
Departamento de Informática

Servicios Web

*Ignacio García, Macario Polo
Francisco Ruiz, Mario Piattini*

Universidad de Castilla-La Mancha, España.

Informe Técnico UCLM DIAB- 05 - 01 - 1
Enero 2005



Universidad de
Castilla-La Mancha

Este trabajo está parcialmente financiado por el proyecto MÁS (Ministerio de Ciencia y Tecnología/FEDER, TIC2003-02737-C02-02).

DEPÓSITO LEGAL:

ÍNDICE

ÍNDICE

1. INTRODUCCIÓN.....	3
1.1. Paradigma SOC (<i>Service-Oriented Computing</i>).....	3
1.2. Terminología básica.....	6
1.2.1 ¿Qué es un Servicio Web?	6
1.2.2. Elementos notables de los Servicios Web	6
2. ARQUITECTURA DE SERVICIOS WEB.....	13
2.1. Estilo arquitectónico.....	13
2.1.1. Arquitectura de interoperabilidad	13
2.1.2. Arquitectura orientada a servicios	14
2.1.3. Arquitecturas SOA y REST	16
2.2. Tecnologías de los SW	18
2.3. Modelos	21
3. ESTÁNDARES BÁSICOS.....	29
3.1. WSDL (<i>Web Service Description Language</i>)	29
3.1.1. Introducción.....	29
3.1.2. Definición del Servicio	32
3.1.2.1 Estructura del documento WSDL.....	32
3.1.2.2 Elemento <i>types</i>	33
3.1.2.3 Elemento <i>messages</i>	34
3.1.2.4 Elemento <i>portType</i>	36
3.1.2.5 Vinculaciones o <i>Bindings</i>	39
3.1.2.6 Puertos o <i>Ports</i>	39
3.1.2.7 Servicios o <i>service</i>	40
3.1.3. Vinculaciones SOAP o <i>SOAP Binding</i>	41
3.1.3.1 Elemento <i>soap:operation</i>	42
3.1.3.2 Elemento <i>soap:body</i>	42
3.1.3.3 Elemento <i>soap:fault</i>	44
3.1.3.4 Elementos <i>soap:header</i> y <i>soap:headerdefault</i>	45
3.1.3.5 Elementos <i>soap:adres</i>	46
3.2. SOAP (<i>Simple Object Access Protocol</i>).....	46
3.2.1. Introducción.....	46
3.2.2. Conceptos importantes.....	48
3.2.3. Modelo de Procesamiento SOAP.....	51
3.2.3.1. Nodos SOAP.....	51
3.2.3.2. Roles y Nodos SOAP	52

3.2.3.3. Apuntar a Bloques de Cabecera SOAP.....	53
3.2.3.4. Comprensión de los Bloques de Cabecera SOAP	53
3.2.3.5. Estructura e Interpretación de los Cuerpos SOAP	54
3.2.3.6. Procesamiento de los Mensajes SOAP.....	54
3.2.3.7. Retransmisión o reenvío de Bloques de Cabecera SOAP	56
3.2.3.8. Modelo de Versiones SOAP	58
3.2.4. Modelo de Extensibilidad de SOAP.....	59
3.2.4.1. Características SOAP	59
3.2.4.2. Patrones de Intercambio de Mensajes SOAP (MEPs)	60
3.2.4.3. Módulos SOAP	61
3.2.5. Marco de las Vinculaciones de Protocolo SOAP	62
3.2.5.1. Objetivos del Marco de Vinculación	63
3.2.5.2. Marco de Vinculación	63
3.2.6. Construcción de un Mensaje SOAP	64
3.2.6.1 Sobre SOAP (<i>SOAP Envelope</i>).....	64
3.2.6.2. Cabecera SOAP.....	65
3.2.6.3. Cuerpo SOAP	67
3.2.6.4. Fallo SOAP o SOAP <i>Fault</i>	68
3.2.7. Consideraciones sobre la Seguridad.....	70
3.2.7.1. Nodos SOAP.....	70
3.2.7.2. Intermediarios SOAP	71
3.2.7.3. Asociaciones del Protocolo Subyacente.....	71
3.2.8. Modelo de Datos SOAP	72
3.2.8.1. Aristas del Grafo	72
3.2.8.2. Nodos del Grafo	73
3.2.8.3. Valores	73
3.2.8.4. Ejemplo	73
3.2.9. Codificación SOAP.....	75
3.2.9.1. Equivalencia entre XML y el Modelo de Datos SOAP.....	75
3.2.9.2. Fallos en la Decodificación	77
3.2.10. Representación RPC de SOAP.....	78
3.2.10.1. Uso de RPC en la World Wide Web	79
3.2.10.2. RPC y el Cuerpo SOAP.....	80
3.2.10.3. RPC y las Cabeceras SOAP	81
3.2.10.4. Fallos RPC	82
3.3. UDDI (Universal Description, Discovery, and Integration)	82
3.3.1. Introducción.....	82
3.3.2. Especificaciones de UDDI.....	86
3.3.3. Programación UDDI	87
3.3.3.1. Estructuras de datos UDDI	88
3.3.3.2. Información Básica.....	90
3.3.3.3. Información Detallada.....	91
3.3.3.4. Publicar en un Registro UDDI.....	95
3.3.3.5. Seguridad y Autenticación	99

4. OTROS ESTÁNDARES.....	103
4.1. Orquestación y Coreografía.....	103
4.1.1. Introducción.....	103
4.1.2. Estándares relacionados.....	104
4.1.2.1. Lenguaje de Ejecución de Procesos de Negocio (BPEL)	104
4.1.2.2. Interfaz de Coreografía de Servicios Web (WSCI).....	106
4.1.2.3. Lenguaje de Gestión de Procesos de Negocio (BPML)	107
4.1.3. Colaboración Orquestada y Coreografiada	108
4.2. Coordinación y Transacciones.....	109
4.2.1. Composición de Servicios.....	113
4.2.2. WSCoordination y WS-Transaction.....	113
5. ENTORNOS TECNOLÓGICOS.....	119
5.1. Introducción.....	119
5.2. Microsoft .NET	120
5.2.1. Infraestructura de .NET para los Servicios Web.....	120
5.2.1.1. Espacios de nombres de Microsoft .NET	120
5.2.1.2. Proveedores de Servicios Web en .NET	120
5.2.2. Servicios Web en .NET	121
5.2.2.1. Publicación, Descubrimiento e Integración de SW en .NET	121
5.2.2.2. Publicación de un Servicio Web.....	123
5.2.2.3. Búsqueda de Servicios Web.....	125
5.2.2.4. Métodos como Servicios.....	126
5.2.2.5. <i>Proxy</i> para un Servicio Web.....	128
5.2.3. Seguridad en los Servicios Web.....	130
5.3. J2EE.....	133
5.3.1. JAXP	134
5.3.2. JAXB (<i>Java Architecture for Java Binding</i>).....	136
5.3.3. JAX-RPC (<i>Java API for XML-based Remote Procedure Call</i>).....	137
5.3.4. JAXM (<i>Java API for XML Messaging</i>)	138
5.3.4.1. Obtención de una conexión	139
5.3.4.2. Obtención de una conexión punto a punto.....	139
5.3.4.3. Obtención de una conexión al proveedor de mensajes	139
5.3.4.4. Creación de mensajes	140
5.3.4.5. Adición de adjuntos.....	140
5.3.4.6. Adición de contenido a la parte <i>Attachment</i>	141
5.3.4.7. Envío de mensajes.....	141
5.3.5. JAXR (<i>Java API for XML Registries</i>).....	141
5.3.6. Nociones avanzadas de JAXB	142
5.4. IBM WebSphere	144
5.4.1. ¿Qué es WebSphere?.....	144

5.4.2. WebSphere: Una familia de productos	145
5.4.3. SDK de WebSphere para el desarrollo de Servicios Web	146
5.4.3.1. Elementos del WSDK	147
5.5. CalculadoraWeb. Implementación de un Servicio Web Sencillo.....	149
5.5.1. Descripción genérica del servicio	149
5.5.2. Implementación en Microsoft.NET	152
5.5.3. Implementación en JAVA	159
DICCIONARIO	167
REFERENCIAS.....	173
ACRÓNIMOS	179

ÍNDICE
DE FIGURAS

ÍNDICE DE FIGURAS

FIGURA 1. SOA EXTENDIDA CON LAS CAPAS DE SERVICIO, LA FUNCIONALIDAD Y LOS ROLES [18]	4
FIGURA 2. ELEMENTOS DE LOS SW.	10
FIGURA 3. DIAGRAMA DE UNA SOA GENÉRICA.	16
FIGURA 4. TECNOLOGÍAS RELACIONADAS [28]	18
FIGURA 5. MODELOS ARQUITECTÓNICOS Y SUS RELACIONES [28]	21
FIGURA 6. RELACIONES ENTRE LOS CONCEPTOS RELACIONADOS CON LOS MENSAJES [28]	22
FIGURA 7. RELACIONES ENTRE LOS CONCEPTOS RELACIONADOS CON LOS SERVICIOS [28]	23
FIGURA 8. RELACIONES ENTRE LOS CONCEPTOS RELACIONADOS CON LOS RECURSOS [28]	24
FIGURA 9. RELACIONES ENTRE LOS CONCEPTOS RELACIONADOS CON LAS POLÍTICAS [28]	25
FIGURA 10. RELACIÓN ENTRE CONCEPTOS RELACIONADOS CON LA GESTIÓN Y ADMINISTRACIÓN DE SW [28]	26
FIGURA 11. CÓMO SE SUPERPONEN LOS ELEMENTOS DE WSDL PARA DESCRIBIR EL SW.	31
FIGURA 12. ATRIBUTO <i>IMPORT</i> .	33
FIGURA 13. ELEMENTO <i>TYPES</i> .	33
FIGURA 14. ELEMENTO <i>MESSAGE</i> .	34
FIGURA 15. ELEMENTO <i>PART</i> .	35
FIGURA 16. DESCRIPCIÓN DE UN MENSAJE DE CONTENIDO COMPLEJO.	35
FIGURA 17. ELEMENTO <i>PORTTYPE</i> .	36
FIGURA 18. OPERACIÓN <i>ONE-WAY</i> .	37
FIGURA 19. OPERACIÓN <i>REQUEST-RESPONSE</i> .	37
FIGURA 20. OPERACIÓN <i>SOLICIT-RESPONSE</i> .	38
FIGURA 21. OPERACIÓN <i>NOTIFICATION</i> .	38
FIGURA 22. GRAMÁTICA DE UNA VINCULACIÓN.	39
FIGURA 23. DEFINICIÓN DE UN PUERTO.	40
FIGURA 24. DEFINICIÓN DE UN SERVICIO.	40
FIGURA 25. ELEMENTO <i>SOAP:OPERATION</i> .	42
FIGURA 26. ELEMENTO <i>SOAP:BODY</i> .	43
FIGURA 27. ELEMENTO <i>SOAP:FAULT</i> .	45
FIGURA 28. ELEMENTOS <i>SOAP:HEADER</i> Y <i>SOAP:HEADERDEFAULT</i> .	45
FIGURA 29. ELEMENTO <i>SOAP:ADDRES</i> .	46
FIGURA 30. EJEMPLO DE MENSAJE SOAP DE ERROR.	70
FIGURA 31. FRAGMENTO DE UN MENSAJE SOAP	74
FIGURA 32. GRAFO DEL MODELO DE DATOS CORRESPONDIENTE AL FRAGMENTO DE LA FIGURA 31	74
FIGURA 33. RELACIÓN ENTRE EMPRESAS Y TECNOLOGÍA.	85
FIGURA 34. RELACIONES ENTRE LAS PRINCIPALES ESTRUCTURAS DE DATOS DE UDDI.	89
FIGURA 35. CONSULTA DE INFORMACIÓN DETALLADA SOBRE UN SERVICIO.	92

FIGURA 36. CATEGORIZACIÓN EN ISO 3166 DE UNA ZONA, DEVUELTO COMO PARTE DE UN <CATEGORYBA G>	94
FIGURA 37. PETICIÓN DE INFORMACIÓN DETALLADA DE UN TMODEL.	95
FIGURA 38. ESTRUCTURA DEL ELEMENTO <i>TMODEL</i> .	95
FIGURA 39. PETICIÓN DEL TOKEN DE AUTENTICACIÓN.	99
FIGURA 40. TOKEN DE AUTENTICACIÓN DEVUELTO A UNA PETICIÓN DE AUTENTICACIÓN.	99
FIGURA 41. RELACIÓN ENTRE ORQUESTACIÓN Y COREOGRAFÍA DE SW.	103
FIGURA 42. COLABORACIÓN AL ESTILO WSCI.	106
FIGURA 43. RELACIÓN ENTRE LOS ESTÁNDARES DE ORQUESTACIÓN Y COREOGRAFÍA.	108
FIGURA 44. PROTOCOLO DE CONFIRMACIÓN DE DOS FASES [15]	110
FIGURA 45. FALLO EN UNA TRANSACCIÓN “LÓGICA” DE LARGA DURACIÓN [15]	111
FIGURA 46. PILA TECNOLÓGICA DE MICROSOFT.NET	120
FIGURA 47. CLASE QUE SOPORTA LA INFORMACIÓN DE AUTENTICACIÓN.	123
FIGURA 48. SALVANDO UN TMODEL DE NUESTRO SW.	124
FIGURA 49. REGISTRO DE LA INFORMACIÓN PARA EL BUSINESSENTITY.	124
FIGURA 50. PUBLICACIÓN DE UN BUSINESSSERVICE.	124
FIGURA 51. CREANDO UNA PLANTILLA BUSINESSTEMPLATE PARA ASOCIAR EL TMODEL Y EL BUSINESSSERVICE.	125
FIGURA 52. BÚSQUEDA DEL NEGOCIO.	125
FIGURA 53. OBTENCIÓN INFORMACIÓN DE LOS SERVICIOS.	126
FIGURA 54. RECUPERAMOS LA INFORMACIÓN DE VINCULACIÓN PARA INVOCAR AL SW.	126
FIGURA 55. SINTAXIS DE WSDL.EXE	129
FIGURA 56. DOCUMENTO XML (IZQUIERDA) Y SU REPRESENTACIÓN ARBORESCENTE (DERECHA)	135
FIGURA 57. TRANSFORMACIÓN DE UN FICHERO EN XML DE ACUERDO A UNA HOJA DE ESTILO XSL	136
FIGURA 58. CÓDIGO NECESARIO PARA LA ADICIÓN DE CONTENIDO AL MENSAJE SOAP	140
FIGURA 59. CÓDIGO NECESARIO PARA LA ADICIÓN DE CONTENIDO AL MENSAJE SOAP	140
FIGURA 60. ADICIÓN DE UN ELEMENTO COMO <i>ATTACHMENT</i> A UN MENSAJE	141
FIGURA 61. ENVÍO DE UN MENSAJE	141
FIGURA 62. ENVÍO DE UN MENSAJE A TRAVÉS DE UN PROVEEDOR DE CONEXIÓN	141
FIGURA 63. ESQUEMA DE FUNCIONAMIENTO DEL JAXB	142
FIGURA 64. OBJETO JAXBCONTEXT	143
FIGURA 65. INTEGRACIÓN DEL SERVIDOR DE APLICACIONES	145
FIGURA 66. GRUPOS Y PRODUCTOS DE WEBSHERE [30]	146
FIGURA 67. DIAGRAMA DE CLASES DE <i>MICALCULADORAWEB</i>	150
FIGURA 68. CLASE <i>MICALCULADORAWEB</i>	150

FIGURA 69. CLASE <i>MIOPERANDOSENCILLO</i>	151
FIGURA 70. CLASE <i>MIOPERANDOCOMPLEJO</i>	151
FIGURA 71. LANZAR VISUAL STUDIO .NET	152
FIGURA 72. SELECCIÓN DEL LENGUAJE DE DESARROLLO.	152
FIGURA 73. SELECCIÓN DEL TIPO DE PROYECTO Y NOMBRE DEL MISMO.	153
FIGURA 74. ENTORNO DE DESARROLLO LISTO PARA COMENZAR.	154
FIGURA 75. IMPLEMENTAMOS <i>MIOPERANDOSENCILLO</i> .	154
FIGURA 76. IMPLEMENTAMOS <i>MIOPERANDOCOMPLEJO</i>	155
FIGURA 77. OPERACIONES QUE OFRECERÁ EL SW.	155
FIGURA 78. SW ACCEDIDO A TRAVÉS DEL NAVEGADOR.	156
FIGURA 79. PETICIÓN DE PERMISO PARA INVOCAR LA OPERACIÓN <i>GETOPERACIONESENCILLAS</i>	157
FIGURA 80. RESPUESTA CON LA LISTA DE OPERACIONES.	157
FIGURA 81. INTERFAZ DEL “ <i>INVOCADOR DE CALCULADORA WEB</i> ”	158
FIGURA 82. INVOCACIÓN DE UNA SUMA	158
FIGURA 83. INVOCACIÓN DE LA ECUACIÓN DE SEGUNDO GRADO	159
FIGURA 84. CREACIÓN DE UN ESPACIO DE TRABAJO	160
FIGURA 85. DIALOGO DE LA PLANTILLA DEL NUEVO PROYECTO	160
FIGURA 86. CLASES DEL SW “ <i>MICALCULADORA WEB</i> ”	161
FIGURA 87. <i>MIOPERANDOSENCILLO</i> CON SUS MÉTODOS DE ACCESO.	162
FIGURA 88. PRIMER PASO EN LA CONFIGURACIÓN DEL SW.	162
FIGURA 89. SEGUNDO PASO EN LA CONFIGURACIÓN DEL SW.	163
FIGURA 90. SW <i>MICALCULADORA WEB</i>	163

ÍNDICE
DE TABLAS

ÍNDICE DE TABLAS

TABLA 1. ESTÁNDARES BÁSICOS SOBRE SW	29
TABLA 2. ROLES RELEVANTES PARA LOS MENSAJES SOAP.....	52
TABLA 3. CÓDIGOS DE FALLOS SOAP.	69
TABLA 4. URLS DE PUNTOS DE ACCESO DE LOS PRINCIPALES NODOS OPERADORES.....	88
TABLA 5. DOCUMENTOS XML UTILIZADOS PARA LA RECUPERACIÓN DE INFORMACIÓN.	90
TABLA 6. DOCUMENTOS XML UTILIZADOS PARA LA RECUPERACIÓN DE INFORMACIÓN DETALLADA	91
TABLA 7. TAXONOMÍAS DE CATEGORIZACIÓN	93
TABLA 8. TIPOS DE IDENTIFICADORES SOPORTADOS.	94
TABLA 9. MENSAJES DE LA API DE PUBLICACIÓN UDDI.	98
TABLA 10. ESPACIOS DE NOMBRES DEL MICROSOFT UDDI SDK	123
TABLA 11. PROPIEDADES DEL ATRIBUTO <i>WEBMETHOD</i>	127
TABLA 12. PROPIEDADES DEL ATRIBUTO <i>WEBSERVICE</i>	128

1. INTRODUCCIÓN

1. INTRODUCCIÓN

En este apartado se pretende describir algunos conceptos básicos relacionados con la arquitectura orientada a servicios (SOA), la computación orientada a servicios o SOC y los Servicios Web (SW en adelante).

1.1. Paradigma SOC (*Service-Oriented Computing*)

El desarrollo de nuevos estilos arquitectónicos basados en construcciones tales como objetos y componentes, podría provocar que no se produjeran avances significativos en el campo del software. Para evitar esto, los desarrolladores tendrían que adquirir otro punto de vista, viendo cómo siguiendo otros paradigmas, el software puede entregar su funcionalidad a los usuarios de una manera distinta a la habitual.

Esta nueva concepción entiende que el software debe entregarse en forma de servicio. El concepto de software como servicio, es una visión de futuro que nos permite ofrecer soporte a las necesidades de un mercado del software muy competitivo en el que los negocios recopilarán o proveerán servicios según los requisitos que necesiten cubrir. Este nuevo paradigma pretende separar los conceptos de *posesión y propiedad* del concepto de *uso* [24].

Este paradigma recibe el nombre de computación orientada a servicios (SOC). La SOC abre nuevos mercados en el mundo del software, tanto para los proveedores de servicios específicos a pequeña escala como para las grandes organizaciones que provean servicios de carácter más general. Además el aprovisionamiento de servicios puede incluir la creación dinámica y el desarrollo de nuevos servicios a partir de los existentes mediante técnicas de composición de servicios.

En parte, esta tendencia es posible gracias al soporte ofrecido por los estándares y la presencia dominante de las tecnologías de la red. Los servicios deben definir de manera declarativa sus requisitos funcionales y no funcionales, además de sus capacidades mediante un formato inteligible por las máquinas y previamente convenido. Mediante estas descripciones, se puede llevar a cabo el descubrimiento de manera automática de servicios, la selección y la vinculación.

La SOC toma los servicios como elementos fundamentales para el desarrollo de aplicaciones. Los principios en los que se basa la SOC se reflejan en las capas que establece la arquitectura orientada a servicios (SOA) (Figura 1). Por un lado tenemos los servicios básicos, sus descripciones y las operaciones básicas (publicación, descubrimiento, selección y vinculación) que producen o utilizan dichas descripciones. Las capas más altas de la pirámide proporcionan un soporte adicional requerido para la composición y gestión de servicios [18]. Los servicios se convierten en los bloques básicos para la construcción de nuevas aplicaciones. De esto último se desprende que la composición es realmente uno de los conceptos principales de la SOC, ya que se convierte en el núcleo del proceso de desarrollo de aplicaciones. La composición de servicios combina servicios siguiendo patrones de composición para alcanzar los objetivos del negocio, resolver algún problema concreto o simplemente proveer determinadas funcionalidades. La operación de composición puede dar lugar a nuevos servicios, por lo que la composición es una operación recursiva.

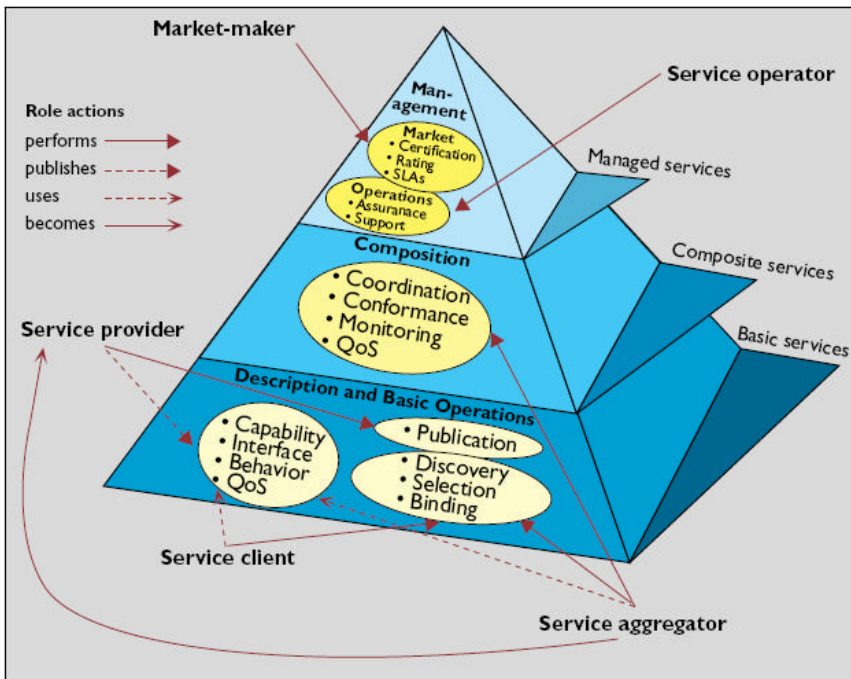


Figura 1. SOA Extendida con las capas de servicio, la funcionalidad y los roles [18]

La composición de servicios abarca los roles y la funcionalidad que nos permite generar un servicio mediante la unión de otros. De manera recursiva, estos servicios compuestos pueden ser usados por *agregadores de servicios* a modo de componentes o servicios básicos para realizar otras composiciones a mayor escala o, utilizarlos directamente como soluciones para los clientes. De esta forma, los *agregadores* de servicios se convierten en proveedores de servicios mediante la publicación de la descripción de los servicios compuestos que crean. Es responsabilidad de los *agregadores* el implementar la lógica necesaria (especificaciones o código) para que una composición realice funciones tal como pueden ser la coordinación, la monitorización, la conformidad y la calidad del servicio (QoS). En la capa superior de la Figura 1, se sitúa la capa de gestión de servicios [18].

La aplicación de la SOC a la Web se manifiesta en los SW. Un SW, es un tipo especial de servicio que se identifica mediante una URI, que utiliza estándares de Internet para definir su descripción y método de transporte. Las interacciones entre los SW ocurren mediante mensajes SOAP, que transportan los datos mediante XML [10]. Las descripciones de las interfaces de los SW son expresadas mediante el lenguaje de definición de Servicios Web (WSDL). El estándar UDDI (*Universal Description, Discovery and Integration*) define un protocolo para servicios de directorio que contienen descripciones de SW. UDDI permite a los clientes de los SW localizar candidatos, según el cumplimiento o no de una serie de criterios. Los clientes y los proveedores de servicios utilizan estos estándares para llevar a cabo las operaciones básicas de la SOA. Los *agregadores de servicios* pueden utilizar el lenguaje de ejecución para procesos de negocios para SW (BPEL4WS) para crear nuevos SW mediante la definición de la correspondiente composición de las interfaces y los procesos internos de los servicios existentes.

Cualquier arquitectura que implemente la filosofía SOC, deberá proveer definiciones basadas en estándares sobre el protocolo de comunicación que utilice para implementar la interoperabilidad, mecanismos para la descripción de los servicios, para el descubrimiento y la composición, así como un conjunto básico de protocolos para la calidad de servicio (QoS) [5].

1.2. Terminología básica

1.2.1 ¿Qué es un Servicio Web?

El término Servicio Web es un concepto del que actualmente se abusa con cierta frecuencia y más aun en la actualidad ya que al tratarse de una tecnología relativamente reciente, no se tiene una idea clara y concisa acerca de los requisitos que debe cumplir un sistema software para que sea realmente un SW. Son muchas y muy variadas las definiciones de SW que actualmente se pueden leer en las distintas publicaciones que abordan este tema. A continuación veremos algunas de ellas.

Según [1] una definición precisa de SW sería la emitida por la W3C: “una aplicación software identificada por una URI, cuyas interfaces y vinculaciones son capaces de ser definidas, descritas y descubiertas como artefactos XML. Un SW soporta la interacción con otros agentes software mediante el intercambio de mensajes basado en XML a través de protocolos basados en Internet”.

