un espacio de nombres cualificado de documento XML, donde los parámetros se encuentren en los elementos y atributos de los documentos XML.

C.2.2.4. tModel

Se trata de una estructura que permite describir especificaciones, conceptos o diseños compartidos, proporcionan un sistema de referencia basado en abstracciones. Se utilizan también como fuente para determinar la compatibilidad entre servicios y como referencias a espacios de nombres o sistemas de codificación de cualquier tipo. Se referencian en varias entidades y APIs. Representan especificaciones técnicas o conceptos, como formatos de intercambio, protocolos, etc. El conjunto de documentos de la especificación no forma parte del registro UDDI, no se almacena en él. Los publicadores deben dar un formato conocido y utilizar lenguajes descriptivos en esos documentos. Una vez publicado el modelo técnico cualquiera puede expresar que un servicio cumple con él incluyendo una referencia al mismo, a través de un atributo tModelKey en su descripción técnica.

Dentro de identifierBag, categoryBag, address y publisherAssertion se utilizan para especificar sistemas de identificación y de categorización o clasificación organizativas. Representan conjuntos de valores (value set) utilizados para identificar o clasificar entidades UDDI.

Otro uso de los modelos técnicos es representar calificadores de búsquedas (findQualifiers), que son valores que modifican el funcionamiento de las funciones find_xxx de la API inquiry.

Un tModel contiene dos atributos opcionales, tModelKey, que lo identifica unívocamente y un indicador booleano de borrado lógico del registro (deleted). El nombre (name) del tModel debe tener formato URI y no debe utilizarse su atributo xml:lang, es decir no se admite utilizar varios idiomas para nombrar un tModel. La estructura de un tModel, cuyos elementos han sido ya todos descritos previamente en la sección de BusinessEntity, salvo el overviewDoc, que ha sido descrito en la sección de bindingTemplate, es como la que se ilustra en la figura C.15:

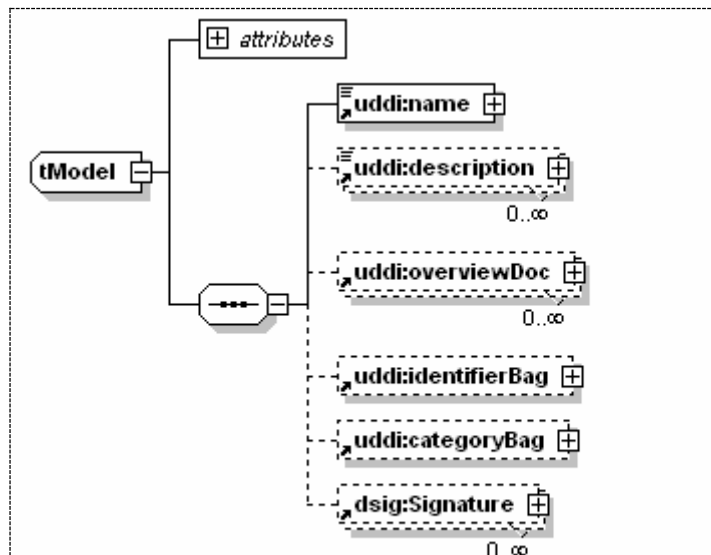


Figura C15. Estructura del tModel.

C.2.2.5. PublisherAssertion

La última estructura de datos importante de UDDI es publisherAssertion, que permite representar y almacenar relaciones entre proveedores de servicio en el registro UDDI. Deben publicarse por las dos entidades relacionadas. Su estructura es la siguiente:

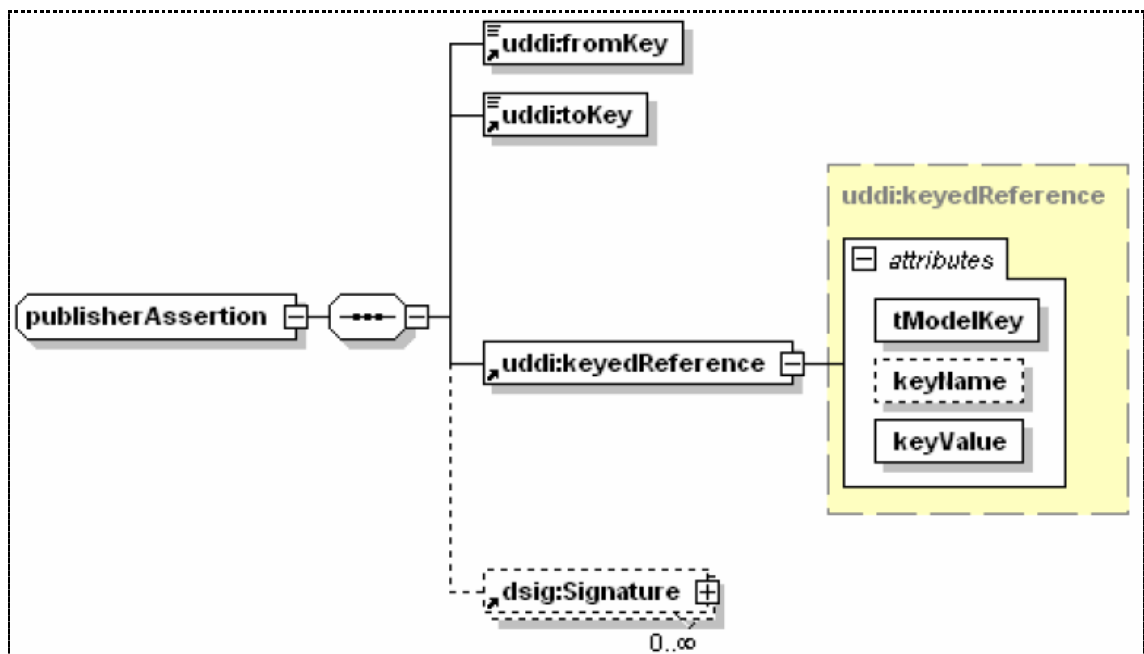


Figura C.16. Estructura de publisherAssertion.

- ✚ **fromKey** y **toKey**, son las claves de los proveedores de servicios relacionados.
- ✚ **keyedReference**, describe la relación. A través de su correspondiente tModelKey, se especifica el tipo de relación, el keyValue designa la relación concreta entre los

proveedores de servicio, pudiendo dar un nombre a cada relación que se almacena en el keyName. Esta estructura ha sido comentada en la sección C.2.2.1 businessEntity, concretamente en la estructura identifierBag.

Cuando se realiza una publicación de una entidad UDDI, se captura la información del momento de la publicación en una estructura contenida en la entidad publicada. Esta estructura es operationalInfo, y contiene la fecha y hora de creación y modificación de la entidad, created, (instante en que la entidad aparece en el registro por primera vez) y modified (instante del último cambio realizado sobre la entidad); el identificador del nodo UDDI donde se publicó, nodeID; y el identificador del proveedor de servicio que lo publicó, a través del atributo obligatorio entityKey. El elemento modifiedIncludingChildren, permite conocer el momento de la última modificación de la estructura completa de la entidad, incluidas las estructuras que dependan de ella. El elemento authorizedName indica el propietario de la entidad publicada, su contenido e interpretación depende de las políticas del registro. Se accede a la estructura con la función get_operationalInfo de la API inquiry. La estructura operationalInfo es mostrada en la figura C.17:

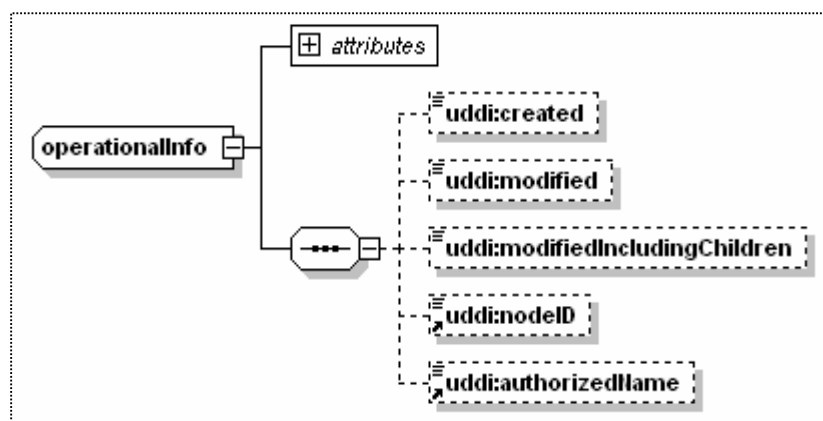


Figura C.17. Estructura de operationalInfo.

C.2.3 APIs UDDI

Las APIs permiten estandarizar el comportamiento y la comunicación entre registros UDDI y entre los clientes y los registros UDDI. UDDI proporciona APIs de publicación o registro de servicios y de consulta o búsqueda y obtención de esos servicios registrados, además de otras menos utilizadas.

Las APIs de los nodos UDDI son las siguientes: Inquiry, Publication, Security, Custody & Ownership Transfer, Subscription y Replication, utilizada en la operación entre los nodos de un registro para transferir información y notificar dichas transferencias. Las APIs de los clientes son: Subscription Listener (incluida en la API Subscription) y Value Set. La implementación de estas dos últimas APIs es opcional para todos los nodos y registros, de hecho la implementan los clientes, es decir, lo que es opcional para UDDI es su invocación.