Por otro lado, según [17], los SW son interfaces Web genéricas a servicios componente. A fin de soportar la interoperabilidad entre todas las arquitecturas, los SW utilizan protocolos del W3C como pueden ser XML, WSDL y SOAP.

Los autores de [3], definen los SW como aplicaciones basadas en la Web compuestas por funcionalidades de granularidad gruesa que son accesibles a través de Internet. En [3] se continúa comentando que, desde una perspectiva técnica, los SW son una forma estandarizada de integrar aplicaciones basadas en la Web mediante estándares abiertos, incluyendo XML, SOAP, WSDL, y UDDI.

1.2.2. Elementos notables de los Servicios Web

Los SW, constituyen un avance tecnológico que no está ligado a ninguna tecnología, ya que una de sus premisas es la interoperabilidad multiplataforma. Un SW es más una idea, un concepto cuya implementación debe respetar una serie de reglas e interfaces para poder ser accedido por cualquier sistema conectado a la red, siempre que disponga de permiso para ello.

Fuera de toda implementación, como hemos dicho, existen una serie de partes bien diferenciadas que tenemos que tener en cuenta a la hora de implementar y utilizar un SW. Estas son los agentes y los servicios, el cliente y el proveedor, la descripción del servicio, la semántica, y por último las personas, la semántica y los agentes.

Agentes y Servicios

Un SW, constituye una idea o concepto que debe ser implementado por algún agente. El agente es un trozo o pieza de software que implementa esa funcionalidad que realiza el SW. Este agente tiene la peculiaridad de estar capacitado para enviar y recibir mensajes. De esto se desprende, que el agente es el encargado de recibir las peticiones y enviar las respuestas. Sin embargo, el SW es el conjunto abstracto de funcionalidades ofrecidas.

El agente será escrito usando algún lenguaje de programación y se ejecutará bajo una plataforma en concreto. Podemos variar este agente según la implementación que deseemos o necesitemos darle. Sin embargo, el SW que implemente el agente será siempre el mismo, es decir, sea cual sea la implementación que demos a un agente para un SW determinado, este último será semánticamente siempre el mismo, lo que variará será el agente.

El cliente y el proveedor

El SW pertenece a un propietario que puede ser bien una persona, bien una organización. La *entidad proveedora* será la persona u organización que desarrolle un agente capaz de soportar un determinado servicio.

La *entidad que realiza la consulta* puede ser también una persona u organización que desea hacer uso del servicio que expone la entidad proveedora. Esta comunicación tipo *petición-respuesta* se realiza entre agentes, ya que en el lado del que consulta también hay implementado un agente que se comunica con el agente de la entidad proveedora, que es el que implementa el servicio o el conjunto de servicios. Para que la comunicación se lleve a cabo de manera correcta, las entidades participantes deben ponerse de acuerdo, tanto en la semántica como en los mecanismos utilizados en el intercambio de mensajes.

Descripción del Servicio

El intercambio de mensajes entre las entidades está guiado por un protocolo que éstas deben seguir para que las transacciones lleguen a buen fin. Este protocolo está documentado en la descripción del SW (WSD, Web Service Description). La WSD es una descripción completa del SW, escrito en un lenguaje denominado WSDL [29], y que es inteligible tanto por las máquinas como por las personas, aunque no sin cierta dificultad para estos últimos.

En esta descripción se incluyen todos los elementos de un SW susceptibles de ser descritos, tal y como son el formato de los mensajes, los tipos de datos, los protocolos de transporte a través de los cuales el servicio está disponible y los distintos formatos de serialización utilizados para enviar el contenido de los mensajes entre los agentes que llevan a cabo esta comunicación.

En esta descripción, también se pueden incluir una o varias direcciones o localizaciones de red o *endpoints*, mediante las cuales podremos invocar al agente proveedor o podemos obtener información acerca del patrón de comunicación que hay seguir.

Semánticas

Definiremos *semántica* de un SW como el comportamiento que se espera que tenga el mismo. Este comportamiento, se refiere a qué esperamos del SW cuando le enviamos un mensaje de petición.

De la misma forma que la sintaxis de un objeto es su estructura, la semántica de un objeto se refleja en las relaciones que éste establece con otros objetos. Mientras que en la sintaxis identificamos diferentes partes, en la semántica identificamos distintos *contextos*.

Esta semántica también puede verse como un contrato entre la entidad que realiza la consulta y la entidad a la que va dirigida la consulta. Este contrato supone un total acuerdo entre las dos entidades, y trata sobre cómo y porqué los agentes que intervienen en la comunicación deberán interactuar. Este acuerdo puede ser explícito o implícito, oral o escrito, inteligible por la máquina o por las personas.

Personas, agentes y semánticas

Uno de los propósitos principales de los SW, es el de automatizar procesos que de lo contrario deberían ser llevados a cabo manualmente. El papel que tienen las personas en el uso y en la arquitectura de los SW se puede ver en dos aspectos:

1. Las personas deben estar de acuerdo en la semántica y en la descripción del servicio. Los usuarios de los SW deberán estar implícita o explícitamente de acuerdo con la semántica y la descripción del servicio al cual dirigen las peticiones, y con el que realizan la interacción, mientras que este servicio pertenezca a una persona u organización. La entidad proveedora publicará la semántica y la descripción de su servicio como un contrato del tipo “*¿ lo tomas o lo dejas*”, que tendrá que ser aceptado por la entidad que contrate el servicio.

2. Son las personas las que crean los agentes que intervienen en la comunicación, el agente proveedor y el agente que realiza la petición. Los creadores de los agentes deben asegurar que estos cumplen la semántica y la descripción del servicio. Hay varias formas de asegurar esto último:
 - Un agente puede ser codificado, de manera que permanentemente implemente la semántica y la descripción de un servicio.
 - Podemos codificar el agente, de una manera más general, de forma que dinámicamente le pasemos la descripción y/o la semántica del servicio deseado en ese momento.
 - Podemos crear en primer lugar el agente, y luego generar la descripción y/o la semántica del servicio después, a partir del código del agente.

Resumen

En la Figura 2, resumimos y situamos todos los elementos citados.

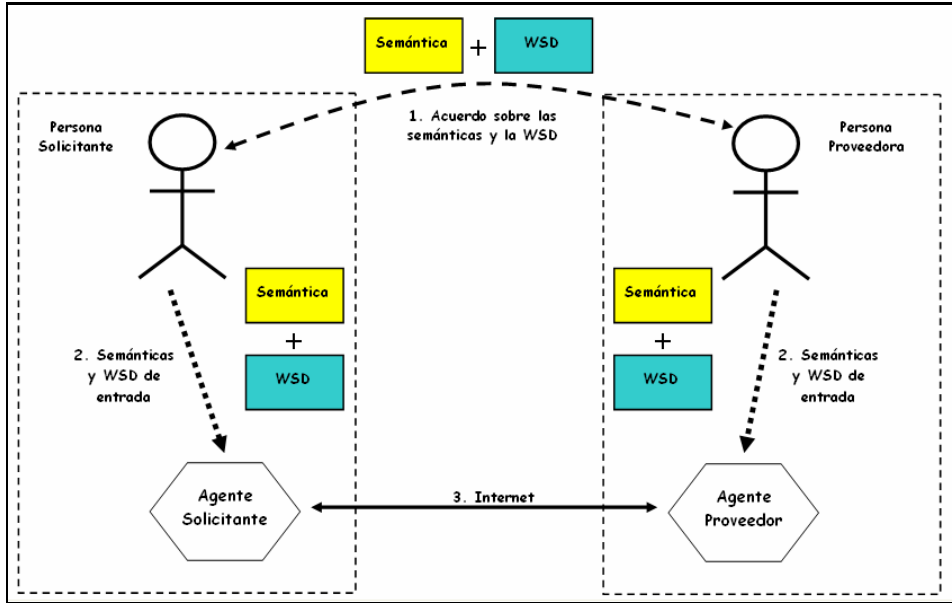


Figura 2. Elementos de los SW.

Como se puede ver, la entidad proveedora y la que realiza la petición, se ponen de acuerdo respecto a la descripción del servicio (mediante un documento en WSDL) y la semántica que guiarán la interacción entre los agentes. Cada uno de los agentes, implementa la semántica del servicio, pero desde el punto de vista que le corresponde, es decir, desde el punto de vista del proveedor o del consumidor (el que realiza la petición). Ambos agentes (el solicitante y el proveedor) intercambian mensajes SOAP en nombre de los respectivos propietarios.

2. ARQUITECTURA DE LOS SERVICIOS WEB

2. ARQUITECTURA DE SERVICIOS WEB

Para el desarrollo de esta sección nos hemos basado entre otros, en la *Web Service Architecture* [28], publicada por la W3C.

2.1. Estilo arquitectónico

El concepto de *Arquitectura de SW* [31] es un concepto muy amplio que podríamos desglosar en otras arquitecturas a su vez:

- Arquitectura de interoperabilidad.
- Arquitectura orientada a servicios.
- Arquitecturas SOA y REST.

A continuación comentaremos las distintas arquitecturas y las bases sobre las que se asientan.

2.1.1. Arquitectura de interoperabilidad

La arquitectura de SW, es también una arquitectura de interoperabilidad que identifica aquellos elementos de la *red global de servicios Web* necesarios para asegurar la interoperabilidad entre los SW. Los SW no son entidades destinadas a existir y operar separadas, por lo que hay que definir una arquitectura que soporte la interoperabilidad entre los mismos. Por otro lado, hay que tener en cuenta que los servicios tendrán una implementación privada que dependerá de los desarrolladores. Para asegurar esta interoperabilidad, la arquitectura identifica y define conceptos importantes como son las restricciones y las relaciones.

Esta arquitectura persigue alcanzar una serie de objetivos:

- Interoperabilidad entre SW.
- Integración con la *World Wide Web*.
- Fiabilidad de los SW.

- Seguridad en los SW.
- Escalabilidad y extensibilidad de los SW.
- Administración de los SW.

Esta arquitectura está pensada para mostrar cómo cooperando con otras tecnologías relacionadas, se obtienen muchos beneficios para los SW. La arquitectura de interoperabilidad nos da una perspectiva global de la *arquitectura de servicios en red*.

2.1.2. Arquitectura orientada a servicios

La arquitectura de la Web y la Arquitectura de SW que venimos desarrollando, son simplemente instancias de la *Arquitectura Orientada a Servicios* (SOA), que es un tipo de arquitectura de sistemas distribuidos caracterizada por tres propiedades:

1. *Orientación a la conversación*. El foco de atención recae sobre los mensajes intercambiados entre los nodos, y no sobre los nodos en sí.
2. *Abstracción del agente*. En SOA, un nodo es una entidad computacional que participa en conversaciones con otros nodos. En nuestro caso, estas entidades pertenecen a un humano o a una organización, de ahí la abstracción del agente.

En SOA, hay una serie de conceptos como la estructura interna de un agente, incluyendo características como pueden ser el lenguaje de implementación, la estructura de proceso y la estructura de la base de datos, que son abstraídos adrede, ya que no es necesario conocer este tipo de particularidades para que la comunicación se lleve a cabo.

Un hecho relevante que tiene importantes consecuencias, es obviar la implementación de los agentes, ya que de esta forma podríamos incluir sistemas heredados con un coste mínimo, ya que estos se agregarían como un agente en la red de servicios.

Esto nos evitaría serios problemas arquitectónicos derivados de la necesidad de conocer determinados sistemas a nivel estructural.

3. *Metadatos*. En una carrera por alcanzar la escalabilidad global, las Arquitecturas Orientadas a Servicios se asocian con metadatos. Estos metadatos son descripciones acerca de la forma y tipo de los elementos que transportan los mensajes, el orden de los mensajes, el significado de los mensajes, etc.

Estas amplias descripciones, permiten validar los mensajes de una forma sencilla. Usar metadatos es algo similar a utilizar estándares abiertos, ya que permite a terceras partes beneficiarse de los recursos empleados en la Red de Servicios, sin necesidad de requerir ninguna autorización para validar las peticiones.

Para ver cómo se relacionan entre sí y con tecnologías como CORBA, podemos decir que SOA es un tipo de *sistema distribuido*. Un sistema distribuido consiste en un conjunto de agentes software que deben colaborar juntos para implementar una funcionalidad deseada. Puesto que estos agentes no suelen trabajar en el mismo entorno, necesitarán algún método para comunicarse. Esta comunicación se realizará a través de pilas de protocolos hardware/software, que serán intrínsecamente menos fiables que la invocación directa de código y la memoria compartida. Esto tiene consecuencias arquitectónicas importantes, como el hecho de que los sistemas distribuidos requieran de los desarrolladores, la precaución de tener en cuenta que la latencia de los accesos remotos, puede llegar a ser impredecible. Además también existe otro problema, el de la concurrencia y los posibles fallos parciales que podrían producirse en el sistema [14].

Una SOA es un tipo específico de sistema distribuido en el que los agentes son *servicios*, donde un *servicio* es un agente software que lleva a cabo una operación bien definida (esto es, que “provee un servicio”) y que puede ser invocado desde afuera del contexto de una gran aplicación. Mientras que un servicio puede ser implementado para exponer una característica de una aplicación, los usuarios de ese servidor sólo deben preocuparse de la definición de la interfaz del servicio. Además, muchas de las definiciones de SOA, acentúan el hecho de que los servicios tienen una interfaz de red direccionable, y que se comunican mediante protocolos y formatos de datos estándares.

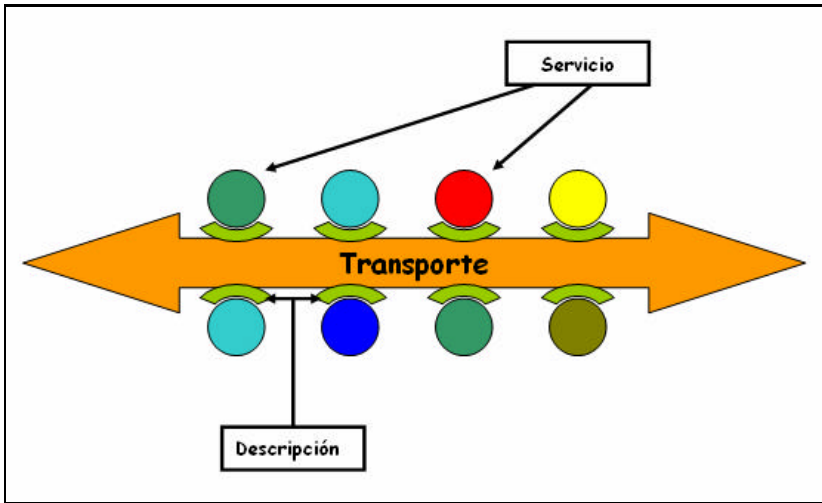


Figura 3. Diagrama de una SOA genérica.

Los elementos clave de una Arquitectura Orientada a Servicios son los mensajes intercambiados. Los agentes actúan solicitando los servicios o haciendo de proveedores de los mismos, compartiendo los mecanismos de transporte que permiten el flujo de los mensajes. En el caso de que la SOA sea pública, incluiremos la descripción pública de esos elementos, descripciones de los mensajes, etc.

2.1.3. Arquitecturas SOA y REST

La World Wide Web es una SOA, que opera como un sistema de información en red con algunas restricciones adicionales. Algunas de estas restricciones pueden ser que los agentes identifican recursos en la red llamados *resources*, mediante el identificador único de recurso o URI. Un agente representa, describe y comunica el estado del recurso mediante *representaciones* del recurso mediante una serie de formatos de datos generalizados como pueden ser XML, HTML, CSS, JPEG, etc. Los agentes intercambian esas representaciones mediante protocolos que usan URIs para identificar y direccionar directa o indirectamente a los agentes y los recursos [11]

Hay un modelo arquitectónico más restringido para aplicaciones Web más fiables conocido como “Transferencia de Representación de Estado” (*Representation State Transfer* o *REST* [19],[21]), propuesto por Roy Fielding. La Web REST constituye un subconjunto de la WWW, cuyos

agentes son restringidos, exponiendo y mostrando servicios mediante semánticas de interfaz uniforme y manipulando recursos sólo mediante el intercambio de *representaciones*. Esta arquitectura utiliza a la hipermmedia como motor del estado de la aplicación.

Podemos identificar dos clases principales de SW:

- Servicios conformes a REST o servicios de “manipulación directa del recurso”, en los que, el principal propósito es manipular representaciones en XML del recurso utilizando un conjunto mínimo y uniforme de operaciones.
- Servicios de “Objeto distribuido” o servicios de “operaciones a través de la Web”, en los que el principal propósito es realizar un conjunto complejo de operaciones sobre los recursos que no deben “estar en la Web”, los mensajes en XML contienen la información necesaria para invocar esas operaciones.

Estas clases de “Servicios Web” usan URIs para identificar los recursos, los protocolos Web y los datos, y utilizan el formato XML para los mensajes. Donde difieren básicamente, es que el “objeto distribuido” utiliza vocabularios específicos de aplicación, como el motor del estado de la aplicación en lugar de la hipermmedia. Se hace más hincapié sobre los mensajes, en lugar de centrarse en las acciones causadas por estos, significa que SOA tiene una buena *visibilidad*, esto es, terceras partes pueden inspeccionar el flujo de mensajes a fin de asegurarse tanto de los servicios invocados, como de los roles jugados por las distintas partes. Ejemplo de esto pueden ser algunos programas como los cortafuegos (*firewall*), cuyas tareas son controlar del tráfico de mensajes, controlar la estructura de los mensajes y tomar de decisiones para la seguridad.

En las SOA conforme a REST, la visibilidad se debe a las semánticas de interfaz uniforme. En concreto, se deben al protocolo HTTP, ya que un intermediario puede inspeccionar la URI del recurso que está siendo manipulado, la dirección TCP/IP y la interfaz de operación de aquel que realiza la petición, determinando de esta forma cuando la operación del solicitante debe ser llevada a cabo. Si los servicios no son REST pero sí están basados en XML, esa visibilidad se debe a que XML es el metaformato universal para los datos. Los intermediarios, pueden ser programados o configurados para usar el lenguaje XML para los

mensajes SOAP, las cabeceras estandarizadas de SOAP o reconocer expresiones genéricas en XPath [33], a fin de poder hacer enrutamientos y filtros.

2.2. Tecnologías de los SW

Entorno a la arquitectura de los SW, podemos encontrar una gran cantidad de tecnologías relacionadas. De la misma forma que existen muchas formas de visualizar estas tecnologías, también hay multitud de maneras de construir y usar nuestros SW.

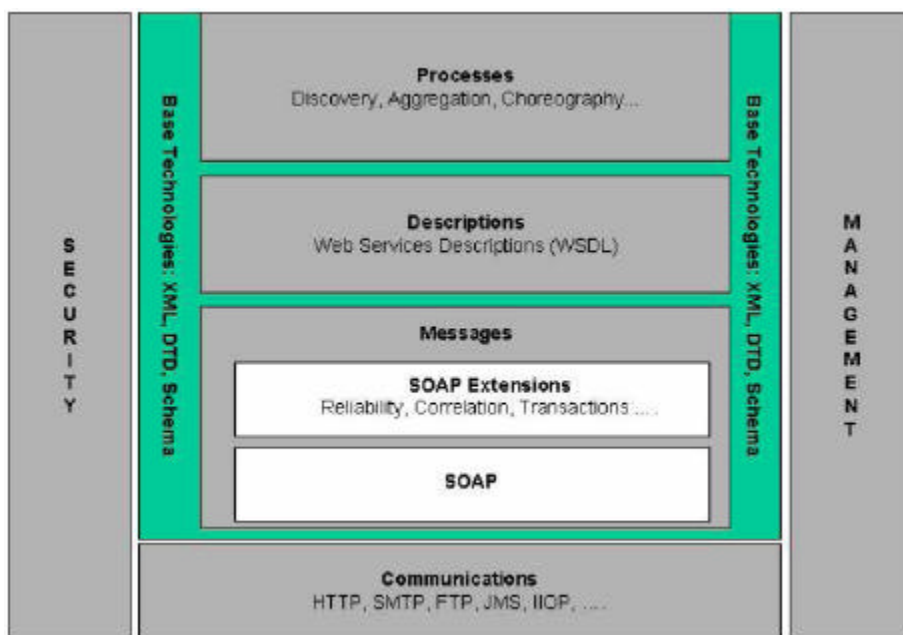


Figura 4. Tecnologías relacionadas [28]

En la Figura 4, podemos ver algunas tecnologías relacionadas. Una de ellas es WSDL, que describe, entre otras cosas, el formato de los mensajes SOAP, y UDDI, que hace las veces de servicio de descubrimiento, publicación e integración de SW.

Gracias a la disposición de las tecnologías en la Figura 4, podemos imaginar qué papel juegan dentro de los SW y a qué nivel. Sin ir más lejos, mientras que WSDL y SOAP son necesarios en distintos momentos o niveles dentro de los SW, la seguridad es una premisa que se debe cumplir siempre, o cuanto menos, es más que recomendable. Los

protocolos para la comunicación, como se pueden observar, están en la base de los SW, justo debajo de SOAP, ya que en formato SOAP serán los mensajes que enviaremos mediante estos protocolos.

Como se puede observar en la Figura 4, XML está presente en todos los niveles, y es necesario para implementar o dar soporte a muchos de los estándares que intervienen en los SW. Si hemos dicho que los SW tienen la propiedad de ser multiplataforma [21], es debido, además de a su organización arquitectónica, a que utiliza XML que es uno de los estándares de la red más importantes, ya que al no tener un formato propietario y reducirse simplemente a cadenas de caracteres, constituye el medio de comunicación multiplataforma por excelencia, aportándonos esa neutralidad tan deseable en la red. Así que podemos asegurar que XML supone una de las bases de la Arquitectura de SW. Otra propiedad importante de XML es que al ser un meta-lenguaje, nos permite distinguir dentro de los mensajes entre los datos útiles y los datos referentes al protocolo que estemos utilizando. Esto último, es útil, entre otras cosas, para agilizar y hacer más sencillas algunas de las tareas de procesamiento. XML, como tecnología, se apoya sobre otros estándares y especificaciones, tales y como pueden ser la especificación de XML (sus diferentes versiones), *XML Schema Description Language* [34], *XML DTD* [6], *XPath* [33], etc. Aunque la base tecnológica de la Arquitectura de SW sólo contempla las especificaciones de XML, *XML Schema Description Language* y la especificación Base de XML.

Los clientes se comunican con los SW mediante mensajes, y éstos, mandan las respuestas a las peticiones de la misma manera. Todos los mensajes, ya sean de invocación a un SW o de respuesta, necesitan algún método para viajar, alguna forma de ser transportados. La Arquitectura de SW abarca una gran cantidad de mecanismos de comunicación, entre los que podemos encontrar el protocolo de transporte HTTP, el protocolo SMTP, FTP, APIs genéricas (como puede ser JMS [12]), protocolos de distribución de objetos (como IIOP [4]), etc. En este sentido, la Arquitectura de SW, es muy versátil, ya que permite que los mensajes sean enviados de múltiples formas. Posiblemente, el único requisito para que un mensaje sea enviado a un SW o remitido desde uno de ellos, es que el medio de transporte permita enviar caracteres. Un mensaje SOAP puede ser enviado desde cualquier nivel del modelo propuesto por OSI [23]. Incluso protocolos que fueron diseñados con fines distintos pueden

encapsular los mensajes de los SW, como ocurre con el SMTP, que fue diseñado para correo electrónico.

Alrededor del envío de los mensajes se despiertan múltiples preguntas e incertidumbres. Algunas de estas cuestiones son la seguridad, la fiabilidad, la existencia de aplicaciones que trabajan con distintas redes, con distintas entidades, la existencia de aplicaciones que están fragmentadas en partes, etc. Todas estas cuestiones se solucionan si disponemos de un mecanismo que gestione las comunicaciones, una tecnología que se encargue de los mensajes. Por esto, en la Arquitectura de SW, existe una tecnología para gestionar todo lo referente a los mensajes, el estándar SOAP. SOAP permite la autenticación, cifrado, control de acceso, procesamiento de las transacciones, enrutamiento de los mensajes, confirmación de entrega, etc. Todo esto es posible gracias a la estructura del mensaje SOAP, que consta de un “sobre” donde se ubica el mensaje propiamente dicho, y una cabecera o modelo de procesamiento. Mediante esta estructura podemos guardar toda la información necesaria para realizar todas esas tareas necesarias para la gestión de los mensajes que indicábamos anteriormente.

Hoy en día vivimos en un mundo heterogéneo, y más aun dentro del campo de la tecnología. Existen multitud de sistemas heterogéneos, y la interoperabilidad entre ellos es una propiedad muy deseable. Por ello, se buscan mecanismos que permitan que la estructura y los tipos de datos de los mensajes sean inteligibles, tanto para los SW como para aquellos que realizan las peticiones. Es muy importante que los que necesitan hacer uso de un SW no tengan que preocuparse de la plataforma bajo la que éste está funcionando. De la misma forma, tampoco interesa a los productores de SW que sus clientes potenciales sean sólo usuarios de la misma plataforma que la utilizada para desarrollar los servicios.

Para remediar los problemas inherentes a la heterogeneidad de los sistemas, se adoptan protocolos y estándares de la red que eliminan este tipo de barreras. SOAP establece una base y unas reglas para crear los mensajes que viajaran entre los agentes que implementan los servicios. El utilizar el formato de mensaje que SOAP ofrece, entre otros beneficios, la garantía al emisor de que el receptor entenderá el contenido de su petición, y al receptor que comprenderá lo que se le pide, siempre y cuando el cliente haya seguido el formato del mensaje SOAP. Además de todo esto, los mensajes SOAP se escriben mediante el meta-lenguaje

XML, que al no tener formato propietario y ser simplemente caracteres, es el candidato idóneo para la comunicación multiplataforma.

Además de la descripción de los mensajes, en la Arquitectura de SW, se contemplan los procesos de descubrimiento de descripciones de servicios mediante algún criterio dado, la descripción de secuencias de mensajes con estado entre múltiples partes, agregaciones de procesos dentro procesos de más alto nivel, etc.

2.3. Modelos

¿Qué es un modelo?

Un modelo es un subconjunto coherente de la arquitectura que se centra básicamente en un aspecto concreto dentro de la arquitectura. Cada modelo intenta explicar al máximo el aspecto en el que se ha centrado. Un modelo también debe definir todas las dependencias que pueda presentar con respecto a otros aspectos incluidos en la arquitectura dentro de la cual se ubica. En la Figura 5 mostramos las relaciones existentes entre los distintos modelos mediante una representación gráfica de los mismos.