Las funciones de las APIs pueden producir mensajes de error informando de cualquier circunstancia anómala a través de fallos SOAP, o errores de aplicación,

normalmente, mediante un elemento dispositionReport que tiene la estructura que muestra la figura C.18:

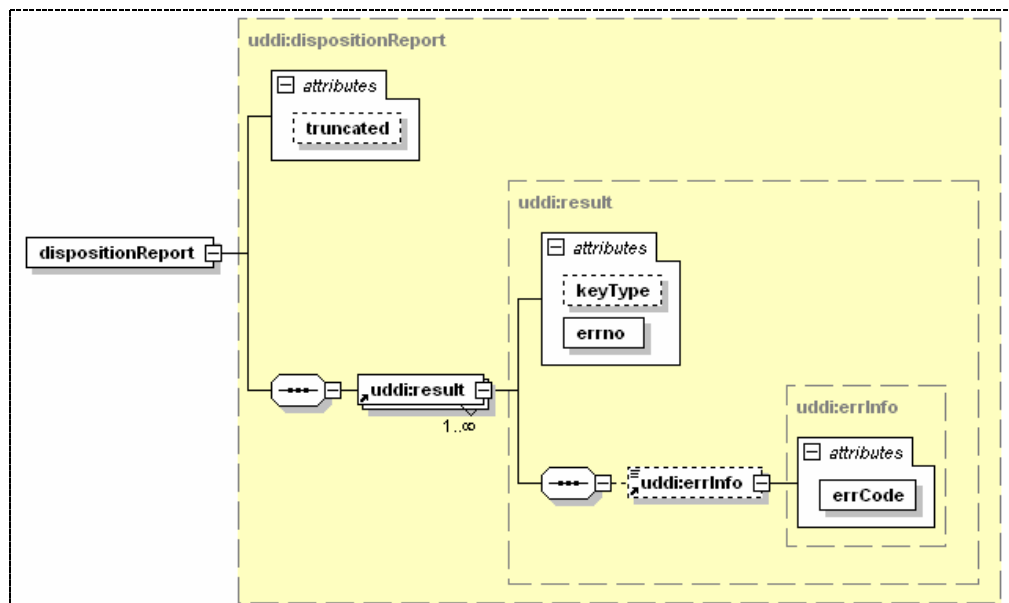


Figura C.18. Estructura del elemento dispositionReport.

Esta estructura contiene la información producida por las funciones de las APIs, fundamentalmente el código de error, errno. keyType hace referencia al tipo de clave que está siendo reportado, puede ser: bussinesKey, serviceKey, bindingKey, tModelKey o subscriptionKey. Se puede dar información adicional en forma de cadena (errInfo) repitiendo el código de error en un atributo de la misma. Cada función de las APIs puede producir algunos códigos de error descritos en el capítulo 12 de la especificación [SpecUDDI3].

Los errores se empaquetan dentro del error 500 “Internal Server error” de HTTP. Deben producirse antes de procesar las peticiones y las anulan por completo.

Cuando una llamada devuelve una secuencia como resultado, se puede devolver también un elemento de descripción de la secuencia (listDescription), formado por una secuencia con el número de elementos devueltos (includeCount), el número de resultados disponibles actualmente (actualCount) y un índice al primer elemento de la secuencia de resultados (listHead). No se trata de un cursor, puesto que la información puede cambiar entre una consulta y la siguiente.

El proceso de búsqueda o consulta consiste en hacer coincidir los requisitos indicados en la petición o llamada a la función de la API, con la información del registro. Este emparejamiento puede complicarse en el caso de tipos de datos complejos como keyedReferences en categoryBags e identifiersBags realizándose el emparejamiento cuando las tModelKeys coinciden, es decir, se refieren al mismo tModel y coincide el keyValue, ignorándose el keyName.

A continuación se muestra un resumen de las funciones de las APIs UDDI:

✚ API Inquiry:

- `find_business`, permite localizar proveedores de servicios.
- `find_service`, permite localizar servicios.
- `find_binding`, permite localizar los detalles técnicos de un servicio.
- `find_tModel`, permite localizar un modelo técnico.
- `find_relatedBusinesses`, permite localizar relaciones.
- `get_businessDetail`, localiza `businessEntities` a partir de su clave.
- `get_serviceDetail`, localiza `businessServices` a partir de su clave.
- `get_bindingDetail`, localiza `bindingTemplates` a partir de su clave.
- `get_tModelDetail`, localiza `tModels` a partir de su clave.
- `get_operationalInfo`, recupera `operationalInfo` a partir de su clave.