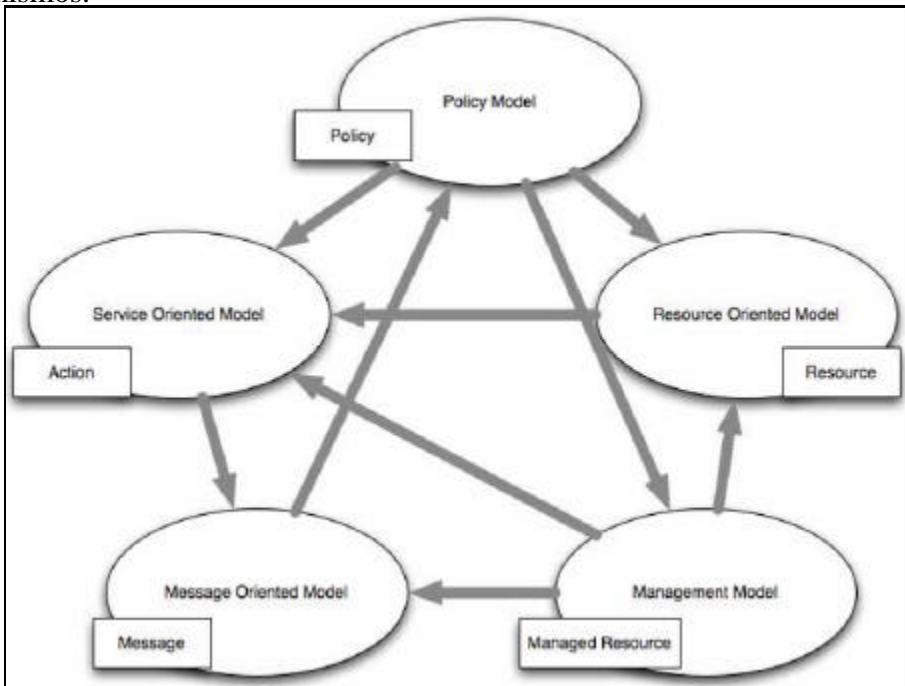


Figura 5. Modelos arquitectónicos y sus relaciones [28]

Los Modelos Arquitectónicos

La Arquitectura de SW posee cinco modelos:

- *El Modelo Orientado a Mensajes*, que se centra en los mensajes, su estructura, el método de transporte, etc. Este modelo no atiende ni al porqué de los mensajes ni a su significado. El interés principal de este modelo se sitúa en la estructura de los mensajes, las relaciones existentes entre los que envían los mensajes, los que los reciben y otros intermediarios que tengan que procesar también los mensajes para reenviarlos. Relacionados con los mensajes, hay una serie de conceptos que resultan interesantes dentro de la arquitectura y que este modelo comenta, mostrando cual es la relación que guardan unos conceptos con otros, aunque la profundidad con que debe ser tratado se escapa a la profundidad con la que trataremos la cuestión.

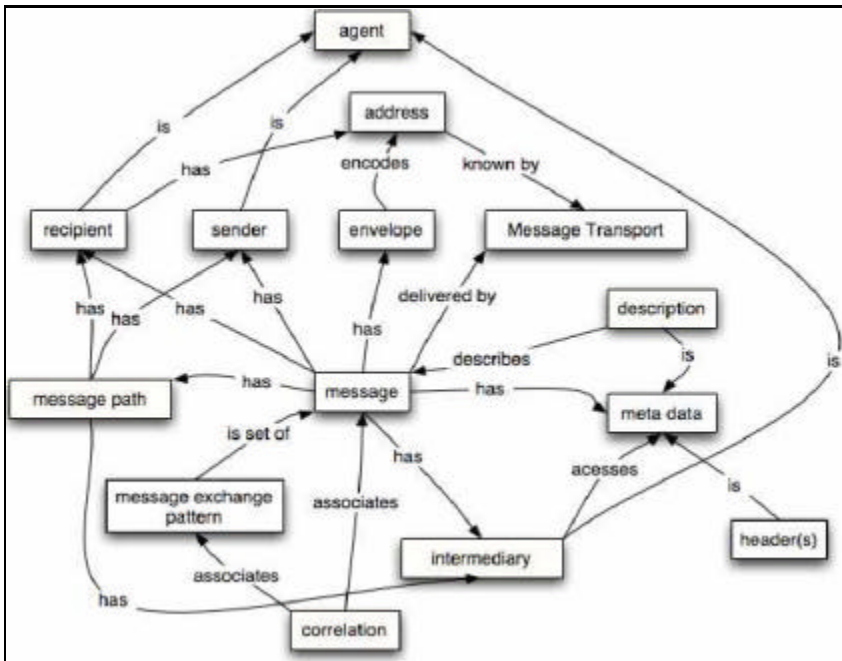


Figura 6. Relaciones entre los conceptos relacionados con los mensajes [28]

- *El Modelo Orientado a Servicios*, se centra en aspectos de la arquitectura relacionados con los servicios y las acciones principalmente. El propósito principal de este modelo es explicar

las relaciones existentes entre un agente, los servicios que ofrece o implementa y los solicitantes de los servicios. Como es lógico, un agente no podría llevar a cabo la tarea de ofrecer servicios a los clientes si no tuviera la capacidad de enviar y recibir mensajes, pero este modelo no hace referencia los mensajes o al método de transporte de los mismos. El Modelo Orientado a Servicio se construye sobre el Modelo Orientado a Mensajes, pero centrándose en las acciones en lugar de los mensajes. A continuación mostramos algunos conceptos relacionados con este modelo y las relaciones existentes entre ellos:

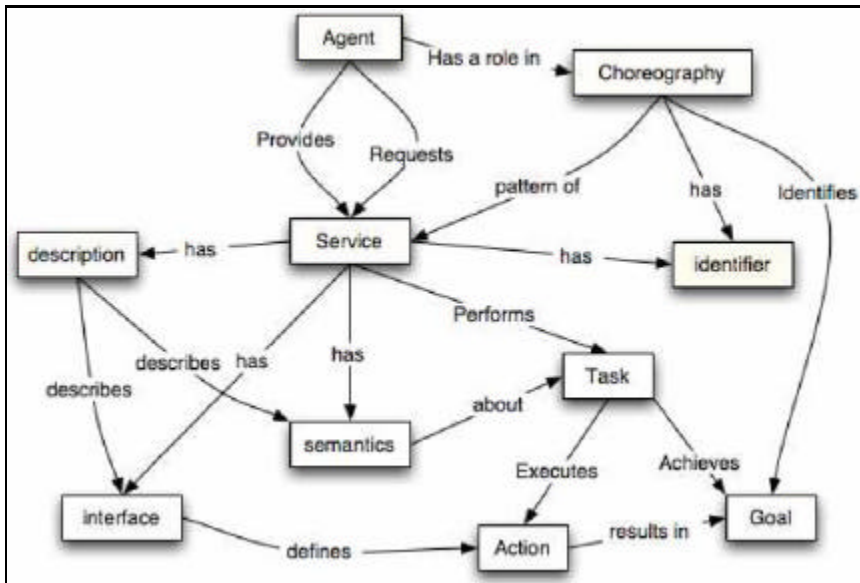


Figura 7. Relaciones entre los conceptos relacionados con los servicios [28]

- *El Modelo Orientado a Recursos*, que se centra en aquellos aspectos de la arquitectura relacionados con los recursos y el modelo de servicio asociado con la manipulación de los mismos. El Modelo Orientado a Recursos se construye sobre el Modelo Orientado a Servicios, principalmente por el desarrollo del modelo de servicio asociado al recurso y el conjunto de acciones para su manipulación. La función principal de esta parte de la arquitectura es explicar la Web en sí, y cómo se relaciona con los SW. Este modelo intenta explicar esto último mostrando como un los recursos son un concepto independiente, y como la manipulación de éstos son solamente una instancia del Modelo de Servicios, sólo que con sus propios servicios y acciones sobre los

recursos. A continuación, mostramos algunos conceptos relacionados con este modelo y las relaciones existentes entre ellos:

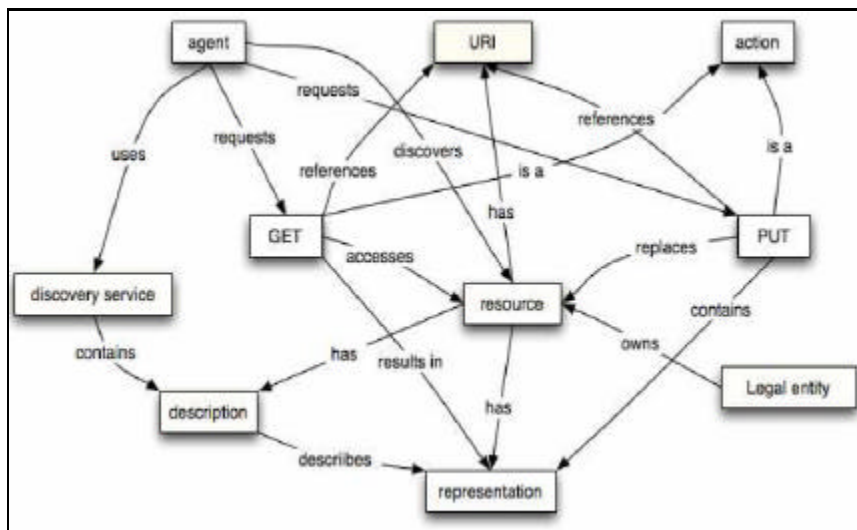


Figura 8. Relaciones entre los conceptos relacionados con los recursos [28]

- El Modelo de Políticas*, se centra en aquellos aspectos de la arquitectura relacionados con las políticas, y por extensión, con la seguridad y la calidad del servicio. El concepto de *política* es muy simple, es simplemente una restricción sobre el comportamiento de los agentes y los servicios. La seguridad, es básicamente un conjunto de restricciones respecto del comportamiento de las acciones y del acceso a los recursos. De manera similar, la calidad se considera también un tipo de restricción que puede ser aplicada a los servicios. En el Modelo de Políticas, estas restricciones se modelan alrededor de los conceptos de *política* y sus relaciones con otros elementos de la arquitectura, por ello, el Modelo de Políticas resulta ser un marco de trabajo en el que implementar la seguridad. Sin embargo, existen muchos otros tipos de restricciones y políticas relevantes a los SW, incluyendo restricciones a nivel de aplicación.

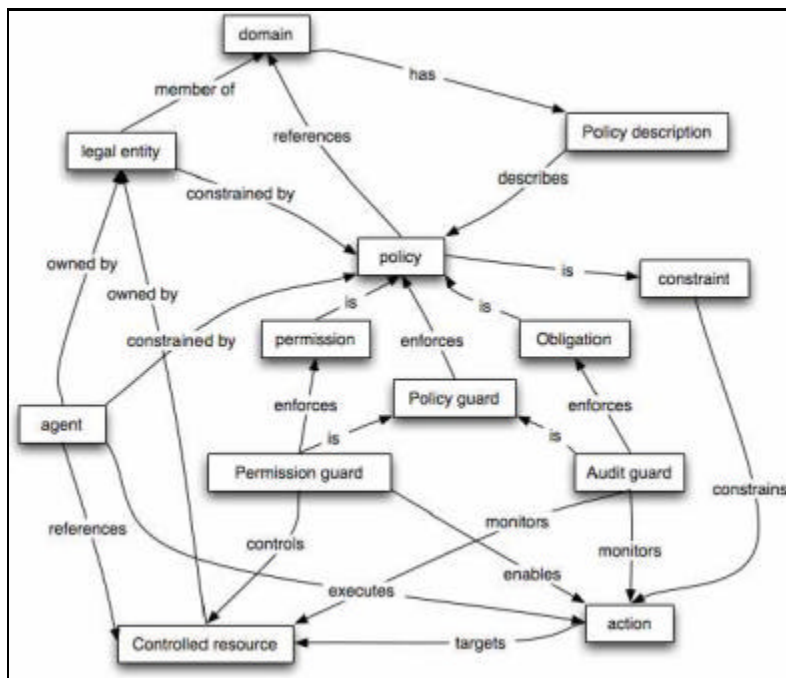


Figura 9. Relaciones entre los conceptos relacionados con las políticas [28]

- El Modelo de Gestión*, se centra en los aspectos relacionados con la gestión y la administración de los SW. En este caso, el propósito de este modelo es intentar implementar un *meta-control* de los SW, utilizando a su vez la tecnología de servicios. Este modelo abarca muchas características y conceptos de la arquitectura, incluyendo entre ellos, conceptos como recurso, descripción, etc. Típicamente, la vista administrativa de un recurso, será una *meta-vista* del mismo. La idea principal sobre la que gira este modelo es la de que controlar un SW es diferente que usar un SW. No obstante, para controlar un SW hay que utilizar interfaces que lo describen.

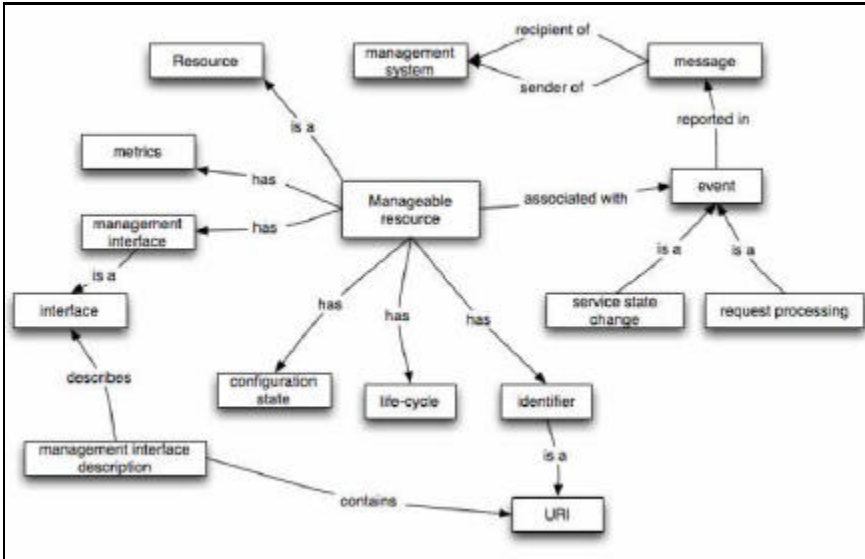


Figura 10. Relación entre conceptos relacionados con la gestión y administración de SW [28]

Como hemos comentado anteriormente, los distintos modelos arquitectónicos tratan cada uno un aspecto distinto de la arquitectura, no pudiendo evitar de ninguna forma, que estos aspectos estén relacionados entre sí. La relación interna de los distintos aspectos presentes en la arquitectura tiene como implicación directa que los modelos arquitectónicos que los abordan también estén relacionados.

3. ESTÁNDARES BÁSICOS

3. ESTÁNDARES BÁSICOS

Los SW cumplen las propiedades de interoperabilidad, tienen interfaces fuertemente tipadas, usan los estándares de Internet existentes, tienen soporte multilinguaje, soportan otras infraestructuras de componentes, etc. Estas propiedades sólo son alcanzables gracias a la base que sustenta a esta joven tecnología. Bajo el concepto de SW, subyacen una serie de estándares que ofrecen seguridad y confiabilidad a todos los aspectos que rodean a los SW. Estos estándares (Tabla 1) tratan de describir y definir todo lo referente a los servicios, desde cómo se describen o se localizan hasta cómo se representa toda la información, pasando por la forma mediante la cual se deben comunicar.

Estándares	
WSDL	Descripción formal de los servicios
UDDI	Publicación y descubrimiento de SW
SOAP	Comunicación en entornos distribuidos
DISCO	Búsqueda de servicios en el servidor
XML	Formato universal de descripción de datos

Tabla 1. Estándares básicos sobre SW

Algunos de estos estándares, como WSDL, UDDI y SOAP serán tratados con más profundidad, ya que son los que cobran más importancia dentro de los SW, o por así decirlo, conforman el esqueleto de esta tecnología. Estos estándares sientan las bases y reglas a seguir por todos los usuarios y proveedores para una utilización universal, más allá de las plataformas y los lenguajes.

Para el desarrollo de esta sección, nos hemos basado en las especificaciones de los estándares publicadas por la W3C y en otras publicaciones como pueden ser [25] y [13], entre muchas otras.

3.1. WSDL (*Web Service Description Language*)

3.1.1. Introducción

Gracias a que los protocolos de comunicación y los formatos de los mensajes están estandarizados en la web, está llegando a ser posible el

hecho de poder describir las comunicaciones de forma estructurada. Respondiendo a esta posibilidad y necesidad aparece WSDL, que mediante XML define una gramática mediante la cual podemos describir los servicios de la red como colecciones de nodos de comunicación que tienen la capacidad de intercambiar mensajes. Las definiciones del servicio WSDL proveen documentación destinada a los sistemas distribuidos y hace las veces de especificación para automatizar los detalles involucrados en las aplicaciones de comunicación.

Un documento en WSDL define los *servicios* como una colección de nodos de red o *puertos*. En WSDL, las definiciones abstractas de los nodos y los mensajes se separan del uso concreto en la red o del formato de datos al que están ligados, ya que se describe de manera genérica. De esta forma, podemos reutilizar las definiciones abstractas: como *messages*, que son las descripciones abstractas de los datos que están siendo intercambiados, y los *tipos de puerto* (port types), que son las colecciones abstractas de operaciones. El protocolo concreto y las especificaciones del formato de los datos para un *tipo de puerto* en concreto, constituyen una vinculación que se puede reutilizar en función de nuestro contexto. Un puerto se define asociando una dirección de red a una vinculación reutilizable, y una colección de estos puertos definen un *servicio*.

Un documento WSDL utiliza los siguientes elementos en la definición de los servicios de red:

- *Tipos*: es algo así como un contenedor de las definiciones de los tipos de datos, utilizando algún sistema de tipos para ello, como por ejemplo XSD [34]
- *Mensaje*: definición abstracta de la clase de datos que están siendo transmitidos.
- *Operación*: descripción abstracta de una operación soportada por el servicio.
- *Tipo de Puerto*: conjunto abstracto de operaciones soportadas por uno o más nodos.

- *Vinculación*: protocolo concreto y especificación del formato de los datos de un *tipo de puerto* en particular.
- *Puerto*: nodo definido como la combinación de una vinculación y una dirección de red.
- *Servicio*: colección de nodos relacionados.

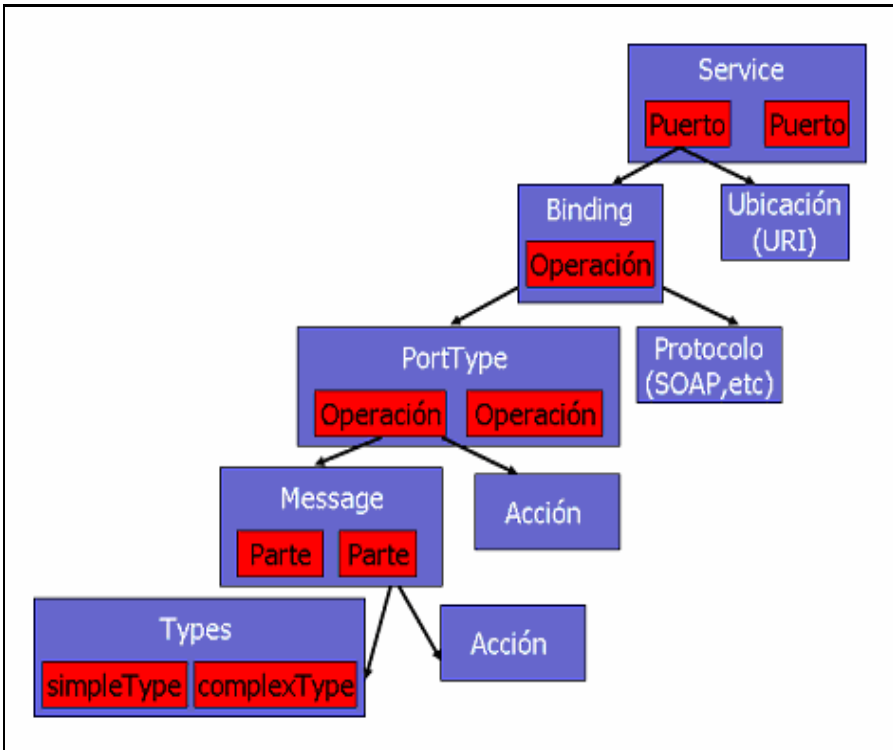


Figura 11. Cómo se superponen los elementos de WSDL para describir el SW.

El sistema de tipos que utiliza WSDL es el empleado en *XML Schema*, aunque podemos utilizar otros sistemas de tipos simplemente importándolos, ya que al igual que XML, es extensible. Lo que si hace WSDL es definir un mecanismo común de vinculación, mediante el cual vinculamos un protocolo específico, un formato de datos o una estructura a un mensaje abstracto, operación o nodo.

Además del marco de definición de servicios, en la especificación del WSDL de la W3C [29] se agregan un conjunto de *extensiones de*

vinculación para los siguientes protocolos y formatos de mensajes: SOAP 1.1, HTTP GET/POST y MIME.

3.1.2. Definición del Servicio

En este apartado, profundizaremos sobre los distintos elementos que suponen la base y el núcleo de la gramática WSDL.

3.1.2.1 Estructura del documento WSDL

Como veremos más adelante, un documento WSDL se compone de un conjunto de definiciones. Encontramos un elemento *definitions* en la raíz y diversas definiciones en su interior.

Los servicios son definidos utilizando seis elementos principales:

- *types*: proveen definiciones de los tipos de datos utilizados para describir los mensajes intercambiados.
- *message*: que representa una definición abstracta de los datos que están siendo transmitidos. Un mensaje se divide en una serie de partes lógicas, cada una de las cuales se asocia con alguna definición de algún sistema de tipos.
- *portType*: que es un conjunto de operaciones abstractas. Cada operación hace referencia a un mensaje de entrada y uno de salida.
- *binding*: que especifica un protocolo concreto y las especificaciones del formato de los datos de las operaciones y los mensajes definidos por un *portType* en concreto.
- *port*: que especifica una dirección para un *binding*, para así definir un único nodo de comunicación.
- *service*: que se utiliza para unir un conjunto de puertos relacionados.

Nominación y vinculación de documentos

WSDL permite asociar un *espacio de nombres* o *namespace* mediante la localización de un documento gracias a la sentencia *import*.

```
<definitions ... >
  <import namespace="uri" location="uri"/>
</definitions>
```

Figura 12. Atributo *import*.

Una referencia a una definición WSDL se hace usando un *QName* [34]. Se pueden referenciar los siguientes tipos de definiciones en un documento WSDL:

- Definiciones WSDL: *service*, *port*, *message*, *types*, *bindings* y *portType*.
- Otras definiciones: otras definiciones que se añadan mediante extensión, usando, eso sí, enlaces QName.

Documentación

La directiva *wSDL:document*, actúa como contenedor de información destinada a los usuarios humanos. Este elemento se puede colocar dentro de cualquier elemento existente dentro del documento WSDL.

3.1.2.2 Elemento *types*

Este elemento es el encargado de adjuntar las definiciones de los tipos de datos relevantes para el intercambio de mensajes. A fin de conservar la propiedad de interoperabilidad multiplataforma, WSDL utiliza el sistema de tipos descrito en XSD, considerándolo como el sistema de tipos por defecto.

```
<definitions ... >
  <types>
    <xsd:schema .../>
  </types>
</definitions>
```

Figura 13. Elemento *types*.

Se puede usar el sistema de tipos de XSD para definir los tipos en los mensajes sin reparar en que el formato utilizado sea XML, o si el esquema XSD resultante valida el formato utilizado en lugar de XML.

Es bastante lógico pensar que WSDL podrá describir todos los tipos abstractos de datos existentes, ya sean los actuales o los que tengamos que crear. Por ello, WSDL incorpora la posibilidad de, mediante extensión, ampliar la gramática del sistema de tipos mediante directivas destinadas a la extensión. Estos elementos de extensión aparecerán siempre dentro del elemento *types*, a fin de identificar los sistemas de definición de tipos, y proveer a la vez de un contenedor XML para las definiciones de tipos. Estos elementos de extensibilidad juegan un papel similar al que juega el elemento *xsd:schema* en la Figura 13.

3.1.2.3 Elemento *messages*

El elemento *messages* consiste en una o más partes lógicas, cada una de las cuales se asocia a un tipo perteneciente a un sistema de tipos mediante un atributo específico para tal fin. El conjunto de atributos que definen el tipo del mensaje también es extensible. Particularmente, WSDL define una serie de atributos para que mediante XSD, podamos establecer el tipo de los mensajes. Estos son:

- *element*: se refiere a un elemento XSD usando un QName.
- *type*: se refiere a un *simpleType* o *complexType* de XSD, mediante el uso de un QName.

La semántica para la definición de un mensaje es la siguiente:

```
<definitions ... >
  <message name="nmtoken">
    <part name="nmtoken" element="qname" type="qname"/>
  </message>
</definitions>
```

Figura 14. Elemento *message*.

El atributo *name* de *message* asigna un nombre único para ese mensaje dentro del documento WSDL. El atributo *name* del elemento *part* provee

un nombre único dentro de todas las partes de las que pueda disponer un mensaje.

Elemento *parts* del elemento *message*

Este es un mecanismo muy flexible, que sirve para describir el contenido abstracto de un mensaje. Un vínculo puede referenciar el nombre de un *part* con el fin de poder especificar información de vinculación acerca de ese *part*.

Si el mensaje consta de diferentes unidades lógicas, usaremos también múltiples elementos *part*.

```
<message name="MiMensaje">
  <part name="mimensaje" element="tns:MiMensaje"/>
  <part name="invoice" element="tns:Invoice"/>
</message>
```

Figura 15. Elemento *part*.

Podemos utilizar otra sintaxis diferente siempre y cuando los contenidos del mensaje sean lo suficientemente complejos de representar. Representaremos en este caso la estructura del mensaje directamente mediante el sistema de tipos, en cuyo caso, sólo podremos especificar un único elemento *part*.

```
<complexType name="Composicion">
  <choice>
    <element name="MiMensaje" minOccurs="1"
      maxOccurs="1" type="tns:MiMensajeType"/>
    <element name="Invoice" minOccurs="0"
      maxOccurs="unbounded" type="tns:InvoiceType"/>
  </choice>
</complexType>

<message name="MiMensaje">
  <part name="composicion" type="tns:Composite"/>
</message>
```

Figura 16. Descripción de un mensaje de contenido complejo.

3.1.2.4 Elemento *portType*

Un *portType* es un conjunto formado por operaciones abstractas sus mensajes abstractos relacionados.

```
<wsdl:definitions ...>
  <wsdl:portType name="Ejemplo">
    <wsdl:operation name="Ejemplo" .../>
  </wsdl:portType>
</wsdl:definitions>
```

Figura 17. Elemento *portType*.

El atributo *name* provee un identificador único al *portType* sobre todos los elementos del mismo tipo definidos dentro del documento WSDL. Una operación será referenciada mediante el atributo *name*.

En WSDL, existen cuatro primitivas de transmisión que puede soportar un nodo:

- *One-way (mensajes unidireccional)*: El nodo recibe un mensaje.
- *Request-response (mensajes de petición-respuesta)*: El nodo recibe un mensaje y envía el mensaje correspondiente.
- *Solicit-response (mensaje de solicitud-respuesta)*: El nodo envía un mensaje y recibe el mensaje correspondiente.
- *Notification (Mensaje de Notificación)*: El nodo envía un mensaje.

En WSDL, estas cuatro primitivas se denominan *operaciones*. A pesar de que podemos modelar las operaciones petición-respuesta y solicitud-respuesta como dos mensajes unidireccionales, es mejor modelar las operaciones como una operación atómica, porque entre otras cosas, son operaciones muy comunes y no es necesaria ningún tipo adicional de información entre los dos mensajes correlativos. Además, algunos nodos sólo pueden recibir mensajes si son los receptores de un mensaje de respuesta síncrono.

A pesar de que la estructura básica de WSDL soporta las cuatro primitivas de comunicación, WSDL solamente define vinculaciones para las primitivas *unidireccionales* (One-way) y de petición-respuesta (Request-response), por lo que las especificaciones que definen los protocolos para las primitivas de solicitud-respuesta (Solicit-response) y

notificación (Notification), incluirán vínculos a extensiones WSDL para permitir el uso de estas primitivas.

Las operaciones hacen referencia a los mensajes involucrados usando el atributo *message*, que es de tipo QName.

Operación del tipo *One-way*

La gramática para una operación de este tipo es la siguiente:

```
<wsdl:definitions ...>
  <wsdl:portType ...>
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:input name="nmtoken" message="qname"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figura 18. Operación *One-way*.

El elemento *input* especifica el formato del mensaje abstracto para esta operación.

Operación del tipo *Request-response*

La gramática para una operación de este tipo es la siguiente:

```
<wsdl:definitions ...>
  <wsdl:portType ...>
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:input name="nmtoken" message="qname"/>
      <wsdl:output name="nmtoken" message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figura 19. Operación *Request-response*.

Los elementos *input* y *output* especifican los formatos de los mensajes abstractos de petición y respuesta respectivamente. El elemento *fault* es opcional, y hace referencia al formato del mensaje abstracto que se

enviaría en caso de que durante la ejecución de una operación solicitada sucediera un error inesperado.

Operación del tipo *Solicit-Response*

La gramática para una operación de este tipo es la siguiente:

```
<wsdl:definitions ...>
  <wsdl:portType ...>
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:output name="nmtoken" message="qname"/>
      <wsdl:input name="nmtoken" message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figura 20. Operación *Solicit-response*.

Los elementos *output* e *input* especifican los formatos de los mensajes abstractos de petición y respuesta respectivamente. Al igual que en la operación *Request-response*, el elemento *fault* es opcional, y hace referencia al formato del mensaje abstracto que se enviaría al ocurrir un error por causa de la ejecución de la operación.

Operación *Notification*

La gramática para una operación de este tipo es la siguiente:

```
<wsdl:definitions ...>
  <wsdl:portType ...>
    <wsdl:operation name="nmtoken" >
      <wsdl:output name="nmtoken" message="qname"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figura 21. Operación *Notification*.

El elemento *output* especifica el formato del mensaje abstracto de la operación de notificación.

3.1.2.5 Vinculaciones o *Bindings*

Una vinculación define el formato de un mensaje y los detalles del protocolo para las operaciones y mensajes definidos por un *portType* concreto. Para un *portType* dado puede haber un número indefinido de asociaciones. La gramática para una vinculación es la siguiente:

```
<wsdl:definitions .... >
  <wsdl:binding name="nmtoken" type="qname" > *
    <!-- extensibility element (1) --> *
    <wsdl:operation name="nmtoken" > *
      <!-- extensibility element (2) --> *
      <wsdl:input name="nmtoken"? > ?
        <!-- extensibility element (3) -->
      </wsdl:input>
      <wsdl:output name="nmtoken"? > ?
        <!-- extensibility element (4) --> *
      </wsdl:output>
      <wsdl:fault name="nmtoken" > *
        <!-- extensibility element (5) --> *
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Figura 22. Gramática de una vinculación.

El atributo *name* del elemento *binding* provee de un nombre único para todos los elementos *binding* definidos dentro del documento WSDL. Un elemento *binding* referencia a un *portType* mediante el atributo *type* (Figura 22).

Los elementos de extensión del elemento *binding*, se usan para especificar la gramática en concreto para los mensajes de entrada, salida y fallo.

Un elemento *binding* especificará exactamente un protocolo, pero no debe especificar información de la dirección.

3.1.2.6 Puertos o *Ports*

Un puerto define un nodo individual mediante la especificación de una única dirección para una vinculación o *binding*.

```
<wsdl:definitions .... >
  <wsdl:service .... > *
    <wsdl:port name="nmtoken" binding="qname"> *
      <!-- extensibility element (1) -->
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figura 23. Definición de un puerto.

El atributo *name* provee un único nombre a través de todos los puertos definidos dentro del mismo documento WSDL.

Los elementos de extensibilidad de la vinculación son usados para especificar la información de la dirección para el puerto.

Un puerto no debe especificar más de una dirección ni tampoco debe especificar más información de vinculación que la información de dirección.

3.1.2.7 Servicios o *service*

Un servicio agrupa un conjunto de puertos relacionados:

```
<wsdl:definitions .... >
  <wsdl:service name="nmtoken"> *
    <wsdl:port .... /*>
  </wsdl:service>
</wsdl:definitions>
```

Figura 24. Definición de un servicio.

Al igual que en los anteriores elementos definidos, el atributo *name* provee un nombre único a través de todos los servicios definidos dentro del documento WSDL.

Los puertos que se hallan dentro del mismo servicio tienen la siguiente relación:

- Ninguno de los puertos se comunica con ninguno de los otros, es decir, en ningún caso la salida de un puerto será la entrada de otro.
- Si un servicio tiene múltiples puertos que comparten un *portType*, pero empleando diferentes vinculaciones o direcciones, entonces los puertos son alternativos. Cada puerto tiene el mismo comportamiento (semánticamente). Con esto, los usuarios de los documentos WSDL pueden elegir entre el puerto que mejor se ajuste a sus necesidades.
- Mediante el examen de sus puertos, podemos determinar los tipos de puerto de un servicio. Esto permite a los consumidores del documento WSDL determinar si desean comunicarse con un servicio en particular basándose en si soporta o no determinados tipos de puertos. Esto es útil si existe alguna relación implícita entre las operaciones de los tipos de puertos, y si el conjunto completo de tipos de puertos debe ser presentado de forma que se lleve a cabo alguna tarea en particular.

3.1.3. Vinculaciones SOAP o *SOAP Binding*

WSDL incluye una vinculación para los nodos que utilizan el estándar SOAP 1.1, y que soportan la siguiente especificación específica sobre el protocolo:

- Una indicación de que una vinculación está sujeta al protocolo SOAP 1.1.
- Una forma de especificar una dirección para un nodo SOAP.
- La URI para la cabecera del protocolo HTTP, *SOAPAction*, para la vinculación de SOAP con el protocolo.
- Una lista de definiciones para las cabeceras que serán transmitidas como parte del contenedor del mensaje SOAP.

Las vinculaciones SOAP que utilicen un esquema de direccionamiento distinto a un URI, lo especificarán reemplazando el elemento *soap:address* por el correspondiente. Aparte, en caso de que la

vinculación no necesite especificar el atributo *SOAPAction* podrá omitirlo.

3.1.3.1 Elemento *soap:operation*

Este elemento provee información acerca de la operación. Por ejemplo:

```
<definitions .... >
  <binding .... >
    <operation .... >
      <soap:operation soapAction="uri"? style="rpc|document"?>
    </operation>
  </binding>
</definitions>
```

Figura 25. Elemento *soap:operation*.

Mediante el atributo *style*, se indica si la operación es estilo *RPC* o *document*. Si es estilo *RPC*, los mensajes contienen los parámetros y los valores de retorno, si es estilo *document*, los mensajes simplemente contienen documentos. El valor de este atributo afecta en la forma en la que construiremos el cuerpo del mensaje SOAP. En caso de que el valor del atributo no se especifique, tomará como valor por defecto el contenido del elemento *soap:binding*. En caso de que tampoco aquí aparezca el estilo, se asumirá por defecto que es del tipo *document*.

El atributo *soapAction*, especifica el valor de la cabecera *SOAPAction* para esta operación. El valor de esta URI se debe usar directamente como el valor para la cabecera *SOAPAction*. Cuando estamos haciendo una petición, no se debe intentar hacer absoluto el valor de un URI relativa. Cuando el mensaje sea transportado mediante HTTP, es necesario que este atributo tenga valor, no existiendo ningún valor por defecto. Si utilizamos otro protocolo, no deberemos especificar este valor, y además, el elemento *soap:operation* debe ser omitido.

3.1.3.2 Elemento *soap:body*

Este elemento especifica como las distintas partes del mensaje aparecen dentro del elemento *Body* de SOAP.

Su elemento *binding* contiene información acerca de cómo ensamblar las distintas partes del mensajes dentro del elemento *Body* del mensaje

SOAP. El elemento *soap:body* se usa tanto si el documento es orientado a RPC como si es orientado a documento, sólo que el atributo *style* de las operaciones incluidas tendrá importantes efectos en como se estructurará la sección *Body*:

- Si el estilo de la operación es *RPC*, cada parte es un parámetro o un valor de retorno y aparece en el interior de un elemento contenedor dentro del cuerpo. El elemento contenedor recibe el mismo nombre que la operación, y su espacio de nombres es el mismo que el valor del atributo *namespace*. Cada parte del mensaje o parámetro se encuentra dentro del contenedor, representado mediante un nombre de acceso idéntico al correspondiente al del parámetro de la llamada.
- Si es estilo de la operación es *document* no existen contenedores adicionales, apareciendo las partes del mensaje directamente bajo el elemento *Body*.

Para definir el contenido del elemento *Body* los elementos de acceso a los parámetros se siguen los mismos mecanismos.

```
<definitions .... >
  <binding .... >
    <operation .... >
      <input>
        <soap:body parts="nmtokens" use="literal|encoded"
          encodingStyle="uri-list" namespace="uri">
      </input>
      <output>
        <soap:body parts="nmtokens" use="literal|encoded"
          encodingStyle="uri-list" namespace="uri">
      </output>
    </operation>
  </binding>
</definitions>
</definitions>
```

Figura 26. Elemento *soap:body*.

El atributo *parts* es opcional, y si se omite, todas las partes definidas por el mensaje se supondrán incluidas en el elemento *soap:body*. Si por el contrario se incluye, indicará cuales de las partes del mensaje aparecerán en el elemento *soap:body*, las otras partes que aquí no se mencionen, podrán aparecer en otras zonas del mensaje.

El atributo obligatorio *use*, indica cuales de las partes del mensaje son codificadas mediante reglas de codificación, o cuales de las partes del mensaje definen el esquema concreto del mensaje.

Si el parámetro *use* toma el valor *encoded*, cada parte del mensaje referencia a un tipo abstracto mediante el atributo *type*. Estos tipos abstractos son usados para producir un mensaje en concreto mediante la aplicación de algún tipo de codificación especificada mediante el atributo *encodingStyle*.

Sin embargo, si el parámetro *use* toma el valor *literal*, entonces, cada una de las partes referencia a la definición de un esquema concreto, usando el atributo *element* o el *type*. En el primer caso, el elemento referenciado por la parte, aparecerá directamente dentro del elemento *Body* (para asociaciones del estilo *document*) o bajo algún elemento de acceso referenciado en la parte del mensaje (en vinculaciones estilo *RPC*). En el segundo caso, el tipo referenciado por la parte, se convierte en el tipo de esquema del elemento contenedor (el cuerpo o *Body* cuando el estilo sea *RPC* y el elemento de acceso cuando el estilo sea *document*). Cuando el valor del atributo *use* sea *literal*, debemos usar el valor del atributo *encodingStyle* para indicar que el formato de codificación ha sido derivado mediante una codificación en particular.

El valor del atributo *encodingStyle* está constituido por una lista de URIs, separadas entre si por un espacio en blanco. Las URIs representan formatos de codificación utilizados en el mensaje, por orden desde el más restrictivo al menos restrictivo.

3.1.3.3 Elemento *soap:fault*

Este elemento especifica los contenidos del elemento *fault* de SOAP, ubicado antes del elemento *soap:fault* (Figura 27).

```
<definitions .... >
  <binding .... >
    <operation .... >
      <fault>
        <soap:fault name="nmtoken" use="literal|encoded"
                    encodingStyle="uri-list" namespace="uri">
        </fault>
      </operation>
    </binding>
  </definitions>
```

Figura 27. Elemento *soap:fault*.

El atributo *name* relaciona el elemento *soap:fault* con el elemento *wSDL:fault* definido por la operación.

El elemento *fault* debe tener solamente un parte. Los atributos *use*, *encodingStyle* y *namespace* se usan de la misma forma en la que se usan con *soap:body*, solo que para el argumento *style* asumimos por defecto el valor *document*, ya que los fallos no contienen argumentos.

3.1.3.4 Elementos *soap:header* y *soap:headerdefault*

Estos elementos que nos permiten definir la cabecera, se incluyen dentro del elemento *header* del sobre SOAP (*SOAP Envelope*).

```
<definitions .... >
  <binding .... >
    <operation .... >
      <input>
        <soap:header message="qname" part="nmtoken" use="literal|encoded"
                    encodingStyle="uri-list" namespace="uri">
          <soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
                           encodingStyle="uri-list" namespace="uri"/>
        </soap:header>
      </input>
      <output>
        <soap:header message="qname" part="nmtoken" use="literal|encoded"
                    encodingStyle="uri-list" namespace="uri">
          <soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
                           encodingStyle="uri-list" namespace="uri"/>
        </soap:header>
      </output>
    </operation>
  </binding>
</definitions>
```

Figura 28. Elementos *soap:header* y *soap:headerdefault*.

Los atributos *use*, *encodingStyle* y *namespace* se usan de la misma manera en la que se usaban en el elemento *soap:body*, sólo que

asumimos que el elemento *style* toma el valor *document*, ya que las cabeceras no contienen parámetros.

Los atributos *message* y *part* hacen referencia a la parte del mensaje que define el tipo de cabecera. El esquema referenciado mediante el atributo *part* debería incluir definiciones para los atributos *soap:actor* y *soap:mustUnderstand* si el atributo *use* toma el valor *literal*, pero no deberá hacerlo si toma el valor *encoded*. El mensaje referenciado necesita ser el mismo que el definido en el cuerpo SOAP.

El elemento opcional *headerdefault* (que está ubicado dentro del elemento *soap:header*, y que tienen su misma sintaxis) permite especificaciones del/los tipo/s de cabecera que se usará para transmitir información de error perteneciente a la cabecera definida mediante el elemento *soap:header*. La especificación SOAP dice que los errores pertenecientes a las cabeceras deberán ser devueltos en las mismas, y este mecanismo admite especificaciones del formato de las cabeceras.

3.1.3.5 Elementos *soap:address*

La vinculación de dirección de SOAP se usa para dotar a cada puerto de una sola dirección o URI. El esquema URI especificado para la dirección debe corresponder al transporte especificado en *soap:binding*.

```
<definitions .... >
  <port .... >
    <binding .... >
      <soap:address location="uri"/>
    </binding>
  </port>
</definitions>
```

Figura 29. Elemento *soap:address*.

3.2. SOAP (Simple Object Access Protocol)

3.2.1. Introducción

SOAP es un protocolo que proporciona un mecanismo estándar de empaquetar mensajes [21]. Este protocolo está pensado para el intercambio de información en entornos descentralizados y distribuidos. Usa las tecnologías relacionadas con XML a fin de definir un marco de

trabajo extensible para los mensajes. Provee una estructura de mensajes capaz de ser intercambiada sobre una gran cantidad de protocolos de soporte. Este marco ha sido diseñado con el fin de que fuera independiente del cualquier modelo de programación y otras implementaciones de semánticas.

Los dos objetivos de diseño principales de SOAP son la simplicidad y la extensibilidad. Para alcanzar estos objetivos, SOAP simplemente elimina de su arquitectura aquellos aspectos que con más frecuencia se encuentra en los sistemas distribuidos. Podemos agregar las características que nosotros queramos simplemente extendiendo la especificación.

La actual versión de SOAP (v 1.2) [26][27] está distribuida básicamente en tres partes:

- *Primera parte.* Definición del marco de trabajo para los mensajes de SOAP. Consiste en:
 1. El modelo de procesamiento de SOAP, que define las reglas para procesar un mensaje SOAP.
 2. El modelo de Extensibilidad de SOAP, que define los conceptos relacionados con las características y los módulos de SOAP.
 3. La construcción del mensaje SOAP, que define la estructura de un mensaje que sigue el protocolo SOAP.
- *Segunda parte.* Documento de introducción cuyo propósito no es otro que el de proveer de un sencillo tutorial sobre las características de la versión 1.2 de SOAP.
- *Tercera parte.* Describe el conjunto de adjuntos que puede usarse en conexión con el marco de mensajes de SOAP.

Las ventajas que ofrece SOAP son las siguientes [21]:

- No está asociado a ningún lenguaje. SOAP no especifica una API, por lo que la implementación de esta se deja al lenguaje de programación que se elija.

- No se encuentra fuertemente asociado a ningún protocolo de transporte. Un mensaje de SOAP no es sino un documento en XML, por lo que la única característica que se le exigirá al protocolo de transporte es que tenga la capacidad de transmitir cadenas de texto.
- No está ligado a ninguna infraestructura de objetos distribuidos. SOAP brinda cierto grado de interoperabilidad entre sistemas diferentes en los que se ejecutan componentes *middleware* de distintos fabricantes.
- Aprovecha los estándares existentes en la industria. Al crear SOAP, no se reinventaron conceptos, sino que se tomaron estándares existentes y se extendieron. Como ocurre con XML, el cual se toma para codificar los mensajes. SOAP también toma como sistema de tipos propio el definido en la especificación de Esquema XML [34]. En lo que respecta al protocolo de transporte, SOAP sólo impone como restricción que éste pueda transmitir cadenas de caracteres, que es en lo que consiste al fin y al cabo un mensaje SOAP.
- Permite la interoperabilidad entre múltiples entornos. La neutralidad y la interoperabilidad son algunas de las premisas de este protocolo. Gracias a estar basado íntegramente en estándares de la red y utilizar XML para dar formato a la información, SOAP permite una interoperabilidad íntegra entre plataformas distintas. Así y de esta forma, una aplicación puede comunicarse con una aplicación del *back-end* ejecutándose en un *mainframe* bajo *UNIX* capaz de enviar y recibir XML sobre HTTP o algún otro protocolo.

3.2.2. Conceptos importantes

En esta sección veremos algunos conceptos relativos a SOAP importantes para comprender bien su funcionamiento.

Conceptos del Protocolo

- SOAP → Es el conjunto de convenciones que gobiernan el formato y las reglas procesamiento de los mensajes SOAP.

Dentro de estas convenciones, también se incluye la interacción entre nodos SOAP que generan y aceptan mensajes SOAP con el propósito de intercambiar información a lo largo del camino de un mensaje SOAP.

- **Nodo SOAP (*SOAP node*)**→ Es la encarnación del procesamiento lógico necesario para transmitir, recibir procesar y/o retransmitir un mensaje SOAP de acuerdo a las convenciones dictadas por SOAP. Un nodo accede a los servicios provistos por los protocolos subyacentes a través de una o más asociaciones SOAP.
- **Papel o Rol SOAP (*SOAP role*)**→ Una funcionalidad esperada en el receptor SOAP durante el procesamiento de un mensaje. Un receptor SOAP puede jugar múltiples papeles.
- **Vinculación SOAP (*SOAP binding*)**→ Es el conjunto formal de reglas para el transporte de un mensaje SOAP a través o sobre otro protocolo con el propósito de intercambiarlo. Un ejemplo de ello puede ser el transporte del mensaje empaquetado sobre HTTP, o mediante una cadena sobre TCP.
- **Característica SOAP (*SOAP feature*)**→ Extensión del marco de mensajes SOAP, como pueden ser la fiabilidad, la seguridad, la correlación, el enrutamiento y los *Patrones de Intercambio de Mensajes* (MEPs, Message Exchange Patterns).
- **Módulo SOAP (*SOAP module*)**→ Es una especificación que contiene combinadas la sintaxis y la semántica de los bloques de cabeceras SOAP de acuerdo a las reglas de los Módulos SOAP. Un módulo SOAP realiza o implementa cero o más características de SOAP.
- **Patrón de intercambio de mensajes SOAP (*Message Exchange Pattern, MEP*)** → Es algo así como una plantilla para el intercambio de mensajes SOAP entre nodos SOAP mediante una o más asociaciones subyacentes.
- **Aplicación SOAP (*SOAP application*)**→ Es una entidad, típicamente software, que produce, consume, o que actúa de

alguna manera sobre mensajes SOAP pero siempre de acuerdo al modelo de procesamiento SOAP.

Conceptos sobre Encapsulación de Datos

A continuación se enumeran algunos aspectos relacionados con la encapsulación de los datos:

- Mensaje SOAP → Unidad básica de comunicación entre dos nodos SOAP.
- Sobre SOAP → Elemento más externo de un mensaje SOAP.
- Cabecera SOAP → Es una colección de cero o más bloques de cabecera SOAP, cada uno de los cuales puede ser dirigido a un receptor SOAP dentro del camino del mensaje SOAP.
- Bloque de cabecera SOAP → Elemento de información usado para delimitar datos que constituyen lógicamente una única unidad computacional dentro de la cabecera SOAP.
- Cuerpo SOAP → Es una colección de cero o más elementos de información que apuntan al último receptor SOAP del camino del mensaje SOAP.
- Fallo SOAP → Elemento de información que contiene información del fallo generado por un nodo SOAP.

Conceptos sobre el emisor y el receptor de un mensaje

A continuación se enumeran algunos aspectos relacionados con emisores y los receptores de los mensajes SOAP:

- Emisor SOAP → Nodo SOAP que transmite un mensaje.
- Receptor SOAP → Nodo SOAP que acepta un mensaje SOAP.
- Camino de mensaje SOAP → El conjunto de nodos SOAP a través de los cuales debe pasar un mensaje SOAP para viajar del

emisor inicial al receptor final. En este camino se incluyen el emisor inicial SOAP, cero o más intermediarios SOAP y un receptor SOAP en último lugar.

- Emisor inicial SOAP → El emisor SOAP que origina el mensaje SOAP en el punto de inicio del camino del mensaje SOAP.
- Intermediario SOAP → Un intermediario SOAP puede ser tanto un receptor SOAP como un emisor SOAP. Este intermediario es apuntado por un mensaje SOAP. Procesa los bloques de cabecera SOAP que apuntan a él y actúa para enviar el mensaje SOAP hacia el receptor SOAP final.
- Receptor final SOAP → Es el destinatario final del mensaje SOAP. Es el responsable de procesar los contenidos del cuerpo SOAP y cualquier bloque de cabecera SOAP que apunte hacia él. En algunas circunstancias, un mensaje SOAP puede no alcanzar un receptor final SOAP, como ocurriría si hubiera algún problema con un intermediario. Un receptor final SOAP nunca podrá ser a la vez un intermediario SOAP para el mismo mensaje.

3.2.3. Modelo de Procesamiento SOAP

SOAP provee un modelo de procesamiento distribuido en el que un mensaje es originado por un emisor inicial SOAP, y enviado a un receptor final SOAP a través de cero o más intermediarios SOAP. Este modelo puede soportar muchos *Patrones de Intercambio de Mensajes* o MEPs, incluyendo mensajes unidireccionales, interacciones del tipo petición/respuesta y conversaciones punto a punto.

Este modelo especifica cómo un receptor SOAP procesa el mensaje que le ha llegado. En el modelo no está contemplado el mantener ningún estado en la comunicación, o el llevar un control de la correlación de los mensajes. Sería responsabilidad de las características adicionales el definir de alguna forma este tipo de procesamiento.

3.2.3.1. Nodos SOAP

Un nodo SOAP puede ser un emisor inicial SOAP, un receptor final SOAP o un intermediario. Cuando un nodo SOAP reciba un mensaje

debe procesarlo de acuerdo al modelo de procesamiento descrito anteriormente. Un nodo SOAP se identifica mediante un URI.

3.2.3.2. Roles y Nodos SOAP

Cuando se procesa un mensaje SOAP, un nodo SOAP puede jugar uno o más roles SOAP durante este procesamiento. Cada uno de los roles SOAP se identifica mediante un URI conocida como *nombre del rol SOAP*. El rol que juega un nodo no puede variar a lo largo del procesamiento de un único mensaje SOAP. No hay nada escrito acerca de si un nodo SOAP debe jugar distintos roles en el caso de que esté procesando más de un mensaje SOAP.

Algunos de los roles más importantes para los mensajes SOAP son:

Prefijo	Espacio de Nombres	Notas
env	"http://www.w3.org/2003/05/soap-envelope"	Puede encontrarse una normativa para XML Schema para el espacio de nombres del sobre SOAP en http://www.w3.org/2003/05/soap-envelope .
xs	"http://www.w3.org/2001/XMLSchema"	Espacio de nombres de la especificación de XML Schema

Tabla 2. Roles relevantes para los mensajes SOAP.

El propósito de un nombre de rol SOAP es identificar uno o varios nodos SOAP, pero no tiene asociadas semánticas para el enrutamiento o el intercambio de mensajes. Por ejemplo, los roles SOAP pueden ser nombrados mediante un URI que sirva para enrutar los mensajes SOAP a un nodo SOAP apropiado. Y al contrario, también es apropiado utilizar roles SOAP con nombres que estén relacionados más indirectamente con el enrutamiento de mensajes.