API Publication

- `add_publisherAssertions`, añade relaciones.
- `get_assertionStatusReport`, informa del estado de las relaciones.
- `get_publisherAssertions`, recupera las relaciones del publicador.
- `set_publisherAssertions`, actualiza las relaciones del publicador.
- `delete_publisherAssertions`, elimina las relaciones especificadas.
- `get_registeredInfo`, resume los proveedores de servicios y los `tModels`.
- `save_business`, publica un proveedor de servicios.
- `save_service`, publica un servicio.
- `save_binding`, publica los detalles técnicos de un servicio.
- `save_tModel`, publica un modelo técnico.
- `delete_business`, elimina un proveedor de servicios.
- `delete_service`, elimina un servicio.
- `delete_binding`, elimina los detalles técnicos de un servicio.
- `delete_tModel`, elimina un modelo técnico.

API Security Policy

- `get_authToken`, obtiene un token de autenticación.
- `discard_authToken`, invalida un token de autenticación.

API Custody & Ownership Transfer

- `get_transferToken`, obtiene autorización para transferir datos.
- `transfer_entities`, transfiere datos entre nodos.

- `discard_transferToken`, invalida la autorización de transferencia.
- `transfer_custody`, el nodo informa de la correcta recepción de los datos.

API Subscription

- `save_subscription`, registra una suscripción.
- `get_subscription`, recupera las suscripciones del solicitante.
- `get_subscriptionResults`, recupera la información suscrita.
- `notify_subscriptionListener`, el nodo envía la información suscrita.
- `delete_subscription`, cancela la suscripción especificada.

API Value Set

- `validate_values`, valida externamente un valor de un conjunto de valores.
- `validate_allValidValues`, obtiene todos los valores válidos de un conjunto.

API Replication

- `get_changeRecords`, el nodo destino solicita la recepción de información.
- `notify_changeReordsAvaible`, un nodo informa de cambios disponibles.

A continuación se muestran un par de ejemplos simples de invocación de funciones de la API Inquiry:

Ejemplo 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <find_business
      xmlns="urn:uddi-org:api"
      generic="1.0"
      maxRows="100">
      <findQualifiers/>
      <name>Microsoft</name>
    </find_business>
  </Body>
</Envelope>
```

Ejemplo 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <get_businessDetail xmlns="urn:uddi-org:api" generic="1.0">
      <businessKey>
```

```
c13cc7b2-642d-41d0-b2dd-7bb531a18997
</businessKey>
</get_businessDetail>
</Body>
</Envelope>
```




En ambos ejemplos de invocaciones, se observa la utilización de claves en formato UDDI clásico, no en el formato URI actualmente recomendado.

C.2.3.1. API Inquiry

Las funciones de la API Inquiry son todas síncronas y pueden exponerse vía HTTP POST. Las 10 funciones de esta API son: find_binding, find_business, find_relatedBusinesses, find_service, find_tModel, get_bindingDetail, get_businessDetail, get_operationalInfo, get_serviceDetail y get_tModelDetail.

Esta API permite localizar y obtener los detalles de las entradas de un registro UDDI mediante tres tipos de consulta diferentes, dependiendo de la función de la API que se invoque.

Los tipos de consulta pueden ser:

-  **hojeada** (browse), que permite explorar y examinar grandes cantidades de datos. Se trata de una búsqueda descendente que comienza con gran cantidad de información encontrando resultados generales que se seleccionan para llegar a información más específica. Se realiza a través de las operaciones find_xxx, que devuelven información general sobre las estructuras. Donde xxx puede ser business, service, binding, tModel o relatedBusinesses.
-  **búsqueda descendente.** Cada instancia de las estructuras importantes de UDDI, businessEntity, businessService, bindingTemplate y tModel tiene una clave que se devuelve en la información general de la entidad devuelta por una operación find_xxx. Con esta clave se pueden recuperar los detalles de la entidad a través de las operaciones get_xxx.
-  **invocación,** permite la conexión dinámica con Web Services, obteniendo la información técnica del mismo a través de su bindingTemplate, mediante las formas de consulta anteriores.

C.2.3.2. API Publication

Esta API se utiliza para publicar y modificar la información del registro. Cada llamada debe implementarse síncrona y atómicamente, como una transacción. Esta API asigna las claves a las entidades que el publicador no suministra en el momento de su publicación, de acuerdo con las políticas del registro. Cada clave debe ser única en todo el registro, independientemente de la entidad a la que pertenezca y además se recomienda que sea única globalmente, independientemente del registro en el que se almacene, para facilitar la interoperabilidad. El formato de clave recomendado es el URI que garantiza esta unicidad a través del DNS. El registro UDDI debe implementar un generador de claves que cumpla la unicidad mencionada. La gestión de claves es un

tema crítico en UDDI, sobre todo en la tercera versión de la especificación, que fomenta la interacción entre diferentes registros.

En varias funciones de esta API se almacenan conjuntos de valores (value set) que deben ser chequeados, como identificaciones, categorizaciones o relaciones entre proveedores de servicio. Este chequeo debe realizarlo esta API antes de la grabación, no permitiéndola si los valores no superan el chequeo. Estos conjuntos de valores también pueden definirse como no chequeables, con lo que se evita el chequeo en la publicación.

Las 14 funciones de esta API son: `add_publisherAssertions`, `delete_binding`, `delete_business`, `delete_publisherAssertions`, `delete_service`, `delete_tModel`, `get_assertionStatusReport`, `get_publisherAssertions`, `get_registeredInfo`, `ave_binding`, `save_business`, `save_service`, `save_tModel` y `set_publisherAssertions`. Veamos estas funciones en detalle.

La función `add_publisherAssertions`, permite añadir relaciones entre proveedores de servicio, que toman la forma de una secuencia no vacía de `publisherAssertion` pasada como argumento (Ver Figura 7). El publicador debe ser propietario de algún `businessEntity` de la relación o de ambos. Cuando el publicador sólo es propietario de un extremo de la relación, debe existir otra relación publicada por el propietario del otro extremo. Devuelve un error si no se pudo añadir la relación y un mensaje vacío si tuvo éxito.

La función `delete_binding`, borra los `bindingTemplate` identificados por la secuencia no vacía de `bindingKey` pasada como argumento. Devuelve un mensaje vacío si tuvo éxito.

La función `delete_business`, borra los `businessEntity` identificados por la secuencia no vacía de `businessKey` pasada como argumento. Devuelve un mensaje vacío si tuvo éxito.

La función `delete_publisherAssertions`, borra los `publisherAssertion` identificados por la secuencia no vacía de ellos pasada como argumento. Devuelve un mensaje vacío si tuvo éxito.

La función `delete_service`, borra los `businessService` identificados por la secuencia no vacía de `serviceKey` pasada como argumento. Devuelve un mensaje vacío si tuvo éxito.

La función `delete_tModel`, borra los `tModel` identificados por la secuencia no vacía de `tModelKey` pasada como argumento. Devuelve un mensaje vacío si tuvo éxito.

La función `get_assertionStatusReport`, permite determinar el estado de las relaciones en las que participan proveedores de servicios del publicador. Tiene como argumento una secuencia compuesta por dos elementos opcionales, `authInfo` y `completionStatus` que restringe los resultados a las relaciones que tengan el estado indicado. Los valores posibles son: `status: complete`, `status: toKey_incomplete` (devolvería sólo las relaciones a las que le falte la parte `toKey`), `status: fromKey_incomplete`, `status: both_incomplete` (se aplica sólo en el contexto de la suscripción UDDI). La función devuelve una estructura `assertionStatusReport` que es una secuencia de `assertionStatusItem` ordenada por la fecha de la última modificación.

La función `get_publisherAssertions`, obtiene todas las relaciones asociadas a un publicador, su único argumento es el parámetro opcional común a todas las funciones de las APIs de publicación, suscripción, transferencia de custodia y propiedad y consulta, `authInfo`, que contiene un token de autenticación. Devuelve una secuencia de `publisherAssertion`.

La función `get_registeredInfo`, proporciona un listado abreviado de los proveedores de servicio y los modelos técnicos controlados por un publicador. El publicador es el usuario autenticado y dado de alta como tal en el registro que invoca la función. Tiene un atributo `infoSelection` obligatorio que representa la cantidad de `tModels` que deben devolverse, la lista de valores es: `all`, `visible` (devuelve los que no se hayan eliminado lógicamente) y `hidden` (devuelve los que hayan sufrido un borrado lógico del registro). Sólo tiene el argumento `authInfo` y devuelve una estructura `registeredInfo`, que es una secuencia de `businessInfos` y `tModelInfos` que a su vez son secuencias de `businessInfo` y `tModelInfo`.

La función `save_binding`, almacena y/o actualiza la secuencia de `bindingTemplate` pasada como parámetro, incluyendo su relación con los servicios correspondientes. Devuelve una estructura `bindingDetail` con la información registrada o actualizada.