Con la excepción de los dos nombres de roles SOAP definidos en la Tabla 2, la especificación no describe los criterios mediante los cuales un nodo dado determina el conjunto de roles sobre los que actuar ante un mensaje dado. Es decir, las implementaciones de otros roles no indicados en la Tabla 2, tendrán que decidir si actuar en función de criterios propios, que pueden ir por ejemplo en función de alguna información incluida en el mensaje, entre otras cosas.

3.2.3.3. Apuntar a Bloques de Cabecera SOAP

Un bloque de cabecera SOAP puede llevar un atributo de información llamado *role*, que se usa para apuntar a nodos SOAP que jueguen el rol especificado. Esta especificación se refiere al valor del atributo *role* SOAP como el rol SOAP para el correspondiente bloque de cabecera SOAP.

Se dice que un bloque de cabecera SOAP apunta a un nodo SOAP si el rol SOAP para el bloque de cabecera es el nombre del rol sobre el que opera el nodo SOAP.

3.2.3.4. Comprensión de los Bloques de Cabecera SOAP

Se dice que un bloque de cabecera SOAP será inteligible por un nodo SOAP si el software de este nodo ha sido escrito de forma que esté en conformidad con XML e implemente las semánticas especificadas para los nombres expandidos de XML de los elementos de información más externos de ese bloque de cabecera.

Un bloque de cabecera debe incluir un atributo de información *mustUnderstand*. Cuando el valor de este atributo sea *true*, se dice que el bloque de cabecera SOAP es *obligatorio*, y debe ser inteligible.

Se supone, que de algún modo, los bloques de cabecera obligatorios modifican las semánticas de otros bloques de cabecera o elementos del cuerpo SOAP. Por lo tanto, para todos los bloques de cabecera SOAP obligatorios que apunte a un nodo, este nodo deberá o no, procesar el mensaje SOAP, generando un fallo si no se hiciera de forma correcta. Etiquetando los bloques cabecera SOAP como obligatorios, aseguramos que no serán ignorados por un nodo SOAP al que apunte el bloque de cabecera, de forma que sabremos si hubo un fallo.

El atributo *mustUnderstand* no está pensado como un mecanismo para detectar errores en el enrutamiento, fallos en la identificación de los nodos, fallo de un nodo al actuar en el rol que debía jugar, etc. Cualquiera de estas condiciones puede concluir en un fallo durante el procesamiento de un bloque de cabecera SOAP perteneciente al sobre SOAP (*SOAP Envelope*).

3.2.3.5. Estructura e Interpretación de los Cuerpos SOAP

Un receptor SOAP final, debe procesar correctamente el contenido del cuerpo SOAP. En cualquier caso y con la excepción de los fallos SOAP, no se impone ninguna estructura o procesamiento en particular de los elementos ubicados en el cuerpo SOAP, al igual que tampoco existe ningún estándar que diga el tipo de procesamiento que hay que hacer.

3.2.3.6. Procesamiento de los Mensajes SOAP

Para el correcto procesamiento de los mensajes SOAP hace falta establecer un conjunto de reglas. Nada en esta especificación puede prever el uso de técnicas que puedan ofrecer un incremento de la flexibilidad en el orden de procesamiento de los mensajes. El procesamiento de todos los mensajes SOAP generados, los fallos SOAP y los efectos producidos del lado de la aplicación, deben ser semánticamente equivalentes para realizar los pasos siguientes de manera separada y en el orden dado:

1. Determinar el conjunto de roles que está jugando el nodo SOAP. El contenido del sobre SOAP, incluidos los bloques de cabecera SOAP y el cuerpo SOAP, deben ser inspeccionados para preparar una decisión.
2. Identificar en el nodo todos los bloques de cabecera que son obligatorios.
3. Si uno o más de los bloques de cabecera SOAP identificados en el paso anterior no pueden ser comprendidos por el nodo, entonces se genera un único fallo SOAP con el valor del atributo *value* del elemento *code* establecido a “*env:MustUnderstand*”. Si al menos se genera un fallo, no se debe realizar ningún tipo de procesamiento. Los fallos relacionados con los contenidos del cuerpo SOAP no se deben generar en este paso.
4. En los nodos SOAP intermedios, procesar todos los bloques de cabecera SOAP obligatorios que apunten al nodo, y cuando se trate del último receptor SOAP, procesar el cuerpo SOAP. Un nodo SOAP puede optar por procesar los bloques de cabecera SOAP que lo referencia de manera opcional.

5. En el caso de que se trate de un intermediario SOAP, cuando el patrón de intercambio de mensajes SOAP y los resultados del procesamiento requieran que el mensaje SOAP sea enviado más adelante en el camino del mensaje SOAP, se retransmitirá el mensaje como se describe en Sección 3.2.3.7.

En todos los casos donde el bloque de cabecera SOAP sea procesado, el bloque debe ser comprendido por el nodo SOAP, y el procesamiento para ese bloque de cabecera debe estar de acuerdo con la especificación. El hecho de que una cabecera de bloque haya sido procesada satisfactoriamente no garantiza que otro bloque de cabecera del mismo mensaje con el mismo nombre extendido en XML sea procesado también de forma satisfactoria, ya que la especificación para el bloque de cabecera determina las circunstancias en las que el procesamiento puede resultar en un fallo. Un receptor SOAP final debe procesar el cuerpo SOAP de una manera coherente con la Sección 3.2.3.5.

Los fallos se indican mediante la generación de un fallo. El procesamiento de un mensaje SOAP puede dar lugar a la generación de como mucho un fallo.

Un mensaje puede contener o producir múltiples errores en su procesamiento. Excepto donde el orden de detección está especialmente indicado, un nodo SOAP tiene la libertad de reflejar cualquier fallo del conjunto de posibles fallos prescritos por los errores encontrados durante el procesamiento.

Los nodos SOAP pueden hacer referencia a cualquier información en el sobre SOAP cuando se está realizando el procesamiento del cuerpo o del bloque de cabecera SOAP. Si se deseara, puede ocultarse el mensaje completo mediante una función, a fin de que mantenga la privacidad del contenido.

El procesamiento de uno o más bloques de cabecera SOAP puede controlar o determinar el orden de procesamiento para otros bloques de cabecera SOAP y/o cuerpo SOAP. Por ejemplo, podríamos crear un bloque de cabecera SOAP para forzar el procesamiento de otros bloques de cabecera SOAP en un orden léxico. Si no existiera un bloque de cabecera SOAP que controlara el proceso, el orden de procesamiento de los bloques de cabecera puede realizarse en un orden arbitrario. El

procesamiento de un bloque de cabecera, puede realizarse antes que el procesamiento del cuerpo (por ejemplo un bloque que sirviera para autenticar la identidad del nodo) SOAP, intercalado con este (cuando el bloque de cabecera haga las veces de fichero *log*), o incluso después (un bloque de cabecera que remita una contestación de que una transacción se ha realizado correctamente).

3.2.3.7. Retransmisión o reenvío de Bloques de Cabecera SOAP

Los mensajes se originan en un nodo SOAP inicial y se envían un receptor SOAP final a través de cero o más intermediarios SOAP. Como SOAP no define ninguna semántica de enrutamiento o envío, podemos describir esta funcionalidad mediante una o más características SOAP (*SOAP features*). (Sección 3.2.4). Aquí vamos a ver como el envío de un mensaje interacciona con el modelo de procesamiento distribuido SOAP.

SOAP define dos tipos de intermediarios: intermediarios de envío e intermediarios activos.

Reenvío de Bloques de Cabecera SOAP

El envío de bloques de cabecera SOAP que apuntan a un intermediario SOAP depende de que el bloque de cabecera SOAP sea procesado por el nodo. Se dice que un bloque de cabecera SOAP es *reinsertado* si al procesar ese bloque de cabecera, se determina que ese bloque debe ser *reinsertado* en el mensaje que se enviará. La especificación para un bloque de cabecera SOAP puede demandar que ese bloque sea enviado en el mensaje de envío si el bloque de cabecera está dirigido al rol jugado por el intermediario SOAP, pero en otro caso no será procesado por el intermediario, ya que no le corresponde. Por lo tanto, se dice que es *retransmitible*.

Un bloque de cabecera SOAP puede tener un atributo informativo *relay* (retransmitir). Diremos que el bloque de cabecera es *retransmitible* cuando el valor de este atributo sea *true*. El atributo *relay* no tiene efecto en los bloques de cabecera SOAP que estén dirigidos al rol que juega el intermediario SOAP. El atributo tampoco tendrá efecto sobre el modelo de procesamiento SOAP cuando el bloque de cabecera lleve también un atributo *mustUnderstand* con el valor *true*. Y por último, tampoco tendrá

efecto en el procesamiento de mensajes SOAP cuando se trate del receptor SOAP final.

Intermediarios de Envío SOAP

La semántica de uno o más bloques de cabecera SOAP en un mensaje SOAP, o el patrón de intercambio de mensajes SOAP puede requerir que el mensaje SOAP sea reenviado a otro nodo SOAP en nombre del que inició el envío del mensaje que llega. En este caso, el nodo que realiza el procesamiento es el nodo que juega el rol *intermediario de envío SOAP*.

Los intermediarios de envío SOAP deben procesar el mensaje de acuerdo al modelo de procesamiento SOAP definido en la Sección 3.2.3.6. Además, cuando se genera un mensaje SOAP con el propósito de enviarlo, los nodos deben:

1. Eliminar todos los bloques de cabecera SOAP procesados.
2. Eliminar todos los bloques de cabecera SOAP que apuntaban al nodo que envía el mensaje, que fueron ignorados durante el procesamiento y que no son *reenviables*.
3. Retener todos los bloques de cabecera SOAP que apuntaban al nodo que envía el mensaje, que fueron ignorados durante el procesamiento y que son *reenviables*.

Los intermediarios de envío SOAP deben seguir las especificaciones que hay para las características de envío SOAP que están siendo usadas. En estas especificaciones, se incluyen las reglas que dicen como se construye el mensaje que hay que enviar. Estas reglas también pueden describir el emplazamiento de los bloques de cabecera que son insertados o reinsertados en el mensaje.

Este proceso tan complejo se repite en cada uno de los nodos intermediarios SOAP que forman parte del camino del mensaje SOAP que debe seguir el mensaje para viajar desde el nodo inicial SOAP hasta el nodo final SOAP.

Intermediarios Activos SOAP

Además del procesamiento llevado a cabo por los intermediarios de envío SOAP, los *intermediarios activos SOAP* emprenden un procesamiento adicional que puede modificar el mensaje SOAP de salida en aspectos no descritos. El conjunto potencial de servicios provisto por un intermediario activo SOAP incluye (con posibilidad de ampliación): seguridad en servicios, anotación de servicios, y manipulación del contenido de los servicios.

Los resultados de todos los procesamientos activos, pueden afectar a la interpretación que realizarán los siguientes nodos intermediarios SOAP de los mensajes SOAP que reciban, ya que estos mensajes tienen que pasar por múltiples intermediarios hasta llegar al nodo final. Es muy recomendable que las características SOAP que proveen los intermediarios activos SOAP sean descritas de alguna manera, para que las modificaciones que estos realicen sobre los mensajes SOAP de salida, sean detectadas por los nodos SOAP del camino del mensaje.

3.2.3.8. Modelo de Versiones SOAP

Un nodo SOAP determina si soporta la versión de un mensaje SOAP, donde soportar significa comprender la semántica de la versión del sobre SOAP. Esta información acerca de la versión solamente informa de la versión del sobre SOAP, y no ofrece información de cual pueda ser la versión de cualquiera de los otros componentes del mensaje SOAP.

Un nodo SOAP puede soportar múltiples versiones del nodo SOAP, sin embargo, cuando esté procesando un mensaje, el nodo SOAP deberá usar la semántica que defina la versión del mensaje.

En el caso de que un nodo SOAP reciba un mensaje cuya versión no soporte, con el valor del atributo *Value* del elemento *Code* establecido a “*env:VersionMismatch*” y presentando también algún otro defecto, el valor de este atributo será “*env:Sender*”, indicando que el fallo que se genera se debe al emisor del mensaje SOAP.

3.2.4. Modelo de Extensibilidad de SOAP

SOAP provee un sencillo marco para los mensajes, pero cuyo núcleo está pensado para ser extensible de forma que se puedan agregar nuevas funcionalidades

3.2.4.1. Características SOAP

Se considera que una característica SOAP (*SOAP feature*) es una extensión del marco de trabajo de mensajes SOAP. SOAP no tiene restricciones sobre el alcance de estas características, que pueden incluir, por ejemplo, “fiabilidad”, “seguridad”, “correlación”, “enrutamiento”, y protocolos de intercambio de mensajes o MEPs como lo son los patrones de petición/respuesta (*request/response*), de un solo camino (*one-way*) y conversaciones punto a punto.

El modelo de extensibilidad de SOAP provee dos mecanismos a través de los cuales se pueden expresar estas características: el *modelo de procesamiento SOAP* y el *marco de asociaciones de protocolo SOAP*. El primero describe el comportamiento de un único nodo SOAP con respecto al procesamiento de un único mensaje SOAP. El segundo de los mecanismos, se encarga de arbitrar la acción del envío y la recepción de los mensajes SOAP por un nodo SOAP a través los protocolos subyacentes.

El modelo de procesamiento SOAP, habilita a los nodos SOAP para incluir los mecanismos necesarios para que implementen una o más características, y que puedan expresarlas en los sobres (*envelope*) y los bloques de cabecera SOAP, a fin de que los bloques de cabecera sean inteligibles por todos los nodos SOAP pertenecientes al camino del mensaje SOAP. La sintaxis combinada y la semántica de los bloques de cabecera SOAP se conocen como *módulo SOAP*, y estos son especificados de acuerdo a las reglas dictadas en la Sección 3.2.4.3.

Por contrario, una vinculación de protocolo SOAP opera entre dos nodos SOAP adyacentes del camino del mensaje SOAP. No hay ningún requisito acerca de que entre los distintos saltos que realiza el mensaje en el camino del mensaje SOAP deba usarse el mismo protocolo subyacente. En algunos casos, estos protocolos subyacentes ya están provistos (ya sea por que sean así o mediante extensiones) de los

mecanismos necesarios para ofrecer determinadas características. El marco de vinculación del protocolo SOAP provee un esquema para la descripción de estas características, y de cómo se relacionan con los nodos SOAP a través de una especificación de vinculación.

La especificación de la característica debe incluir lo siguiente:

1. Un URI para nombrar la característica, lo que permite que la característica sea referenciada sin ambigüedad en los lenguajes de descripción durante el proceso de negociación.
2. La información necesaria en cada nodo para implementar dicha característica.
3. El procesamiento requerido en cada nodo para cumplir las obligaciones de la característica, incluyendo la captura de fallos de comunicación que puedan sucederse en los protocolos subyacentes.
4. La información que debe transmitirse de un nodo a otro.

3.2.4.2. Patrones de Intercambio de Mensajes SOAP (MEPs)

Un Patrón de Intercambio de Mensajes (MEP) es una plantilla que establece un patrón para el intercambio de mensajes entre nodos SOAP. Se puede clasificar a los MEPs como un tipo de característica, en los términos en los que antes estas fueron descritas.

La especificación de un patrón de intercambio de mensajes debe:

- Como el MEP es una característica de SOAP, debe proveer un URI que de un nombre al MEP.
- Describir el ciclo de vida del intercambio de un mensaje de acuerdo con el patrón.
- Describir las relaciones temporales/causales, si las hay, de los múltiples mensajes intercambiados de acuerdo con el patrón (es decir, un mensaje de respuesta siempre seguirá a uno de petición).

- Describir lo que sería una terminación normal o anormal para el intercambio de un mensaje, de acuerdo con el patrón.

Además, la especificación de un MEP debe incluir:

- Los requisitos necesarios para generar mensajes adicionales (como respuesta a una petición en un MEP petición/respuesta).
- Reglas para realizar la entrega u otra disposición de los fallos SOAP generados durante la operación del MEP.

3.2.4.3. Módulos SOAP

Se refiere a la especificación de la sintaxis y la semántica de uno o más bloques de cabecera SOAP. Un módulo SOAP implementa cero más características SOAP. Una especificación de módulo debe respetar las siguientes reglas:

1. Debe identificarse mediante un URI. Esto permite al módulo ser identificado sin ambigüedad en los lenguajes de descripción durante la negociación.
2. Debe declarar las características que provee un módulo.
3. Debe especificar de forma clara y completa la semántica y el contenido de los bloques de cabecera SOAP utilizados para implementar el comportamiento en cuestión, incluyendo, si fuera apropiado, las modificaciones sobre el modelo de procesamiento SOAP. El modelo de extensibilidad SOAP no especifica el límite al que SOAP puede ser extendido.
4. Debe indicar con claridad cualquier interacción conocida con el cuerpo SOAP o los cambios sobre la interpretación sobre el cuerpo SOAP que habría que considerar. Más lejos incluso, debe especificar claramente las interacciones conocidas con otros módulos y características SOAP, o los cambios en la interpretación que habría que considerar sobre estos. Por ejemplo, supongamos que tuviéramos un módulo que cifrará y eliminará el cuerpo SOAP y que insertara bloques de cabecera SOAP que contengan una suma de comprobación y una indicación del

mecanismo de cifrado utilizado. La especificación para el módulo debería indicar que el algoritmo de descifrado en el receptor debe ser pasado en primer lugar sobre los módulos que dependan del contenido del cuerpo SOAP.

3.2.5. Marco de las Vinculaciones de Protocolo SOAP

SOAP permite el intercambio de mensajes utilizando una gran variedad de protocolos subyacentes. El conjunto formal de reglas para llevar un mensaje SOAP a través o sobre otro protocolo (protocolo subyacente) con el propósito de intercambiarlo se llama *vinculación (binding)*. El Marco de Vinculaciones de Protocolo SOAP provee de una serie de reglas generales para la especificación de asociaciones del protocolo; de la misma manera, este marco de trabajo también describe la relación entre las asociaciones y los nodos SOAP que implementan esas asociaciones. Se pueden crear asociaciones adicionales mediante especificaciones que estén de acuerdo con el marco de trabajo de las asociaciones.

Una especificación de vinculación SOAP:

- Declara las características provistas por la vinculación.
- Describe cómo los servicios del protocolo subyacente son usados para la transmisión de los conjuntos de información del mensaje SOAP.
- Describe cómo los servicios de los protocolos subyacentes son usados para cumplir los contratos creados por las características soportadas por esa vinculación.
- Describe la captura de todos los fallos potenciales que pueden ser anticipados desde la vinculación.
- Define los requisitos para la construcción de una implementación que se adapte a la vinculación que esta siendo especificada.

Una vinculación no provee un modelo de procesamiento separado, y tampoco constituye un nodo SOAP por sí mismo. Pero una vinculación SOAP es una parte integral de un nodo SOAP.

3.2.5.1. Objetivos del Marco de Vinculación

Los objetivos del marco de vinculación son:

1. Presentar los conceptos y requisitos comunes a todas las especificaciones de vinculación.
2. Facilitar la descripción homogénea en situaciones donde múltiples asociaciones soportan características comunes, promoviendo la reutilización a través de las vinculaciones.
3. Facilitar la consistencia en la especificación de características opcionales.

Dos o más asociaciones podrían ofrecer una característica opcional, cómo podría ser el envío fiable, pero utilizando distintos medios para conseguirlo. La primera de las asociaciones podría llevarse a cabo mediante algún protocolo subyacente que directamente ofrezca esta característica, mientras que la segunda podría ofrecer esta característica implementando ella misma la lógica necesaria. En ambos casos, esta característica podría estar disponible para las aplicaciones de una forma consistente, sin importar cual de las asociaciones usemos.

3.2.5.2. Marco de Vinculación

La creación, transmisión, y procesamiento de un mensaje SOAP, posiblemente a través de uno o más intermediarios, se especifica mediante una máquina de estados distribuida. El estado consiste en la información conocida de un nodo SOAP en un instante de tiempo determinado, incluyendo, entre otras cosas, los contenidos del mensaje que está siendo ensamblado para la transmisión, o que ha sido recibido para su procesamiento. El estado de cada nodo puede ser actualizado tanto por un procesamiento local, como por la información recibida desde un nodo adyacente.

La máquina de estados distribuida que gestiona la transmisión de un mensaje SOAP dado a través de su camino, es la combinación del núcleo del procesamiento SOAP que está operando en cada nodo, en conjunción con las especificaciones de las asociaciones que conectan a cada par de

nodos. Una especificación de vinculación debe habilitar uno o más MEPs.

Cuando una especificación de vinculación soporta múltiples características, las especificaciones de las características deben incluir la información necesaria para poder usarlas correctamente en conjunción. De la misma forma, las dependencias que existan entre ellas, también deben ser especificadas. El marco de vinculación no provee ningún mecanismo implícito que permita controlar el uso de estas características independientes.

El marco de vinculación no provee un modo fijo de nombrar o escribir la información que comprende el estado de un nodo dado. Las características individuales y las especificaciones de asociaciones son libres de adoptar sus propias convenciones para representar los estados. Sin embargo, la consistencia a través de las vinculaciones y las características, es posible que sea mejorada en situaciones donde múltiples especificaciones de características adopten convenciones consistentes para la representación de estos estados.

3.2.6. Construcción de un Mensaje SOAP

Los mensajes SOAP enviados por un nodo inicial SOAP, no deben contener información sobre instrucciones de procesamiento. Los intermediarios SOAP tampoco insertarán información de este tipo en los mensajes que retransmitan. Los receptores SOAP que reciban un mensaje SOAP con información sobre instrucciones de procesamiento, deben generar un fallo SOAP con el atributo *Value* del elemento *Code* a “*env:Sender*”. Sin embargo, en los casos en los que las consideraciones de rendimiento realizadas por un intermediario SOAP, determinen la que detección de este tipo de información en mensajes que debe ser reenviado es poco factible, el intermediario dejará esa información de instrucciones de procesamiento en el mensaje reenviado sin modificarla, sin generar en este caso ningún fallo.

3.2.6.1 Sobre SOAP (*SOAP Envelope*)

El elemento informativo de *SOAP Envelope* tiene:

- El nombre local de *Envelope*.

- El espacio de nombres “*http://www.w3.org/2003/05/soap-envelope*”.
- Cero o más espacios de nombres de atributos cualificados de información entre sus propiedades o atributos.
- Uno o dos elementos de información en sus propiedades o elementos hijos de acuerdo a:
 - Un elemento opcional, *Header*.
 - Un elemento obligatorio *Body*.

3.2.6.2. Cabecera SOAP

El elemento de información *Header* de SOAP, provee de un mecanismo para extender un mensaje SOAP de manera descentralizada y modular. Este elemento tiene:

- El nombre local *Header*.
- El espacio de nombres “*http://www.w3.org/2003/05/soap-envelope*”.
- Cero o más espacios de nombres de atributos cualificados de información entre sus propiedades o atributos.
- Cero o más espacios de nombres de elementos cualificados de información entre sus propiedades o hijos.

Cada uno de estos elementos de información hijos de la cabecera SOAP, reciben el nombre de bloque de cabecera SOAP.

Bloque de Cabecera SOAP

Cada bloque de cabecera SOAP:

- Deberá tener una propiedad que tendrá un valor, que será el nombre de un elemento, que será un espacio de nombres cualificado.
- Podrá tener cualquier número de elementos de información hijos. Tales elementos, puede ser espacios de nombres cualificados.
- Puede tener cero o más atributos de información entre sus propiedades. Algunos de ellos o incluso todos, pueden ser de algunos tipos específicos. Estos tipos tienen un significado especial a la hora de realizar el procesamiento SOAP. Estos son:
 - Atributo *encodingStyle*.
 - Atributo *role*.
 - Atributo *mustUnderstand*.
 - Atributo *relay*.

Atributo *role* de SOAP

Este atributo se utiliza para indicar a que nodo SOAP en particular está apuntando el bloque de cabecera SOAP. El tipo del atributo *role* es “*xs:anyURI*”, y su valor será un URI que el nodo SOAP podrá asumir.

Omitir este atributo, es equivalente a sustituirlo por el valor “*http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver*”. Los emisores SOAP no deberían generar este atributo, pero los receptores a los que les llegue un mensaje que contenga este atributo, deberán aceptarlo si este tiene el valor “*http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver*”. En caso de que el nodo SOAP sea un intermediario que tenga que reenviar el mensaje SOAP, este puede omitir el atributo *role* de SOAP si su valor es el antes citado.

Un receptor SOAP deberá ignorar el atributo si este aparece en algún nodo descendiente de los bloques de cabecera SOAP o en alguno de los elementos hijos (o descendientes de estos) del cuerpo SOAP, ya que el emisor SOAP, tiene la obligación de situar este atributo en los bloques de cabecera SOAP.

Atributo *mustUnderstand* de SOAP

Este atributo se utiliza para indicar cuándo el procesamiento de un bloque de cabecera SOAP es obligatorio u opcional. El tipo de este atributo es “*xs:boolean*”.

En caso de que se omita, se definirá automáticamente tomando el valor *false*. Los emisores SOAP no tienen por que generarlo, pero los receptores SOAP, deberán aceptarlo si este toma el valor *false* o 0 (que es equivalente).

Un emisor SOAP que genere un mensaje SOAP, incluirá este atributo solamente en los bloques de cabecera SOAP. Un receptor SOAP deberá ignorar el atributo si aparece en algún nodo descendiente de los bloques de cabecera SOAP o en alguno de los elementos hijos (o descendientes de estos) del cuerpo SOAP.

Atributo *relay* de SOAP

Utilizado para indicar que si un bloque de cabecera SOAP no es procesado, debe ser reenviado. Como en el atributo *mustUnderstand*, el tipo es también “*xs:boolean*”.

Si se omitiese este atributo, sería como si semánticamente se pusiera el valor *false*. Los emisores SOAP no tienen por que generarlo, pero los receptores SOAP, deberán aceptarlo si este toma el valor *false* o 0.

Al igual que en los dos casos anteriores, un emisor SOAP que genere un mensaje SOAP, incluirá este atributo solamente en los bloques de cabecera SOAP. Y de la misma forma, un receptor SOAP deberá ignorar el atributo si este aparece en algún nodo descendiente de los bloques de cabecera SOAP o en alguno de los elementos hijos (o descendientes de estos) del cuerpo SOAP.

3.2.6.3. Cuerpo SOAP

Este elemento provee de un mecanismo, mediante el cual podemos transmitir información a un receptor SOAP final.