La función `save_business`, almacena y/o actualiza la secuencia de `businessEntity` pasada como parámetro. Devuelve una estructura `businessDetail` con la información registrada o actualizada. Esta estructura es una secuencia de `businessEntity`. Como en el resto de funciones `save_xxx`, al procesar esta llamada, el registro UDDI posiblemente tendrá que generar claves, en el proceso de alta, si el publicador no las indica en la llamada. En las actualizaciones, las claves tienen que indicarse. Esta función puede mover servicios de un proveedor a otro. Se puede borrar información con esta función, cuando la información registrada sea diferente a la proporcionada en esta llamada, no se borra del nodo de custodia, pero la información que no se especifique del proveedor de servicios que ya exista en el registro será borrada, al actualizarse. Cuando un servicio contenido en un proveedor apunta, a través de su “clave ajena” a otro proveedor diferente, se anota en el registro como una proyección del servicio. Cuando se almacena un proveedor con servicios proyectados, se ignora todo su contenido, salvo las claves del servicio y del proveedor de servicios.

La función `save_service`, almacena y/o actualiza la secuencia de `businessService` pasada como parámetro. Devuelve una estructura `serviceDetail` (secuencia de `businessService`) con la información registrada o actualizada.

La función `save_tModel`, almacena y/o actualiza la secuencia de `tModel` pasada como parámetro. Devuelve una estructura `tModelDetail` (secuencia de `tModel`) con la información registrada o actualizada.

La función `set_publisherAssertions`, gestiona las relaciones de un publicador. Tiene como argumento una secuencia, posiblemente vacía, de `publisherAssertion` que sustituirá a todas las existentes controladas por el mismo publicador. Si no se especifica, se borrarán las existentes. Devuelve una estructura `publisherAssertions` con la secuencia de relaciones actuales del publicador.

C.2.3.4. API Security Policy

Esta API sólo incluye dos funciones: `get_authToken` y `discard_authToken`, que permiten obtener un token de autenticación e informar que el token de autenticación actual ya no es válido, respectivamente. La política de autorización del registro define si se implementa y cómo se implementa el mecanismo de control de acceso. El atributo opcional `authInfo` que aparece en todas las funciones descritas hasta el momento, contiene el token de autenticación, que tiene sentido sólo para el nodo que lo crea. Este elemento pretende permitir varios mecanismos de autenticación como usuario-contraseña, donde efectivamente `authInfo` contiene un token de autenticación generado por una operación de autenticación (como tickets Kerberos) o como afirmación de autenticación (como certificados X509). Cuando se usa este elemento simple de tipo cadena `authInfo` deben implementarse las dos funciones de esta API de la política de seguridad, aunque también pueden utilizarse otros mecanismos, como por ejemplo, la autenticación a nivel de transporte. El intento de utilizar un token caducado o descartado producirá un error.

La función `get_authToken`, obtiene un token de autenticación. No tiene ningún parámetro, sólo los atributos obligatorios `userID` y `cred`, ambos de tipo cadena. Representan el nombre de usuario y la contraseña o credencial asignados por el nodo a una persona. Devuelve síncronamente un `authToken`, que es una secuencia con un único elemento `authInfo`.

La función `dicard_authToken`, informa a un nodo que el token de autenticación pasado va a ser descartado, terminando la sesión que se inició con la creación de dicho token. Su único parámetro, obligatorio además, es una secuencia que contiene un único elemento `authInfo`. Devuelve un mensaje vacío si tiene éxito.

C.2.3.5. API Custody & Ownership Transfer

Cuando se publica información en un registro UDDI se establece una relación entre el publicador, la información publicada y el nodo del registro en que se publica. El propietario de la información es el publicador que la creó y tiene autoridad para cambiarla. El publicador es cualquier cliente que utiliza la API de publicación.

El propietario de una información puede transferir la propiedad a otro publicador del registro. Tiene que existir en el registro un nodo de custodia, que mantenga la relación de propiedad entre las entidades y sus publicadores a través de los mecanismos de autorización. Cada nodo del registro custodia la información que reside en él. Todos los nodos del registro tienen que garantizar la integridad de las entidades.

Esta API permite que los nodos de un registro puedan cooperar en la custodia de la información, transfiriendo tanto la custodia de la misma de un nodo a otro, como la propiedad de las estructuras de datos de un publicador a otro, pudiendo combinarse ambas transferencias. Por defecto, cada nodo custodia la información que se crea en el mismo.

Las transferencias de propiedad y custodia tienen sentido ante movimientos organizativos como fusiones, delegaciones, etc.

El espacio de nombres de la API es urn: uddi-org: custody_v3 (uddi_custody), y el esquema es uddi_v3custody.xsd.

Cuando se transfiere un businessEntity se transfieren con él sus estructuras dependientes, businessServices y bindingTemplates, e incluso sus relaciones, publisherAssertions.