El elemento de información *Body* tiene:

- El nombre local *Body*.
- El espacio de nombres “*http://www.w3.org/2003/05/soap-envelope*”.
- Cero o más atributos de información de espacios de nombres cualificados entre sus propiedades.
- Cero o más elementos de información de espacios de nombres cualificados entre sus elementos hijos.

Elemento hijo *Body* de SOAP

Todos los elementos de información hijos del elemento *Body* de SOAP deberán seguir unas reglas similares a las que se establecieron para los bloques de cabecera SOAP, aunque no se profundizará hasta ese nivel.

3.2.6.4. Fallo SOAP o SOAP *Fault*

Este elemento se usa para llevar información de error dentro de un mensaje SOAP. El elemento *Fault* de SOAP tiene:

- El nombre local *Fault*.
- El espacio de nombres “*http://www.w3.org/2003/05/soap-envelope*”.
- Dos o más elementos hijos de información entre sus propiedades, que pueden ser de los siguientes tipos:
 1. Un elemento *Code*, que es obligatorio.
 2. Un elemento *Reason*, que es obligatorio. Cuyo fin es el de proveer una explicación inteligible por los humanos acerca de la razón por la que se produjo el fallo.
 3. Un elemento *Node*, que es opcional. Cuyo fin es proveer información sobre el nodo SOAP del camino del mensaje

SOAP en el que ocurrió el fallo. Este elemento consiste en un URI que identifica al nodo que generó el fallo.

4. Un elemento *Role*, que es opcional. Este elemento contiene el rol que estaba desempeñando el nodo SOAP que generó el fallo. El valor de este elemento debe ser uno de los roles asumidos por el nodo en el momento del procesamiento del mensaje.
5. Un elemento *Detail*, que es opcional. Este elemento está pensado para llevar información sobre el error.

Para poder distinguir que un mensaje SOAP lleva en si información de error SOAP, este deberá contener un elemento *Fault* SOAP como único hijo del elemento de información *Body* de SOAP. Si además del elemento SOAP, incluyéramos otros elementos de información, no existiría una semántica SOAP definida para tal situación, por lo que esta situación no debe darse.

Aunque tampoco hay una semántica definida para cuando el elemento *Fault* se encuentra dentro de un bloque de cabecera SOAP, o como único hijo del elemento *Body* de SOAP, no se considerarán éstas como situaciones anómalas.

Códigos de Fallo SOAP

En la Tabla 3 podemos ver los códigos de los fallos SOAP:

Nombre Local	Meaning
VersionMismatch	El nodo que genera el fallo ha encontrado un elemento desconocido en el lugar del elemento <i>Envelope</i> .
MustUnderstand	Un elemento hijo del elemento <i>Header</i> de SOAP que apunta al nodo que generó el fallo, tiene un atributo <i>mustUnderstand</i> con el valor a <i>true</i> , pero no puede entenderlo.
DataEncodingUnknown	Un bloque de cabecera o algún elemento hijo del elemento <i>Body</i> de SOAP que apunten al nodo que generó el fallo, están codificados mediante una codificación de datos que este nodo no soporta.
Sender	El mensaje no está correctamente formado o no contiene la información necesaria para llegar a su destino con éxito.
Receiver	Este error ocurre cuando el fallo es debido más al procesamiento del mensaje que al contenido del mismo.

Tabla 3. Códigos de fallos SOAP.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.example.org/timeouts"
  xmlns:xm1="http://www.w3.org/XML/1998/namespace">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>m:MessageTimeout</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">Sender Timeout</env:Text>
      </env:Reason>
      <env:Detail>
        <m:MaxTime>P5M</m:MaxTime>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Figura 30. Ejemplo de mensaje SOAP de error.

3.2.7. Consideraciones sobre la Seguridad

El marco de mensajes SOAP no provee directamente de ningún mecanismo para realizar transacciones con control de acceso, confidencialidad, integridad y sin rechazo. Hay una forma de proveer de estas medidas, y es mediante extensiones SOAP, usando el modelo de extensibilidad de SOAP.

Los desarrolladores SOAP deben anticiparse a las aplicaciones SOAP maliciosas que puedan mandar datos de forma malintencionada a nodos SOAP. Es muy recomendado que un nodo SOAP que reciba un mensaje SOAP sea capaz de evaluar tanto el nivel de confianza que le merece el nodo que lo envió, como el propio contenido del mensaje.

3.2.7.1. Nodos SOAP

SOAP puede transportar datos definidos por una aplicación como bloques de cabecera SOAP o como contenidos del cuerpo SOAP. El procesamiento de un bloque de cabecera SOAP puede conllevar transacciones con diferentes efectos, como cambios en el estado del nodo, información de registro, o la generación de mensajes adicionales. Por todo esto, se recomienda que para cualquier escenario, un nodo SOAP sólo procese los bloques de cabecera SOAP que tengan bien

definidas las implicaciones de seguridad por su procesamiento por un nodo SOAP.

De la misma manera, si un nodo procesa el contenido de un cuerpo SOAP cuyas implicaciones de seguridad no están bien definidas, los efectos para el nodo SOAP receptor pueden muy graves.

En cualquier caso, las consideraciones sobre la seguridad no están solamente limitadas al reconocimiento de los elementos hijos inmediatos de los bloques de cabecera y del cuerpo SOAP, si no a todos los datos contenidos en un mensaje SOAP que puedan acarrear la ejecución remota de alguna acción en el entorno el receptor. Aquí también incluimos los datos binarios y las cadenas que representan, por ejemplo, un URI para realizar una consulta.

3.2.7.2. Intermediarios SOAP

En el modelo de procesamiento de SOAP, para que un mensaje SOAP viaje desde el nodo inicial hasta el nodo final, puede tener que atravesar múltiples nodos SOAP, los intermediarios. Los fallos de seguridad en los sistemas que utilizan intermediarios SOAP pueden provocar serios problemas de seguridad y privacidad. Si un intermediario SOAP no está comprometido con las directivas de seguridad y privacidad, ya sea por su implementación o configuración, puede convertirse en el objetivo de múltiples ataques potenciales.

Analizando las implicaciones de la seguridad y sus problemas relacionados en SOAP, nos damos cuenta que el alcance de los mecanismos de seguridad del protocolo subyacente puede no ser el mismo en todo el camino que debe recorrer el mensaje en su viaje desde el nodo SOAP inicial hasta el final, ya que en todos los saltos que el mensaje realiza entre los intermediarios, no tienen por que usarse el mismo protocolo subyacente. Al utilizar intermediarios, es posible que nos estemos escapando del alcance del soporte de seguridad que nos ofrecen los protocolos del nivel de transporte.

3.2.7.3. Asociaciones del Protocolo Subyacente

Algunos protocolos pueden ser diseñados para algún propósito en particular o algún perfil de aplicación. Las asociaciones SOAP para estos

protocolos deberían usar los mismos identificadores finales (como podría ser el número de puerto TCP) que los que usa el protocolo subyacente. Esto se hace así a fin de aprovechar la infraestructura asociada a los protocolos.

Las definiciones para los protocolos subyacentes que cuenten con puertos por defecto ya conocidos (como en el caso de los puertos del protocolo HTTP), deberían documentar las interacciones potenciales con esos puertos de acuerdo a los perfiles de las aplicaciones. Las definiciones de vinculación deberían también ilustrar el uso de la misma sobre un puerto que no pertenezca a los puertos por defecto como medio para eludir la interacción no intencionada con cada servicio.

3.2.8. Modelo de Datos SOAP

El Modelo de Datos de SOAP permite representar directamente las estructuras y valores de datos definidos en las aplicaciones como un grafo cuyos nodos son etiquetas.

El propósito del Modelo de Datos SOAP es permitir la representación de datos que no están basados en XML. El uso de este modelo de datos, es muy recomendable en algunas ocasiones.

3.2.8.1. Aristas del Grafo

Las aristas de un grafo tienen su origen en un nodo y su fin en otro nodo del grafo. Una arista puede originarse y finalizar en el mismo nodo del grafo. Una arista que sale de un nodo del grafo es una arista de salida, si la arista finaliza en un nodo del grafo, es una arista de llegada. También puede darse el caso de que una arista solamente salga de un nodo del grafo, es decir, es una arista que solo va hacia el exterior, y también puede finalizar solamente en un nodo del grafo, con lo que es una arista que sólo llega desde el exterior.

Las aristas de sólo salida al exterior del grafo, pueden ser distinguidas mediante una etiqueta o por la posición. La posición constituye un orden total para todas las aristas; de esta forma, si una arista de salida de un nodo se distingue por el orden, entonces todas las aristas de salida de ese nodo se distinguen también por el orden.

Etiquetas de arista

Una etiqueta de arista es un nombre XML cualificado. Dos etiquetas de arista son iguales si y sólo si los nombres XML expandidos son iguales.

3.2.8.2. Nodos del Grafo

Un nodo de un grafo tiene cero o más aristas de salida. Un nodo de un grafo que no tiene aristas de salida tiene un valor léxico opcional. Todos los grafos tienen un tipo de nombre opcional del tipo ‘*xs:QName*’ del espacio de nombres “*http://www.w3.org/2001/XMLSchema*”.

Nodos referenciados una y múltiples veces

Un nodo de un grafo puede ser referenciado una vez o múltiples veces. Un nodo referenciado una única vez tendrá solamente una arista de llegada, mientras que un nodo referenciado múltiples veces tendrá varias aristas de llegada, tantas como veces referenciado esté el nodo del grafo.

3.2.8.3. Valores

Mediante un nodo con un valor léxico, representamos un único valor. Un valor compuesto se representa como un nodo del grafo con cero o más aristas de salida de la siguiente manera:

1. Un nodo del grafo cuyas aristas se distinguen solamente por sus etiquetas, se conoce como una *estructura* o “*struct*”. Las aristas de salida de una estructura deben ser etiquetadas con nombres distintos.
2. Un nodo del grafo cuyas aristas de salida se distinguen solamente por posición, se conoce como un “*array*”. Las aristas de salida de un array no deben ser etiquetadas.

3.2.8.4. Ejemplo

A continuación, se desarrollará un pequeño ejemplo a fin de ilustrar el funcionamiento del modelo de datos de SOAP, mediante el cual representamos la información del mensaje SOAP como un grafo.

A fin de ilustrar de forma completa el modelo de datos, se ha seleccionado un ejemplo complejo de mensaje SOAP [2], que podemos ver en la Figura 31. Y el grafo generado a partir de este mensaje SOAP se puede observar en la Figura 32.

```
<product xsi:type="n2:Product">
  <download xsi:type="xsd:string">
    http://www.rubylinks.de/download/ruby-spread-0.1.tar.gz
  </download>
  <homepage xsi:type="xsd:string">http://www.rubylinks.de/</homepage>
  <status xsi:type="xsd:string">just started, first alpha release</status>
  <description xsi:type="xsd:string">Ruby/Spread is an interface to
the Spread Library at http://www.spread.org/. Spread is a toolkit and daemon that
provides multicast and group communications support to
applications across local and wide area networks. Spread is
designed to make it easy to write groupware, networked
multimedia, reliable server, and collaborative work
applications.
  </description>
  <version xsi:type="xsd:string">0.1</version>
  <license xsi:type="xsd:string">LGPL</license>
  <name xsi:type="xsd:string">Ruby/Spread</name>
</product>
```

Figura 31. Fragmento de un mensaje SOAP

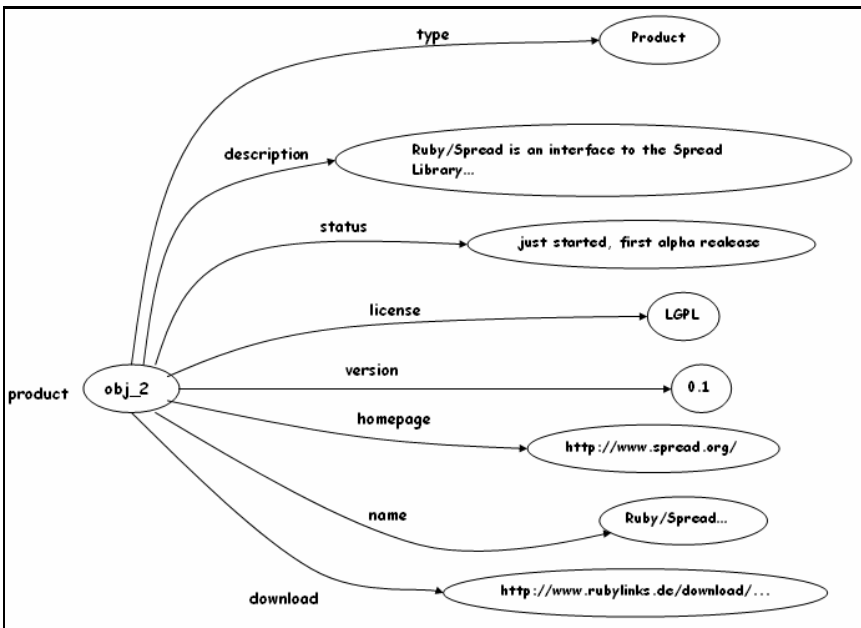


Figura 32. Grafo del modelo de datos correspondiente al fragmento de la Figura 31

Como podemos observar en la Figura 32, hemos extraído el grafo correspondiente al nodo *return* del mensaje SOAP de la Figura 31. En el grafo se ven representados como nodos hijos del *obj_1* (que representa al nodo *return*) a diferentes nodos cuyos arcos denotan al elemento hijo en cuestión. Nótese como la propiedad *type* del nodo *result* también constituye un nodo hijo.

La realización del grafo completo se llevaría a cabo mediante un análisis recursivo de cada uno de los nodos hijos, mediante el cual, completaríamos el grafo que podemos observar en la Figura 32.

3.2.9. Codificación SOAP

La Codificación SOAP provee un método para codificar instancias de los datos de acuerdo al modelo de datos descrito en el apartado anterior. Podemos usar esta codificación para transmitir los datos en los bloques de cabecera SOAP y/o en los cuerpos SOAP. Otros modelos de datos, alternan la codificación del modelo de datos SOAP además de datos sin codificar para los mensajes SOAP.

Las reglas para codificar están identificadas por la URI “<http://www.w3.org/2003/05/soap-encoding>”, por lo que los mensajes SOAP que usen este tipo de serialización deberán indicarlo mediante el atributo de información *encodingStyle*.

3.2.9.1. Equivalencia entre XML y el Modelo de Datos SOAP

La codificación SOAP define un reducido conjunto de reglas para codificar grafos como los descritos en la sección anterior. Las codificaciones son descritas bajo la perspectiva de un *des-serializador*. Supondremos la presencia de una serialización en XML y describiremos el mapeo al correspondiente grafo.

Para un grafo dado, es posible realizar más de una codificación correcta. Cuando serializamos un grafo para enviarlo mediante un mensaje SOAP, debe usarse luego una representación que *traduzca* la información transmitida al mismo grafo que se envió al principio en el mensaje. En el caso que hemos dicho en el que existen múltiples representaciones distintas, podremos usar cualquiera de ellas.

Codificación de Aristas y Nodos del Grafo

Cada arista del grafo es codificada como un elemento de información, y cada elemento de información representa una arista del grafo.

El nodo del grafo en el que finaliza una arista se determina de la siguiente manera:

1. Si el elemento de información que representa la arista no tiene un atributo de información *ref* entre sus atributos, entonces se dice que ese elemento de información representa un nodo en el grafo, y la arista termina en ese nodo. En estos casos, el elemento de información representa una arista del grafo y un nodo del grafo.
2. Si el elemento de información que representa la arista tiene un atributo de información *ref* entre sus atributos, entonces, el valor de ese atributo de información debe ser idéntico al valor del atributo de información *id* del mismo sobre. En este caso, la arista termina en el nodo del grafo representado por el elemento de información que aparece en el atributo de información *id*.

Todos los nodos en el grafo se codifican como se describe en el punto 1. Las aristas que finalizan en un nodo, para los nodos con referencias múltiples, se codifican como se describe en el punto 2.

Codificación de Valores Simples

El valor léxico de un nodo del grafo, representa un valor simple como una secuencia de caracteres en Unicode. El elemento de información que representa el valor simple puede tener entre sus atributos un atributo de información “*nodeType*”. Hay que tener en cuenta que los caracteres Unicode no pueden representarse en XML.

Codificación de Valores Compuestos

Una arista que sale de un nodo de un grafo se codifica como un elemento de información hijo del elemento de información que representa al nodo. Hay una serie de reglas particulares que se aplicarán dependiendo del tipo de valor compuesto que represente el nodo del atributo:

1. Para una arista de un grafo que se distingue de las demás mediante una etiqueta, se utilizará un *nombre local* y una serie de propiedades del elemento de información hijo, que determinan el valor de la etiqueta de la arista.
2. Para una arista de un grafo que se distingue por posición:
 - La posición ordinal de la arista corresponde a la posición del elemento de información hijo relativa a la de sus hermanos.
 - Las propiedades de nombre local y espacios de nombre del elemento de información hijo no son significativas en este caso.
3. Es posible que el elemento de información que representa un nodo con un valor compuesto tenga entre sus propiedades un atributo de información "*itemType*"
4. Para la codificación de un nodo de un grafo que representa un "*array*" se aplican las siguientes reglas:
 - El elemento de información que representa un *array* puede tener entre sus atributos, un atributo de información llamado "*itemType*".
 - El elemento de información que representa un *array* puede tener entre sus atributos, un atributo de información llamado "*arraySize*".
5. Si la arista del grafo no termina en un nodo de grafo, entonces puede ser omitida de la serialización o codificada como un elemento de información con un atributo de información del tipo "*xsi:nil*" con el valor a *true*.

3.2.9.2. Fallos en la Decodificación

Durante la recepción, un nodo receptor SOAP:

- Debe generar un fallo SOAP del tipo “*env:Sender*” con el código “*enc:MissingID*” si el mensaje contiene un atributo de información *ref*, pero no su correspondiente atributo de información “*id*”.
- Debe generar un fallo SOAP del tipo “*env:Sender*” con el código “*enc:DuplicateID*” si el mensaje contiene dos o más atributos de información “*id*” con el mismo valor.
- Puede generar un fallo SOAP del tipo “*env:Sender*” con el código “*enc:UntypedValue*” si no se especifica la propiedad de tipo de nombre del nodo de grafo codificado.

3.2.10. Representación RPC de SOAP

Uno de los objetivos de diseño de SOAP es el de facilitar el intercambio de mensajes que *mapean*, de forma conveniente, los métodos y llamadas a procedimiento a definiciones e invocaciones. Para esto se definió una representación uniforme de las peticiones y las respuestas de las llamadas a procedimiento remoto o RPC. Esta representación es totalmente independiente de la plataforma.

El atributo de información SOAP “*encodingStyle*” se utiliza para indicar el estilo de codificación de la representación RPC. Este estilo de codificación debe soportar al Modelo de Datos SOAP visto anteriormente.

La Representación RPC de SOAP no está basada en ninguna vinculación a protocolo SOAP. Cuando SOAP se vincula a HTTP, una petición RPC se *mapea* como una petición HTTP, y la correspondiente respuesta RPC se *mapea* a una respuesta HTTP. La Representación RPC de SOAP se puede utilizar con otros protocolos que no son HTTP.

Para invocar una RPC, se necesita la siguiente información:

- La dirección del nodo SOAP destino.
- Un nombre de procedimiento o método.

- Las identidades y los valores de los argumentos que se pasan al procedimiento o al método. Los argumentos que se utilizan para localizar recursos Web, deben distinguirse de alguna forma de los que representan datos o información de control.
- Valores para alguna propiedad que se requieran por cualquier característica de la vinculación. Por ejemplo, los valores “GET” o “POST” para la propiedad “<http://www.w3.org/2003/05/soap/features/web-method/Method>”.
- Información opcional de cabecera.

RPC de SOAP depende de la vinculación con el protocolo para proveer un mecanismo para llevar la URI del nodo SOAP al que apunta. Para el protocolo HTTP, la URI de la petición indica el recurso sobre el que se realiza la invocación.

La Representación RPC de SOAP emplea el *Patrón de Intercambio de Mensajes basado en Petición-Respuesta* y el *Patrón de Intercambio de Mensajes SOAP basado en Respuesta*. A pesar de esto, la Representación RPC de SOAP se podría usar con otros patrones de intercambio de mensajes que no sean los citados en este párrafo.

3.2.10.1. Uso de RPC en la World Wide Web

Identificación de Recursos RPC

En la WWW, los recursos se identifican mediante URIs, pero en casi todos los lenguajes de programación, la información se identifica mediante los argumentos de los procedimientos, por ejemplo:

```
ConsultarExistencia(Identificador="ABC123")
```

En esta llamada, el recurso que se actualiza es el *ConsultarExistencia* al que le pasamos un *Identificador* “123”. Por esto, cuando mapeamos un método o una llamada a procedimiento remoto desde un lenguaje de programación, cualquier argumento que pueda servir para identificar un recurso (como en el ejemplo era el *Identificador*) debe ser incluido en la URI a la que se dirige el mensaje SOAP.

La especificación no indica ningún estándar en concreto para la codificación de los nombres de los métodos y los argumentos. No obstante, se puede concretar cómo construir la URI de petición mediante los lenguajes de definición e los SW.

3.2.10.2. RPC y el Cuerpo SOAP

Tanto las invocaciones como las respuestas RPC son transportadas en el elemento *Body* del mensaje SOAP mediante la siguiente representación:

Invocación RPC

La invocación RPC se modela de la siguiente manera:

- La invocación se representa mediante una única estructura que contiene una arista (representación en forma de grafo ya vista) de salida por cada parámetro de entrada o de entrada/salida. La estructura se nombra de la misma manera que el procedimiento o el método, usando también una serie de convenciones para representar nombres de métodos que no son nombres “legales” en XML.
- Cada arista de salida tiene una etiqueta que corresponde al nombre del parámetro al que representa, siguiendo también una serie de convenciones para representar parámetros que no son nombres “legales” en XML.

Las aplicaciones pueden procesar invocaciones con parámetros erróneos, pero tan sólo producirán un fallo durante el procesamiento, devolviendo el correspondiente fallo.

Respuesta RPC

Una Respuesta RPC se modela de la siguiente manera:

- La respuesta es representada mediante una única estructura que contiene una arista de salida para representar el valor de retorno y cada uno de los valores de los parámetros de salida y de entrada/salida. El nombre que se le da a la estructura no es significativo.

- Cada parámetro se representa mediante una arista de salida con una etiqueta que corresponde al nombre del parámetro. Al igual que en la invocación RPC, también seguiremos las mismas convenciones para nombrar nombres de parámetros que no son nombres “legales” en XML.
- Un valor de retorno no nulo se representa de la siguiente manera:
 1. Debe haber una arista de salida con el nombre local “*result*” y con el nombre del espacio de nombres “*http://www.w3.org/2003/05/soap-rpc*”. Esta arista terminará en un nodo terminal.
 2. El tipo de este nodo terminal es “*xs:QName*”, y su valor es el nombre de la arista de salida que termina en el valor de retorno actual.
- Los fallos en las invocaciones son capturados de acuerdo a las reglas definidas en la Sección 3.2.9.4. En caso de que la vinculación del protocolo también añada reglas adicionales para tratar esto, también habrá que seguirlas.

Restricciones en la Codificación SOAP

Cuando se usa la codificación SOAP junto con las convenciones de RPC descritas, el elemento *Body* de SOAP debe contener solamente un único elemento de información hijo, que será la invocación RPC o la estructura de respuesta serializadas.

3.2.10.3. RPC y las Cabeceras SOAP

En el sobre SOAP, se puede incluir información adicional acerca de la codificación de una invocación o respuesta SOAP, pero no se incluye como parte de la signatura de método o procedimiento que se invoca (en caso de que el mensaje SOAP sea de una invocación). Esta información adicional será expresada como bloques de cabecera SOAP.

3.2.10.4. Fallos RPC

La representación RPC de SOAP introduce sub-códigos de fallo SOAP adicionales para utilizarlos junto con los códigos de fallo SOAP (ya descritos en secciones anteriores).

Los errores que suceden durante una invocación RPC son informados de acuerdo a las siguientes reglas:

1. Se generará un fallo con el elemento hijo *Value* del elemento *Code* establecido al valor “*env:Receiver*” cuando el receptor no pueda almacenar el mensaje de manera temporal. Por ejemplo, esto podría ser ocasionado por falta de memoria en el momento en el que se recibió el mensaje.
2. Se deberá generar un fallo con el elemento hijo *Value* del elemento *Code* a “*env:DataEncodingUnknown*” cuando los argumentos sean codificados mediante una codificación desconocida para el receptor.
3. Se puede generar un fallo con el elemento hijo *Value* del elemento *Code* a “*env:Sender*” y el elemento hijo *Value* del elemento *Subcode* a “*rpc:ProcedureNotPresent*” cuando el receptor no soporte el procedimiento o el método especificado.
4. Se puede generar un fallo con el elemento hijo *Value* del elemento *Code* a “*env:Sender*” y el elemento hijo *Value* del elemento *Subcode* a “*rpc:BadArguments*” cuando el traductor no pueda analizar sintácticamente los argumentos, o cuando encuentre alguna discordancia entre el número o tipo de los argumentos de la llamada que recibió y los que esperaba.

3.3. UDDI (Universal Description, Discovery, and Integration)

3.3.1. Introducción

UDDI provee un método estandarizado para la publicación y el descubrimiento de información sobre los SW. Esta iniciativa parte de la industria, que pretendía crear una plataforma independiente, un marco de trabajo abierto para la descripción de servicios, descubrimiento de

organizaciones y la integración de servicios de negocio. UDDI se centra en el proceso de descubrimiento dentro de la arquitectura orientada a servicios.

Los SW se están convirtiendo en la base del comercio electrónico en muchos sentidos, y en este sentido, las organizaciones invocan los servicios creados por otras compañías para realizar sus transacciones de negocio.

Para descubrir y encontrar los SW que pudiéramos necesitar para nuestra organización, hay que utilizar algún medio que permita realizar la búsqueda y descubrimiento de los SW. Afrontar esta empresa por nosotros mismos se convertiría en una labor titánica y en la que seguramente nos dejaríamos sectores y compañías sin explorar.

UDDI es un único registro conceptual distribuido a lo largo de multitud de nodos que replican la información unos de otros. UDDI pretende ser la solución al descubrimiento y la integración de SW, que de otra forma, sería una labor inabordable.