La API está dividida en dos partes, el transferToken con tres de funciones de cliente (get_transferToken, discard_transferToken y transfer_entities) y una función entre nodos: transfer_custody.

El transferToken es una estructura que consiste en una secuencia formada por un identificador del nodo origen de la transferencia (nodeID); una fecha y hora de expiración del token (expirationTime) y una cadena que tiene sentido sólo para el nodo involucrado en la transferencia, codificada en base64 (opaqueToken). Representa la autorización para transferir la propiedad de información a otro publicador y la custodia de la misma a otro nodo del registro. La duración de la autorización la controla la política del nodo. El nodo que recibe el transferToken iniciará la función transfer_custody contra el nodo identificado en éste.

La operación entre nodos transfer_custody, se invoca desde el nodo destino de la transferencia de custodia en respuesta a la función transfer_entities. Asegura la autorización de la transferencia. Esta operación pertenece al espacio de nombres de replicación, urn: uddiorg: repl_v3 (uddi_repl), esquema uddi_v3replication.xsd, puesto que podría utilizarse para replicar información del registro.

C.2.3.6. API Subscription

Esta API proporciona la capacidad de recibir información acerca de los cambios realizados en un registro UDDI. La API permite establecer el ámbito de interés. Se trata de una API de implementación opcional. Los espacios de nombres de la APIs son urn_uddiorg: sub_v3 (uddi_sub) y urn_uddi-org: subr_v3 (uddi_subr) esquemas

uddi_v3subscription.xsd y uddi_v3subscriptionListener.xsd. La suscripción permite monitorizar cambios en cualquiera de las estructuras principales de UDDI: businessEntity, businessService, bindingTemplate, tModel, publisherAssertion o relatedbusinessEntity.

Normalmente se requiere autorización para suscribirse. Las políticas de cada nodo definen el uso de la suscripción. Se admiten tanto notificaciones síncronas, mediante peticiones a través de la función get_subscriptionResults, como notificaciones asíncronas, mediante llamadas a la función notify_subscriptionListener, que se implementa como un servicio “escuchador de suscripciones” por parte del cliente del registro UDDI.

Las notificaciones asíncronas se pueden realizar vía email o vía Web Service HTTP/SOAP que el suscriptor haya implementado.

Estos servicios se llaman “escuchadores de suscripciones”. Las notificaciones son periódicas, en vez realizadas en respuesta al continuo flujo de cambios que ocurren normalmente en el registro.

En peticiones síncronas se devuelve el estado actual de la parte del registro que cumple los criterios de la petición, los estados previos no están disponibles.

Las suscripciones son propiedad del suscriptor que las crea, la clave de la suscripción es visible sólo para su creador.

C.2.3.7. API Value Set

Cuando se salva una `keyedReference` o un `keyedReferenceGroup`, debe comprobarse su validez. El chequeo lo realizan `tModels` conforme a la política del registro. UDDI permite que terceras partes puedan registrar y chequear o validar conjuntos de valores (value sets).

Estos conjuntos de valores están formados por los sistemas de identificación, los sistemas de categorización, los tipos de relaciones y los espacios de nombres. Los conjuntos de valores pueden ser chequeables o no chequeables.

Las funciones de esta API permiten al registro UDDI la validación de los conjuntos de valores. La validación externa requiere la implementación de un Web Service que realice estas funciones. Se permite almacenar las validaciones de un conjunto de valores completo en una sola llamada para evitar múltiples llamadas de validación para cada valor del conjunto de valores, al considerarse información bastante estática, aunque evidentemente siempre existe riesgo de dar por válidos valores que ya no lo son. Esta API contiene dos funciones: `validate_values` y `get_allValidValues`. La política del registro determina qué conjuntos de valores se soportan y cómo. Las validaciones son síncronas.

La función `validate_values`, es invocada por el nodo para validar externamente los valores de los conjunto de valores de las entidades pasadas como argumentos. Mientras que la función `get_allValidValues`, es invocada por el nodo para validar externamente un conjunto de valores completo y almacenar dicha validación en el registro UDDI.

C.2.3.8. API Replication

Se definen 4 funciones: `get_changeRecords` y `notify_changedRecordsAvailable` realizan la replicación y la notifican, `do_ping` y `get_highWaterMarks` soportan otros aspectos de la replicación. El espacio de nombres de esta API es `urn:uddi-org:repl_v3 (uddi_repl)`, esquema `uddi_v3replication.xsd`. Esta API junto con `Custody & Ownership Transfer` permiten la afiliación de registros, al implementar la transferencia de información entre diferentes nodos y registros.