UDDI constituye un registro en el que las empresas “*dan de alta*” los SW que desarrollan, a fin de que posibles usuarios interesados en hacer uso de esos servicios para realizar determinadas acciones en sus negocios, puedan encontrarlos sin demasiada dificultad.

Una organización puede registrar tres tipos de información dentro de un registro UDDI:

- *Páginas blancas.* Constituida por información básica de contacto e identificadores de la empresa, incluyendo el nombre de la empresa, dirección, información de contacto. Esta información permite a otros descubrir los SW de una empresa basándose en la identificación del negocio.
- *Páginas amarillas.* Consiste en información que describe al SW mediante diferentes categorizaciones o taxonomías. Esta información permite descubrir SW basándose en esta categorización.

- *Páginas Verdes*. Consiste en información técnica que describe el comportamiento y las funciones soportadas por un SW. Esta información incluye punteros que, entre otras cosas, indican dónde están ubicados los SW.

Cómo se usa UDDI

UDDI tiene diferentes usos dependiendo de la perspectiva del que lo use. Desde la perspectiva de un analista de negocios, UDDI será similar a un motor de búsqueda de Internet para procesos de negocio. El analista podrá echar una ojeada a uno o más registros UDDI para ver los diferentes negocios que exponen sus SW y las especificaciones de esos servicios (aunque es posible que una persona no desee mirar directamente en un registro UDDI, ya que el formato de la información posiblemente no esté en uno de los formatos más inteligibles para los usuarios humanos).

Para la interacción con los registros UDDI, existe una API que permite a los desarrolladores de software publicar los servicios en los registros y realizar consultas sobre los registros para descubrir servicios realizando búsquedas según los criterios necesarios. Se puede pensar que el propio software pueda realizar el proceso de descubrimiento de los SW dinámicamente y usarlo sin que ninguna persona tenga que mediar en el proceso.

Tanto los analistas de negocios como los desarrolladores de software pueden publicar nuevas entidades de negocio y servicios. Para el descubrimiento de estos, pueden usarse portales web que hacen de mediadores a la hora de acceder a un registro UDDI. Un portal puede realizar búsquedas sobre un registro o tener un carácter más general buscando en varios de estos registros, quedando patente en esto último la relación entre los negocios y la tecnología (Figura 33).

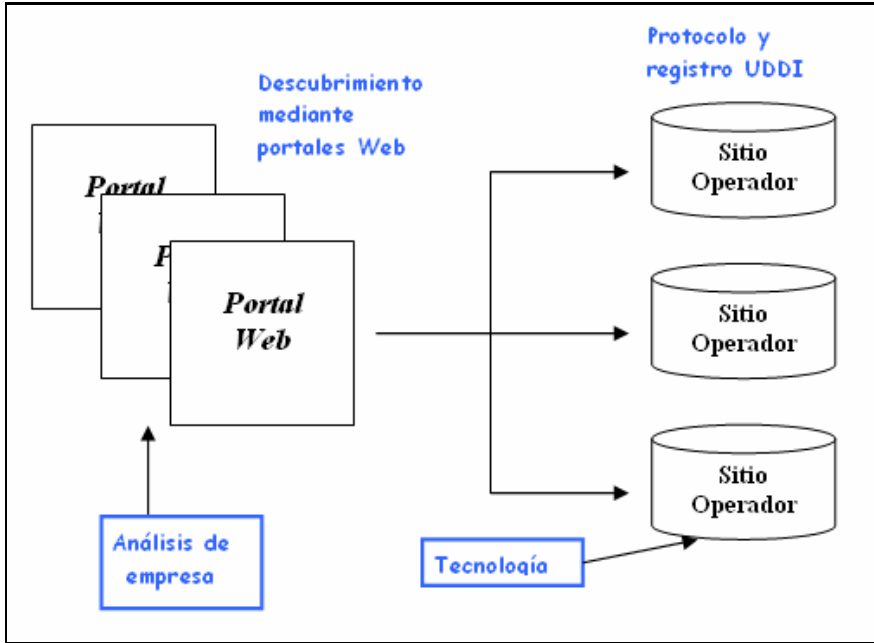


Figura 33. Relación entre empresas y tecnología.

Arquitectura

UDDI, conceptualmente es un único sistema constituido por múltiples nodos que tienen sus datos sincronizados mediante replicación. Existe una serie de nodos operadores que poseen, cada uno de ellos, una copia del contenido. A la agrupación global de nodos operadores se les conoce como *Registro de Negocio UDDI* (UBR). Los nodos operadores replican el contenido de unos a otros. Si accedemos a uno de los operadores, obtendremos la misma información y calidad de servicio que si realizáramos el acceso a cualquier otro nodo operador de la UBR. Cuando se inserta información en la UBR, en realidad se hace sobre uno de los nodos, y ese nodo operador se convierte en el poseedor principal de ese contenido. Cualquier tipo de manipulación de esa información deberá realizarse sobre el poseedor principal o nodo en el que se insertó.

De la misma forma, una compañía puede decidir crear un nodo operador propio y privado. Estos nodos privados no forman parte del mencionado UBR, por lo que los contenidos no tendrán por que ser los mismos, y el proceso de sincronización que decíamos que sucedía en el UBR no tiene consecuencia alguna en estos entornos privados. Un

conjunto de compañías podrían crear su propio registro con sus nodos operadores que tengan información replicada sobre sus contenidos, pero nunca serán parte del UBR, y las operaciones que se realicen en el UBR no tendrán por que tener efecto aquí.

3.3.2. Especificaciones de UDDI

El proyecto UDDI define un conjunto de definiciones en *XML Schema* que describen los formatos de datos utilizados por las distintas especificaciones de las APIs de programación. La versión actual de la especificación de UDDI es la 2.0. Esta especificación incluye:

- *Replicación UDDI.* En este documento se describe el proceso de replicación de los datos y las interfaces que un nodo operador debe asumir para realizar la replicación de datos entre nodos del registro. Esta especificación no consiste en una API, ya que sólo describe el proceso de replicación usado entre los nodos del UBR.
- *Operadores UDDI.* En este documento se esboza el comportamiento y los parámetros operacionales que requieren los nodos operadores UDDI. También se especifica los requisitos de gestión de los datos que deben respetar los nodos operadores. Al igual que la especificación anterior, este documento no supone una API de programación. Los nodos que formen parte de registros privados no tienen por que acogerse a esta especificación.
- *API para los Programadores UDDI.* Esta especificación define un conjunto de funciones que soportan todos los registros UDDI para la consulta sobre los servicios alojados en el registro y publicar información sobre un negocio o sobre un servicio en el registro. Esta especificación define un conjunto de mensajes SOAP que contienen a su vez una serie de documentos en XML que el registro es capaz de aceptar, analizar y responder.
- *Estructuras de Datos de UDDI.* Esta especificación trata sobre las estructuras en XML contenidas dentro de los mensajes SOAP definidos por la API para los Programadores UDDI.

El esquema de la API XML de UDDI no entra dentro de la especificación, pero está contenido como un documento *XML Schema*, que define la estructura y los tipos de datos de las estructuras de datos UDDI.

3.3.3. Programación UDDI

La especificación de UDDI describe dos APIs, la *API de pregunta* y la *API de publicación*. Las APIs difieren en los documentos en XML, las estructuras de datos, y los puntos de acceso.

La API de pregunta sirve para localizar información acerca de los negocios, los servicios que estos ofrecen, la especificación de esos servicios, e información sobre que hacer cuando se produce una situación de fallo. Con esta API, cualquier operación de lectura de un registro UDDI se hace con sólo un mensaje de esta API. Esta API no requiere acceso autenticado, razón por la que los accesos se realizan mediante el protocolo de transporte HTTP.

La API de publicación se usa para crear, almacenar, o actualizar información ubicada en un registro UDDI. Al contrario que ocurre en la API de pregunta, en esta API, todas las funciones de las que dispone requieren que el acceso a un registro UDDI sea autenticado. El registro UDDI debe tener una identidad para la entrada al sistema, y las credenciales de seguridad para esa identidad deben pasarse como parámetros en cada invocación a UDDI en el documento XML. Debido a que esta API requiere un acceso autenticado, el protocolo de transporte a través del cual se accede es HTTPS, pero utilizando una URL distinta a la utilizada para el punto de acceso en las preguntas.

En la Tabla 4, podemos ver las URLs de los puntos de acceso en las dos APIs y para los principales nodos operadores:

Todas las operaciones que un registro UDDI realiza son síncronas, por lo que el cliente que realice una petición se bloqueará hasta que reciba el mensaje de respuesta. Todas las operaciones tienen un sencillo mecanismo petición/respuesta, lo que podemos representar el estado de estas operaciones sin necesidad de usar estados.

Nodo Operador	URL de Pregunta	URL de Publicación
HP	http://uddi.hp.com/inquire	https://uddi.hp.com/publish
IBM Production	http://www-3.ibm.com/services/uddi/inquiryapi	https://www-3.ibm.com/services/uddi/protect/publishapi
IBM Test	http://www-3.ibm.com/services/uddi/testregistry/inquiryapi	https://www-3.ibm.com/services/uddi/testregistry/protect/publishapi
Microsoft Production	http://uddi.microsoft.com/inquire	https://uddi.microsoft.com/publish
Microsoft Test	http://test.uddi.microsoft.com/inquire	https://test.uddi.microsoft.com/publish
SAP Test	http://udditest.sap.com/UDDI/api/inquiry/	https://udditest.sap.com/UDDI/api/publish/
Systinet	http://www.systinet.com/wasp/uddi/inquiry/	https://www.systinet.com/wasp/uddi/publishing/

Tabla 4. URLs de puntos de acceso de los principales nodos operadores.

3.3.3.1. Estructuras de datos UDDI

En este apartado, veremos las principales estructuras de datos que se pasan en los mensajes, en unos casos como parámetro de entrada, en otros como parámetro de salida. En la Figura 34 podemos ver las principales estructuras de datos de UDDI y las relaciones existentes entre ellas.

Desarrollando el contenido de la Figura 34, una estructura *<businessEntity>* representa la información básica de un negocio. Esta consiste en información de contacto, una categorización, identificadores, descripciones y relaciones con otros negocios. El registro UDDI permite a las compañías establecer distintos tipos de relaciones con otras compañías.

La estructura *<publisherAssertion>* se utiliza para establecer relaciones públicas entre dos estructuras tipo *<businessEntity>*. Diremos que una relación entre dos estructuras *<businessEntity>* es pública o visible cuando ambas entidades creen la misma relación mediante dos documentos *<publisherAssertion>* independientes. De esta forma, una compañía puede atribuirse una relación de negocio sólo si la otra compañía establece también la relación. Una relación entre dos compañías no será visible o no pública hasta que la otra compañía cree su propio documento *<publisherAssertion>* para su propia estructura *<businessEntity>*.

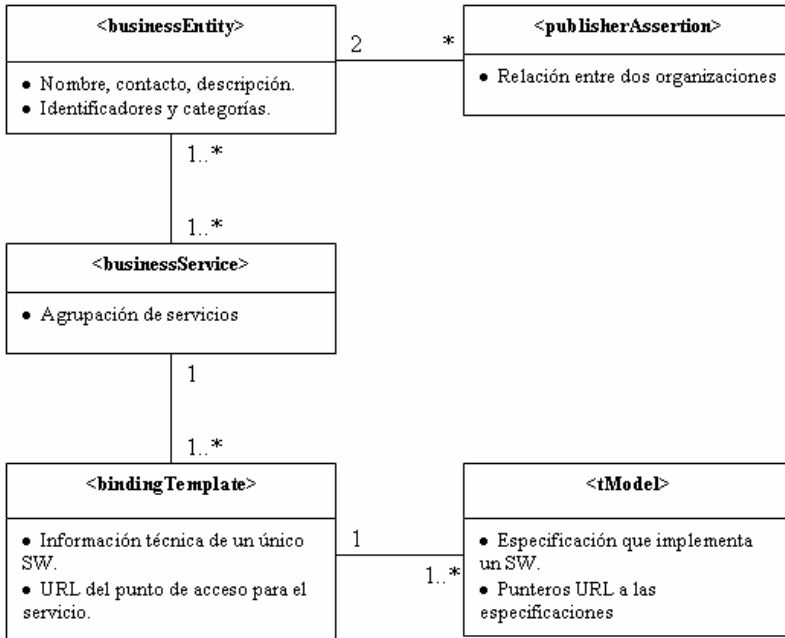


Figura 34. Relaciones entre las principales estructuras de datos de UDDI.

Una estructura *<businessEntity>* contiene una o más estructuras *<businessService>*. La estructura *<businessService>* representa una única clasificación lógica del servicio. Esta estructura se utiliza para describir un conjunto de servicios que son provistos por el negocio. Estos servicios van desde los SW, a cualquier tipo de servicio que pueda ofrecer el negocio, aunque no sean de carácter electrónico.

Una estructura *<businessService>* contiene una o más estructuras *<bindingTemplate>*. La estructura *<bindingTemplate>* contiene punteros a descripciones técnicas y a las URL de los puntos de acceso, pero no contiene información sobre la especificación del servicio. Esta estructura cuenta con una descripción textual, opcional, sobre el SW.

La estructura *<tModel>* es un tipo de huella digital mediante la cual podemos determinar como interactuar con un SW en particular. Esta estructura tampoco provee la especificación del SW, pero en su lugar, cuenta con punteros a la ubicación de la especificación para su acceso. De esta forma, las compañías pueden utilizar la información que aporta el *<tModel>* para determinar cómo o cuándo un SW es compatible con los requisitos de sus negocios.

Para identificar y acceder a estas estructuras de datos, se las identifica y referencia mediante un identificador universal conocido como el UUID. Estos UUID se asignan cuando se crea la estructura de datos por primera vez en el registro UDDI. Son cadenas hexadecimales cuya estructura y algoritmo de generación queda determinado por el estándar ISO/IEC 11578:1996. Mediante este método se garantiza la generación de un identificador único concatenando la hora, la dirección hardware, la dirección IP, y un número aleatorio. La API de pregunta, utiliza este identificador para solicitar cualquier tipo de estructura.

3.3.3.2. Información Básica

Mediante los mensajes SOAP, podemos recuperar del registro UDDI información básica sobre un negocio, un SW, o meta-datos de alguna especificación de un SW. Todos estos mensajes SOAP tienen en común que los elementos del cuerpo del mensaje XML comienzan todos con el prefijo *find*. En la Tabla 5 están reunidos todos los mensajes que podemos utilizar para recuperar la información necesaria.

Nombre del Mensaje	Documento de Respuesta	Descripción
<find_binding>	<bindingDetail>	Dado un UUID de una estructura <businessService>, este mensaje recupera cero o más estructuras <bindingTemplate> con una única estructura <bindingDetail> según el criterio especificado por los argumentos de entrada.
<find_business>	<businessList>	Dada una expresión regular, una categoría de negocio, un identificador de negocio o un <tModel>, el mensaje devuelve cero más estructuras <businessInfo> con una única estructura <businessList> que satisface los argumentos de entrada
<find_relatedBusinesses>	<relatedBusinessesList>	Dado un UUID de un <businessEntity>, el mensaje devuelve una lista de UUIDs en una estructura <relatedBusinessList> con otros negocios que tengan alguna relación con el primero
<find_service>	<serviceList>	Dado el UUID de un <businessEntity> y el nombre del servicio, el <tModel> de una especificación implementada, o la categoría del servicio, el mensaje devuelve una lista de todos los documentos <businessService> encontrados en una estructura <serviceList>
<find_tModel>	<tModelList>	Dado el nombre, o la categoría, o el identificador, este mensaje devuelve todas las estructuras <tModel> que coincidan en una estructura <tModelList>

Tabla 5. Documentos XML utilizados para la recuperación de información.

En la primera columna, podemos ver los nombres de los elementos raíz utilizados como cuerpo del sobre SOAP en las peticiones. La segunda columna, muestra los nombres de los elementos raíz utilizados como cuerpo del sobre SOAP para el mensaje de respuesta.

3.3.3.3. Información Detallada

La función *find_messages* se diseñó para devolver información básica sobre las estructuras que mantiene un registro UDDI. Dado el identificador UUID de una de las estructuras principales, podemos extraer del registro la lista completa de los detalles almacenados sobre esa estructura de la que sabemos el identificador (Figura 35). La API de pregunta de UDDI provee una serie de mensajes que comienzan con el prefijo *get_* para recuperar información sobre el registro. Podemos encontrar una lista de estos mensajes en Tabla 6.

Nombre del Mensaje	Documento de Respuesta	Descripción
<get_bindingDetail>	<bindingDetail>	Dado uno o más UUID de diferentes documentos <bindingTemplate>, este mensaje devuelve una estructura <bindingDetail> conteniendo un documento <bindingTemplate> completo para cada UUID.
<get_businessDetail>	<businessDetail>	Dado uno o más UUID de diferentes documentos <businessEntity>, este mensaje recupera una estructura <businessDetail> que contiene documentos <businessEntity> para cada UUID encontrado.
<get_serviceDetail>	<serviceDetail>	Dado uno o más UUID de diferentes documentos <businessService>, este mensaje devuelve una estructura <serviceDetail> que contiene el documento <businessService> completo para cada UUID encontrado.
<get_tModelDetail>	<tModelDetail>	Dado uno o más UUID, o diferentes documentos <tModel>, el mensaje devuelve una estructura <tModelDetail> con el documentos <tModel> completo para cada UUID encontrado.

Tabla 6. Documentos XML utilizados para la recuperación de información detallada

El uso de estos mensajes es muy sencillo. Tan pronto como tengamos el identificador UUID de la estructura, podremos recuperar todos los detalles que nos sean necesarios.

```
<uddi:get_serviceDetail generic="2.0">  
  <uddi:serviceKey>860eca90-c16d-11d5-85ad-801eef208714</uddi:serviceKey>  
</uddi:get_serviceDetail>
```

Figura 35. Consulta de información detallada sobre un servicio.

El mensaje `<get_serviceDetail>` (Figura 35) no tiene atributos opcionales, sólo un elemento hijo, el `<serviceKey>`, que es el UUID del servicio del que se quieren obtener detalles. El mensaje `<get_serviceDetail>` puede aceptar más de un elemento `<serviceKey>` para la consulta.

La estructura `<businessService>` constituye una agrupación de los servicios de un negocio. Esta estructura se encuentra en los mensajes de respuesta, y tiene como sub-elemento a `<bindingTemplate>`, que provee detalles técnicos sobre como acceder a un SW. El elemento `<accessPoint>` tiene la URL de acceso al SW. El elemento `<bindingTemplate>` contiene elementos `<tModelInstanceInfo>`, que dan información de dónde encontrar información sobre cómo funciona un determinado SW y su especificación. Cada uno de estos elementos, contiene un atributo `tModelKey` cuyo valor es el identificador UUID de una estructura `<tModel>`, que a su vez contiene metadatos de una especificación concreta. El elemento `<tModelInstanceInfo>` contiene también un elemento del tipo `<instanceDetails>`, que es una descripción de cómo usar un SW.

Categorización

El elemento `<businessService>` contiene una estructura `<categoryBag>`. Podemos encontrarnos esta estructura también junto con los elementos `<businessEntity>` y `<tModel>`. La categorización de los datos es un requisito muy importante del registro UDDI.

La categorización permite que los datos en un registro UDDI sean asociados con una industria, un producto o un conjunto de códigos geográficos. De esta forma, obtenemos a priori un conjunto de criterios por los que realizar las búsquedas en el registro, además de organizar de una forma sencilla y eficiente las ingentes cantidades de datos que hay y que habrá en el registro UDDI.

Cuando dentro de un negocio nace la necesidad de utilizar SW, se presenta el problema de localizar ese SW que satisfaga de manera eficiente esa tarea para la que será requerido. Es bastante difícil conseguir que un programa, pos sí solo, descubra y use dinámicamente esos SW. Es por esto, que se presenta otra opción más asequible, y es que los propios analistas utilicen portales especiales que interroguen a los registros UDDI en busca de SW, de forma que la búsqueda se hace menos compleja. Pero gracias a esta categorización, podemos dotar a los programas de la lógica necesaria para localizar e integrar con el negocio esos SW que son nos son necesarios.

Existen diferentes sistemas de categorización para usar con el registro UDDI (Tabla 7):

Nombre de la Taxonomía	Nombre del tModel	Descripción
NAICS	ntis-gov:naics:1997	http://www.census.gov/epcd/www/naics.html .
UNSPSC	unspsc-org:unspsc:3-1	http://www.unspsc.org/ .
ISO 3166	iso-ch:3166:1999	http://www.din.de/gremien/nas/nabd/iso3166ma .
Other	uddi-org:general_keywords	Asociaciones de propósito general que un negocio podría querer realizar. Esta taxonomía permite a los nodos operadores promover entradas que podrían ser no válidas o rechazadas por otros sistemas de clasificación. No existen ninguna clasificación de cómo trabaja, ya que es específico del nodo operador.

Tabla 7. Taxonomías de categorización

Cada taxonomía de categorización se registra como una estructura <tModel> dentro de UDDI. Cada categorización tendrá un nombre tModel y un UUID para referenciarlo. El nombre tModel será el mismo en todos los registros UDDI, pero el identificador UUID para un tModel, podrá no ser el mismo en los distintos nodos operadores.

La estructura <categoryBag> contiene cero o más estructuras de tipo <keyedReference>. Cada una de las estructuras <keyedReference> contiene el nombre y el valor de la categoría a la que pertenecen los datos.

```
<keyedReference keyName="Netherlands"
keyValue="NL"
tModelKey="uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88"/>
```

Figura 36. Categorización en ISO 3166 de una zona, devuelto como parte de un <categoryBag>

En la Figura 36, el atributo *keyName* identifica la categorización, el atributo *keyValue* corresponde al código de categorización, que además se garantiza que es único, y el atributo *tModelKey* representa el valor del UUID de una estructura <tModel> que contiene los meta-datos de la especificación que soporta esa categorización.

Identificadores

Un identificador es una propiedad utilizada para identificar unívocamente un negocio o una especificación. Se pueden aplicar a las estructuras <businessEntity> y <tModel>. Al igual que ocurría con las categorizaciones, podemos utilizar los identificadores como parte una búsqueda cuando enviamos los mensajes de petición <find_business> o <find_tModel>.

Los identificadores están vinculados a las estructuras <businessEntity> y <tModel> mediante la estructura <identifierBag>. Esta última estructura puede tener uno o más estructuras <keyedReference> a su vez, que proveen el nombre, el valor y la referencia del UUID del <tModel> para poder localizar más información.

Actualmente, sólo hay propuestos dos esquemas de identificación incorporados a los nodos operadores, estos son:

Nombre Identificador	Nombre tModel	Descripción
D-U-N-S	dnb-com:D-U-N-S	El número Dun & Bradstreet D-U-N-S es una secuencia de identificación de nueve dígitos única. Más información en http://www.d-u-n-s.com/ .
Thomas Register	thomasregister-com:supplierID	Este esquema provee identificadores para más de 150.000 empresas de comercio electrónico y manufacturación del mundo. Mas información en http://www.thomasregister.com/ .

Tabla 8. Tipos de identificadores soportados.

tModel

Los documentos *<tModel>* proveen meta-datos sobre la especificación de un SW, especificación sobre la categorización, o especificación sobre el identificador. Los documentos *<tModel>* constituyen el núcleo de la estructura de datos en la especificación de UDDI, y representan la información más detallada que un registro UDDI puede proveer.

```
<uddi:get_tModelDetail generic="2.0">  
  <uddi:tModelKey>uuid:93335d49-3efb-48a0-acea-ea102b60ddc6</uddi:tModelKey>  
</uddi:get_tModelDetail>
```

Figura 37. Petición de información detallada de un tModel.

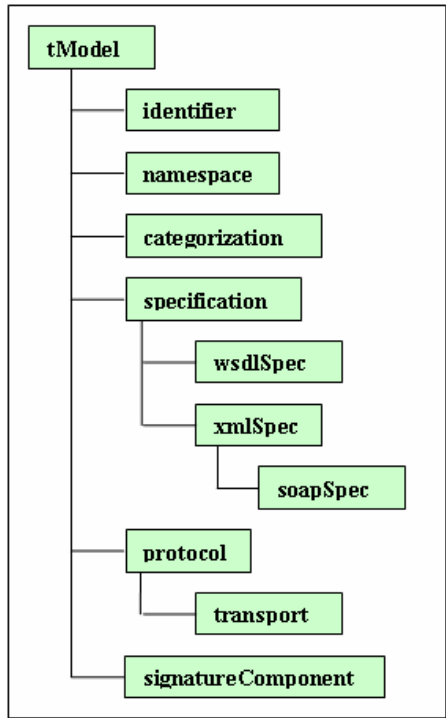


Figura 38. Estructura del elemento *tModel*.

3.3.3.4. Publicar en un Registro UDDI

La publicación en un registro UDDI involucra a cualquier operación que pueda crear, actualizar o destruir datos en un registro UDDI. A un

registro UDDI podemos querer acceder bien para publicar, bien para consultar. Teniendo en cuenta que son dos acciones bastante diferentes, enumeraremos algunas de las diferencias técnicas existentes entre la publicación y la consulta:

- *Acceso autenticado.* Todos los mensajes de publicación requieren un acceso autenticado. El proceso para la autenticación no está definido en la especificación UDDI, por lo que es específico de cada nodo operador. Una vez obtenidas las credenciales de autenticación, se podrá acceder y enviar mensajes de publicación las veces que sean necesarias.
- *Puntos de acceso diferentes.* Los mensajes de petición de publicación usan diferentes puntos de acceso que los mensajes de consulta. El protocolo HTTP es más susceptible de utilizar por los mensajes de consulta, mientras que el protocolo HTTPS será más utilizado por los mensajes de publicación, ya que estos últimos necesitan realizar accesos seguros y autenticados.
- *Limitación de espacio.* Los nodos operadores pueden imponer restricciones acerca del espacio y el registro de una empresa concreta.
- *Asociación al nodo operador.* Cuando se inserta información en un nodo operador, este se convierte en el propietario principal de la copia de los datos. Los cambios o las modificaciones sucesivas sobre los datos, deberán realizarse sobre el mismo nodo operador. UDDI no tiene ningún mecanismo para resolver conflictos en caso de que se haga una entrada duplicada en otro nodo operador.