La función `get_changeRecords`, se usa para iniciar la replicación de los cambios de un nodo a otro, la ejecuta el nodo que desea recibir los cambios, proporcionando un vector alto de marca de agua (high water mark vector), que ayuda al nodo origen a determinar los cambios que debe enviar.

La función `notify_changeRecordsAvaible`, permite que un nodo informe a otros que tiene cambios disponibles para replicar, proporciona un modo proactivo de iniciar la replicación, normalmente precede a la invocación de la función `get_changeRecords`.

La función `do_ping`, no tiene ningún argumento, sólo una secuencia vacía, permite obtener la identificación de un nodo (`operatorNodeID`, de tipo `uddi: uddiKey`) que está preparado para replicar. Devuelve un elemento `operatorNodeID`.

La función `get_highWaterMarks`, proporciona una forma de obtener una secuencia de vectores altos de marcas de agua con los USN más altos conocidos por todos los nodos del grafo de replicación. No tiene ningún argumento, sólo una secuencia vacía. Devuelve una estructura `highWaterMarks`, que es una secuencia de `highWaterMark`, a su vez, este elemento es una secuencia de `changeRecordID_type`. Si no se conoce el USN, éste valdrá 0.

C.2.4. Extensibilidad de UDDI

La actual versión 3.0.2 del estándar UDDI tiene como mejora destacada en el modelo de información, la extensión del mismo [FeaturesUDDI3]. El mecanismo de extensión, recogido en el estándar en su apéndice H, [SpecUDDI3] es la derivación de los esquemas XML. En el estándar se proporcionan reglas y directrices sobre la extensión de la estructura de datos de UDDI. La derivación del esquema se realiza por extensión de las estructuras de datos, añadiendo nuevas estructuras a las mismas. A continuación se expone un ejemplo de extensión de una estructura de datos por derivación de XML esquema.

Este es el tipo base a extender:

```
<xs:complexType name="BaseType">
  <xs:sequence>
    <xs:element name="FirstName" type="xs:token"/>
    <xs:element name="LastName" type="xs:token"/>
    <xs:element name="Mail" type="xs:token" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Y este es el tipo extendido:

```
<xs:complexType name="ExtendedType">
  <xs:complexContent>
    <xs:extension base="BaseType">
      <xs:sequence>
        <xs:element name="Password" type="xs:token"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Puede observarse el uso del elemento *extension* de XML Schema para indicar el tipo base que se extiende. A continuación viene la extensión de la estructura de datos, en este caso un elemento `Password` a añadir al final de la secuencia del tipo base.

En UDDI, al extender cualquier estructura de datos, se exige que la extensión tenga su propio espacio de nombres. Se recomienda utilizar el atributo `substitutionGroup` en el nuevo elemento a definir para indicar en el propio elemento,

además de en su tipo, como ya hemos visto, el tipo base a extender. Además de extender el esquema de UDDI, se recomienda crear tModels para representar la extensión, creando un tModel para cada API que utilice la extensión, como mínimo serán las de publicación y consulta (inquiry). Se necesita una URL que referenciará cada tModel en el elemento overviewDoc, además de la URL propia del esquema (archivo xsd). En cada tModel se recomienda indicar la clave del tModel básico, (tModelKey). El keyValue correspondiente indicaría la API extendida. Este mecanismo permite a los clientes identificar las extensiones. La forma más sencilla de extender un elemento estándar es importarlo en el esquema extendido con el elemento import de XML Schema.

Los servidores UDDI que implementen extensiones, deben implementar tanto las APIs de UDDI estándar con sus estructuras de datos, como las extensiones. Existen dos posibilidades de implementación, o bien crear puntos de acceso diferentes de los estándar para las extensiones, indicándolo en los correspondientes bindingTemplates, o modificando la implementación de las funciones de la API de consulta get_xxx y save_xxx para que éstas distingan ambos servicios. Las extensiones también pueden firmarse digitalmente, bien dentro de la firma del propio elemento a extender o bien como una firma adicional exclusiva de la extensión realizada. También pueden excluirse de la firma digital del elemento a extender.

Hay que tener en cuenta las extensiones realizadas tanto en la operación de replicación del registro como en la promoción de una entidad que se haya extendido. En la replicación, todos los nodos deben soportar la extensión para que dicha replicación se realice correcta y completamente. En la promoción de una entidad extendida a un registro que no soporte dicha extensión puede realizarse la conversión del tipo extendido al básico, pudiendo afectar a las firmas digitales del mismo, si se incluyó la extensión en la firma digital del elemento extendido. Por esta razón es preferible firmar digitalmente la extensión al margen del elemento base, facilita la promoción de las entidades extendidas a registros que no soporten la extensión, si el registro destino soporta la extensión no habrá ningún problema.