A continuación, mostramos la API de publicación que requiere autenticación:

Nombre del Mensaje	Documento Respuesta	Descripción
<add_publisherAssertions>	<dispositionReport>	Dado un token de autenticación válido y un documento <publisherAssertion>, este mensaje agrega a un <publisherAssertion> a una colección individual <publisherAssertion>. De esta forma creamos una asociación entre dos negocios. Cuando los dos negocios han incluido respectivamente los documentos <publisherAssertion>, la relación se hace visible.
<delete_binding>	<dispositionReport>	Dado un token de autenticación válido y el UUDI

Nombre del Mensaje	Documento Respuesta	Descripción
	ort>	de uno o más documentos <bindingTemplate>, este mensaje elimina los documentos <bindingTemplate> que coincidan con el identificador del registro UDDI.
<delete_business>	<dispositionReport>	Dado un token de autenticación válido y el UUID de uno o más documentos <bindingEntity>, el mensaje elimina los documentos <bindingTemplate> que coincidan del registro UDDI. Se eliminan por tanto los <businessService> y los <businessTemplate>. De manera adicional, se eliminarán también las estructuras <publisherAssertions> con el mismo UUID creados en el <businessEntity>.
<delete_publisherAssertions>	<dispositionReport>	Dado un token de autenticación válido y el UUID de uno o más documentos <publisherAssertion>, este mensaje elimina los documentos <publisherAssertion> coincidentes que de la colección.
<delete_service>	<dispositionReport>	Dado un token de autenticación válido y el UUID de uno o varios documentos <businessService>, el mensaje elimina los documentos <businessService> que coincidan del registro UDDI.
<delete_tModel>	<dispositionReport>	Dado un token de autenticación válido y los UUID de uno o más documentos <tModel>, este mensaje elimina <i>lógicamente</i> los documentos <tModel> registro UDDI, marcándolos como ocultos. Aunque los documentos no se destruyen, no se podrán recuperar en una consulta que se realice sobre le registro.
<discard_authToken>	<dispositionReport>	Dado un token de autenticación válido, este mensaje comunica a un nodo operador que se deshaga de una sesión de autenticación.
<get_assertionStatusReport>	<assertionStatusReport>	Dado un token válido de autenticación, este mensaje devuelve un informe que detalla los documentos <publisher-Assertion> que han sido creados en cualquier documento <businessentity> gestionado por este editor.
<get_authToken>	<authToken>	Dado un nombre de usuario y una clave, este mensaje recupera un token de autenticación de un nodo operador para ser utilizado sobre mensajes de otra API de Publicación.
<get_publisherAssertions>	<publisherAssertions>	Dado un token de autenticación válido, este mensaje devuelve una lista completa de documentos <publisherAssertion> que ha sido asociado con la cuenta autenticada del editor.
<get_registeredInfo>	<registeredInfo>	Dado un token válido de autenticación, este mensaje devuelve una lista completa de los documentos <businessEntity> y los <tModel> que son gestionados por aquel que se ha autenticado.
<save_binding>	<bindingDetail>	Dado un token de autenticación y uno o más

Nombre del Mensaje	Documento Respuesta	Descripción
		documentos <bindingTemplate>, este mensaje inserta o actualiza el registro UDDI con los documentos <bindingTemplate> de entrada. Este mensaje puede además actualizar cualquier asociación realizada entre un documento <businessService> y otro <bindingTemplate>. Este mensaje devuelve un mensaje <bindingDetail> que contiene los resultados finales de la invocación inicial, mostrando la información final del registro UDDI.
<save_business>	<businessDetail>	Dado un token de autenticación y uno o más documentos <businessEntity>, este mensaje inserta o actualiza un registro UDDI con los documentos <businessEntity> que se p asan como entrada. Este mensaje devuelve otro mensaje <businessDetail> que contiene los resultados finales con la información actual del registro UDDI.
<save_service>	<serviceDetail>	Dado un token de autenticación y uno o más documentos <businessService>, este mensaje inserta o actualiza un registro UDDI con los documentos <businessService> que hemos pasado como entrada. Este mensaje también puede modificar los datos de los documentos <businessService> y cualquier referencia a estructuras <bindingTemplate>. Este mensaje devuelve un mensaje <serviceDetail> que contiene el resultado final de la llamada que refleja la información actual del registro UDDI.
<save_tModel>	<tModelDetail>	Dado un token de autenticación y uno o más documentos <tModel>, este mensaje inserta o actualiza un registro UDDI. Si uno de los documentos <tModel> pasados como argumentos hace referencia a un <tModel> que ya ha sido eliminado ocultándolo, lo hará visible de nuevo. Este mensaje devuelve un mensaje <tModelDetail> que contiene el resultado final de la invocación y refleja la información actual del registro UDDI
<set_publisherAssertions>	<publisherAssertions>	Dado un token de autenticación y uno o más documentos <publisherAssertion>, este mensaje actualiza un registro UDDI que contenga una colección completa de los documentos <publisherAssertion>, eliminando los documentos que no forman parte de los documentos de entrada. Este mensaje devuelve un documentos <publisherAssertions> con la actual colección de documentos <publisherAssertion> almacenados en el registro UDDI

Tabla 9. Mensajes de la API de publicación UDDI.

3.3.3.5. Seguridad y Autenticación

La autenticación con los nodos operadores es muy sencilla. La mayoría de los nodos operadores implementa un esquema de autenticación basado en nombre/clave que nos permite recuperar el *token de autenticación* que utilizaremos para realizar las operaciones que requieran que nos identifiquemos como sujetos autorizados.

Los nodos operadores que utilizan este esquema de autenticación exponen sus interfaces de autenticación mediante el mensaje `<get_authToken>`. Los nodos operadores que no soportan este esquema de autenticación, proveen a los clientes algún otro medio que les permita conseguir el *token* de autenticación.

Estas técnicas no están especificadas en la especificación de UDDI, por lo que son específicas de cada nodo operador. Los nodos operadores también tienen sus propios métodos de registrar a los nuevos publicadores y de verificar su información.

Para solicitar un *token* de autenticación a un nodo operador tendríamos que mandar un mensaje SOAP con la siguiente estructura (entre otras cosas):

```
<uddi:get_authToken generic="2.0" userID="admin" cred="changeit" />
```

Figura 39. Petición del token de autenticación.

El elemento `<get_authToken>` no tiene elementos hijos, por lo que pasa el nombre y la clave mediante los atributos *userID* y *cred* respectivamente. La supuesta respuesta del nodo operador podría ser la siguiente:

```
<authToken xmlns="urn:uddi-org:api_v2" generic="2.0" operator="SYSTINET">
  <authInfo>
    MIHLMDYbBWFkdbWlUaMB4XDTAxMTIzMDAwNDYzNVoXDTAxMTIzMDAwNDYzNVoEDUFkdb
    WlUaXNOcmF0b3IwDQYJKoZIhvcNAQEEBQADgYEA4Cci/CbDji6RiQFneRt7gVXwX/
    4TA7qCZNUmTnXFJdVNFIDvp4WV+IW+/
    deDCQk0GVAdsuhOvkXJX3dqDGqDsDleXwm7cDN2ENW7K/IeN9ii7/
    pfbVryPtKzzbe07ETcWaoRnkegDteC7I77VpyiqKHUqumi5+kN10XMRXfkTw=
  </authInfo>
</authToken>
```

Figura 40. Token de autenticación devuelto a una petición de autenticación.

El documento de respuesta contiene un elemento `<authToken>` que a su vez tiene un elemento `<authInfo>`, cuyo valor es la clave que será usada como *token* de autenticación en todos los mensajes de publicación.

4. OTROS ESTÁNDARES

4. OTROS ESTÁNDARES

4.1. Orquestación y Coreografía

Para este apartado nos hemos basado principalmente, entre otras publicaciones y fuentes, en [20], ya que ofrecen un buen resumen sobre la orquestación y la coreografía.

4.1.1. Introducción

La combinación de SW para la implementación de procesos de alto nivel, requiere de diversos estándares que nos permitan modelar las posibles interacciones entre los servicios. Los métodos actuales de creación de procesos de negocio no están diseñados para trabajar con procesos que involucren componentes de distintas organizaciones, ya que esto conllevaría una coordinación extra a la que ordinalmente ofrecen. Así pues, introducimos los términos de *orquestación* y *coreografía*, que tratan de describir aspectos relacionados con la creación de procesos de negocio que involucran varios SW.

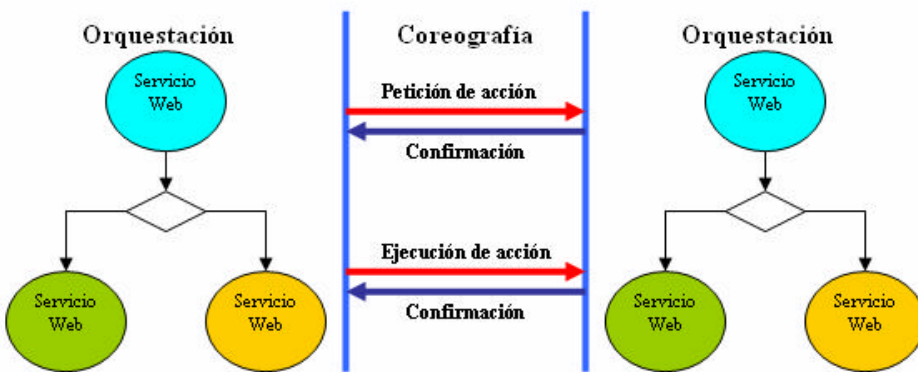


Figura 41. Relación entre orquestación y coreografía de SW.

La orquestación, permite diseñar procesos de negocio ejecutables que puede interactuar (a nivel de mensaje) tanto con SW internos como externos. Con la orquestación, siempre representamos el control del proceso de negocio desde el punto de vista de una de las partes participantes que intervienen en el mismo. Sin embargo, la coreografía es mucho más colaborativa, ya que permite trazar las secuencias de mensajes que se suceden entre todas las partes participantes del proceso

de negocio en lugar de centrarse en los mensajes que intercambian los SW que implementan los procesos de negocio ejecutados únicamente por una parte.

4.1.2. Estándares relacionados

Algunos trabajos relacionados con la combinación de SW en los procesos de negocio, son los de Microsoft, con su estándar *XLANG* e IBM con su *Web Service Flow Language* o *WSFL*. Aunque más tarde, estos estándares se combinaron para dar lugar al *Business Process Execution Language for Web Services* o *BPEL4WS*. *BPEL4WS* es una de las especificaciones actuales para la orquestación de SW.

Inicialmente, Microsoft desarrollo *XLANG* a fin de soportar los flujos de procesos secuenciales, paralelos y condicionales de su servidor BizTalk. *XLANG* se sirve del Lenguaje de Descripción de SW [29] (WSDL) propuesto por el W3C [31] para describir las interfaces de los SW. *XLANG* se centra en la creación de procesos de negocio y en los patrones de intercambio de mensajes entre los SW, ofreciendo además opciones para la captura de excepciones y soporte para transacciones de larga duración.

Por otra parte, el *WSFL* de IBM, describe tanto los flujos de trabajo públicos como los privados. Define tanto los intercambios de datos, como la secuencia de ejecución y la descripción de cada uno de los pasos a realizar dentro del flujo para operaciones específicas. La interfaz de *WSFL* es como la de *WSDL*, por lo que también admite composición recursiva. Aunque *WSFL* no soporta directamente transacciones, si puede capturar las excepciones.

4.1.2.1. Lenguaje de Ejecución de Procesos de Negocio (BPEL)

La especificación de *BPEL4WS* fue codesarrollada por Microsoft, IBM, Siebel Systems, BEA y SAP. Esta especificación modela el comportamiento de los diversos SW que puedan participar en la interacción de un proceso de negocio.

BPEL, como se le llama abreviadamente, provee de una sintaxis en XML para la descripción de la lógica de control necesaria para coordinar los

SW que participen en un flujo de proceso. Esta gramática es ejecutada por un motor de orquestación que coordina todas las actividades y compensa el proceso global cuando ocurre algún error. BPEL puede considerarse como una capa que se tiende sobre WSDL, ya que este último define las operaciones disponibles, mientras que BPEL define como secuenciarlas.

BPEL provee de soporte tanto para procesos de negocio ejecutables como abstractos. Un proceso de negocio ejecutable modela el comportamiento de los participantes en una interacción concreta del negocio, esencialmente, modelando un flujo de trabajo privado. Un proceso abstracto o protocolo de negocio, especifica el intercambio público de mensajes entre las distintas partes que participan en la comunicación. La diferencia entre los procesos ejecutables y los abstractos, es que mientras que los procesos ejecutables modelan la orquestación, los procesos abstractos modelan la coreografía de los servicios.

La especificación de BPEL incluye soporte tanto para actividades básicas como para actividades estructuradas. Una actividad básica consiste en una instrucción que interactúa con algo externo al proceso en sí, mientras que las actividades estructuradas gestionan todo el flujo de procesos, especificando la secuencia para los SW referenciados.

Dos elementos importantes en BPEL son las *variables* y los *partnerLinks*, donde:

- Una *variable* identifica el dato específico que se intercambia en un flujo de mensaje. Cuando un proceso BPEL recibe un mensaje, se guarda en la variable apropiada, de forma que sucesivas peticiones puedan acceder a los datos. De esta forma, las variables permiten gestionar la persistencia de los datos a través de las sucesivas peticiones de los SW.
- Un *partnerLink* puede ser cualquier servicio al que el proceso invoque, o por el contrario, cualquier servicio que invoque al proceso. Cada *partnerLink* se traduce a un rol específico que se desempeña en el proceso de negocio.

A fin de agrupar en una única transacción un conjunto de actividades, BPEL usa una etiqueta *scope*. Esta etiqueta, indica que los pasos que estén dentro de la misma, deberán realizarse de forma atómica. Dentro de este alcance, los desarrolladores pueden definir distintos manejadores de compensación que el motor de orquestación BPEL pueda invocar en caso de error.

4.1.2.2. Interfaz de Coreografía de Servicios Web (WSCI)

La especificación de WSCI fue desarrollada por diversas compañías como Sun, SAP, BEA e Intalio. WSCI define una extensión para WSDL para la colaboración. La especificación define la coreografía global o el intercambio de mensajes entre los SW. WSCI soporta correlación, reglas de secuenciación, manejo de excepciones, transacciones y colaboración dinámica.

WSCI sólo describe el comportamiento observable entre SW (Figura 42). Una única interfaz WSCI describe solamente cual es la participación de uno de los servicios que intervienen en el intercambio de mensajes. Es por esto último por lo que una única interfaz WSCI define a un único SW, una coreografía WSCI constará de un conjunto de interfaces WSCI, uno por cada uno de los servicios que participen en la interacción. Además, dentro de WSCI, ningún proceso gestiona la interacción.

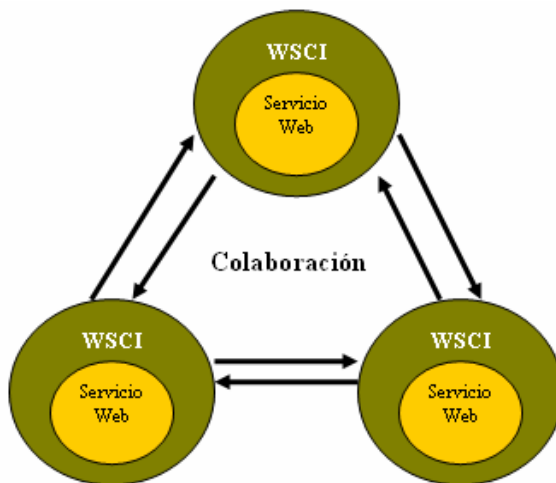


Figura 42. Colaboración al estilo WSCI.

Cada acción WSCI representa una unidad de trabajo que se corresponde con una operación WSDL específica. Como hemos dicho, WSCI extiende a WSDL describiendo cómo coreografiar las operaciones disponibles en el documento WSDL. WSDL describe cuáles son los puntos de entrada para un servicio determinado, y WSCI describe las interacciones entre las operaciones WSDL.

WSCI soporta transacciones de negocio y manejo de excepciones. Mediante WSCI, podemos establecer contextos transaccionales específicos (entendiendo por contexto, el conjunto de actividades que deben ejecutarse de forma atómica), tal y como hacíamos con la etiqueta *scope* en BPEL4WS, cuando definíamos que actividades debían ejecutarse de forma atómica.

4.1.2.3. Lenguaje de Gestión de Procesos de Negocio (BPML)

BPML es un lenguaje basado en XML para la descripción de procesos de negocio. Inicialmente, BPML se diseñó para soportar los procesos que un sistema de gestión de procesos de negocio pudiera ejecutar.

BPML y WSCI comparten el mismo modelo de ejecución de procesos, así, podemos usar WSCI para describir las interacciones públicas entre los procesos de negocio y utilizar BPML para desarrollar implementaciones privadas.

BPML dispone de estructuras similares a BPEL, como pueden ser actividades básicas para el envío y la recepción de mensajes, y para la invocación de servicios. Además, BPML permite planificar las tareas para que se ejecuten en instantes específicos, ofreciendo algunas opciones de control temporal.

Este lenguaje incluye características para dar soporte a la persistencia, ya que fue diseñado para la gestión de procesos de larga duración. BPML ofrece soporte también para realizar composiciones recursivas a fin de poder construir procesos de negocio que a su vez consten de subprocesos de negocio.

BPML soporta transacciones de larga o corta ejecución, gracias a que usa una técnica similar a BPEL para controlar el alcance de las actividades, y gestionar de esta forma las reglas de compensación en caso de que las

operaciones que deben realizarse de forma atómica se vean interrumpidas.

4.1.3. Colaboración Orquestada y Coreografiada

A pesar de sus múltiples similitudes, BPEL, WSCI y BPML difieren en algunos aspectos a la hora de realizar la coreografía y la orquestación de los SW. Tanto BPEL como BPMI, están provistos de capacidades para la definir procesos de negocio ejecutables, mientras que la aproximación que ofrece WSCI está más coreografiada y colaborativa.

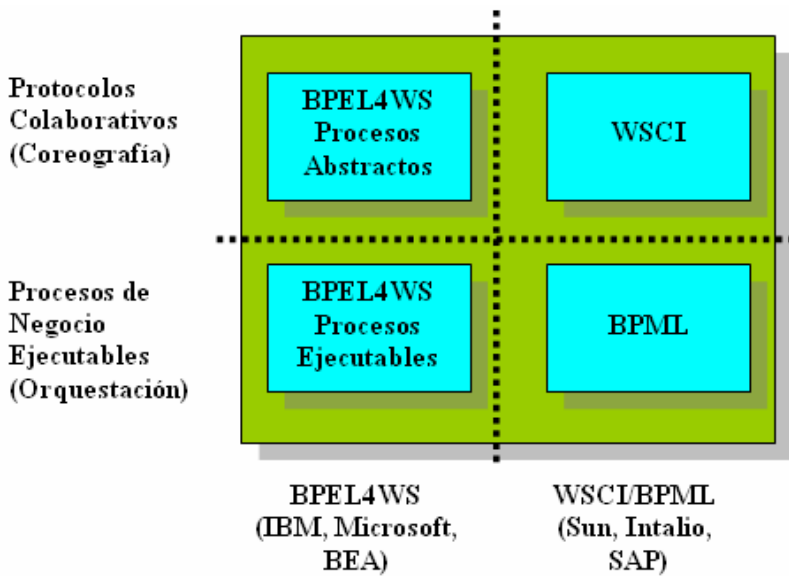


Figura 43. Relación entre los estándares de orquestación y coreografía.

En la Figura 43, podemos ver una clasificación de estas tecnologías según los protocolos colaborativos (referentes a la coreografía) y los procesos ejecutables de negocio (referentes a la orquestación). Un protocolo colaborativo se refiere al intercambio coreografiado de mensajes entre las múltiples partes del negocio, mientras que los procesos ejecutables son flujos de trabajo privados controlados por una única entidad. Esta clasificación también se hace atendiendo a las iniciativas existentes, es decir, por un lado la de BPEL, y por otro la de WSCI y BPML.

4.2. Coordinación y Transacciones

Las transacciones en el ámbito de los SW difieren en cierto modo de las conocidas hasta ahora, y las semánticas utilizadas para las transacciones clásicas junto con los protocolos existentes han demostrado no ser suficientes para abordar de manera consistente las transacciones que ocurren en los procesos de negocios. En el ámbito de los SW, las transacciones tienen un periodo de duración que puede ser bastante más largo que el acostumbrado.

El modelo de transacción que soporta las nuevas transacciones de negocio está basado en los estándares de los SW. Para ello se define tanto un protocolo de transacción interoperable, como los flujos de mensajes que ayudan a negociar las garantías de las transacciones.

La OASIS BTP (*Organisation for Advance Structured Information Business Transaction Protocol*) es un protocolo que satisface este tipo de transacciones de larga duración entre varias entidades, permitiendo también relajar (de una forma controlada) las propiedades ACID en los entornos orientados a SW.

El avance de las nuevas infraestructuras de Internet y el desarrollo de nuevos protocolos han hecho mella en los entornos de negocio, motivando la aparición de modelos de transacción extendidos mejor equipados para la interoperación a través de Internet.

Consideramos que una relación de negocio es cualquier estado distribuido mantenido por dos o más partes sujeto a un conjunto de restricciones contractuales acordadas por las partes participantes. Por otra parte, una transacción de negocio puede verse como un cambio consistente en el estado de una relación de negocio entre distintas partes. Dentro de esta transacción de negocio, cada una de las partes, alberga su propio estado de la aplicación correspondiente a la relación de negocio establecida con las otras partes participantes en la transacción.

Las partes participantes en una transacción deben estar preparadas para reaccionar ante la terminación de la transacción, ya sea por terminación normal (actualizarán su estado de acuerdo a los resultados) o por un fallo (recuperaran su estado anterior).

Tradicionalmente, las transacciones atómicas han sido la mejor técnica para garantizar la consistencia ante la presencia de fallos [8]. Las propiedades ACID (*Atomicidad, Consistencia, Aislamiento y Durabilidad*) permiten preservar el estado a pesar de los fallos que puedan ocurrir, pero estas propiedades no se cumplen en las transacciones de negocio en el ámbito de los SW, ya que las transacciones se consideran de larga duración (no como las clásicamente conocidas, que eran de corta duración, cumpliendo con ello las propiedades ACID).

En las transacciones tradicionales, se utilizan protocolos de dos fases (Figura 44) para lograr la atomicidad entre los participantes. Este protocolo debe tener también la capacidad de bloquear para garantizar el consenso entre las entidades envueltas en la transacción, esto es, que una vez que los participantes aceptan el primer paso del protocolo de dos fases, bloquean sus recursos hasta saber si la transacción ha finalizado correctamente o, por el contrario, ha fallado en algún punto. De esta forma, los recursos bloqueados no están disponibles para otras transacciones. En caso de que el coordinador falle antes de distribuir el mensaje de la segunda etapa del protocolo los participantes quedan bloqueados hasta que éste se recupere.

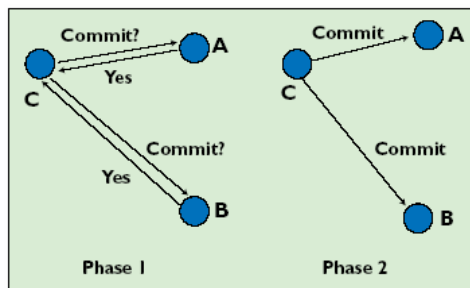


Figura 44. Protocolo de confirmación de dos fases [15]

Sin embargo, las actividades de larga duración pueden ser estructuradas como transacciones independientes de corta duración, que formen una transacción lógica [7], lo que permite acaparar los recursos solo el tiempo necesario. En caso de que una de estas actividades se vea interrumpida, el comportamiento de la transacción “lógica” de larga duración no poseería las propiedades ACID [15]. Tras esto se inician actividades de compensación que deshace los posibles cambios realizados por las transacciones independientes.

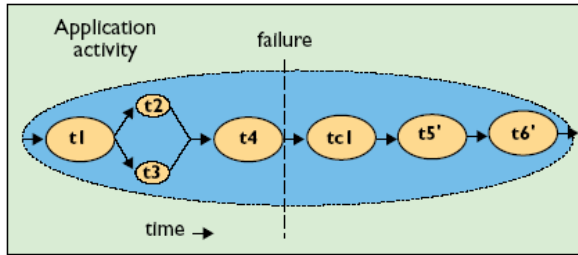


Figura 45. Fallo en una transacción “lógica” de larga duración [15]

Mediante el protocolo BTP podemos coordinar las distintas partes autónomas que intervienen en una transacción de negocio. Sus aspectos fundamentales son los siguientes:

- *Consenso en la opinión*: BTP utiliza un protocolo de finalización de dos fases con tres etapas: preparación, confirmación y cancelación. Es similar al protocolo de confirmación de dos fases anteriormente mencionado, mediante el cual se logra la atomicidad entre los distintos participantes de una transacción. Este protocolo no asegura el cumplimiento de las propiedades ACID.
- *Coordinación Open-Top*: En los sistemas transaccionales clásicos, las acciones que se podían realizar estaban realmente limitadas a sólo tres: “begin”, “commit” y “rollback”. Para BTP, un protocolo de confirmación *closed-Top* es aquel que se sigue cuando una aplicación solicita la realización de una transacción, y el coordinador de la misma ejecuta el protocolo de dos fases completo antes de retornar una respuesta (siendo muy breve el tiempo entre las dos fases, variando entre milisegundos y segundos).

BTP permite que el tiempo a transcurrir entre las dos fases del protocolo sea determinado por la misma aplicación que solicita la ejecución de la transacción. Esto se consigue mediante la extensión del conjunto de acciones que se pueden realizar para obtener un control más preciso sobre las dos fases del protocolo, estas nuevas acciones son: “prepare”, “confirm” y “cancel”. BTP denomina a esto “protocolo de confirmación *Open-Top*”. Ahora la aplicación tiene un control total sobre la preparación de la transacción y mediante la lógica de negocio necesaria, podría determinar después que transacción confirmar y cual cancelar.

- *Soporte XML para BTP*: Existe un lenguaje en XML especificado por el correspondiente esquema para el envío de mensajes entre los participantes de una transacción y la infraestructura de coordinación, así como un formato para la descripción del contexto de las transacciones. BTP se ha integrado perfectamente con la tecnología de SW debido al gran auge que ha experimentado. Para ello se ha definido una vinculación entre el conjunto de mensajes BTP con la versión 1.1 de SOAP sobre HTTP 1.1. Los mensajes se dividen en dos tipos: uno tipo para la infraestructura BTP y otro para la aplicación.
- *Optimizaciones*: BTP dispone de un conjunto de optimizaciones para mejorar el rendimiento, algunas de ellas son:
 - Dimisión de los participantes: En la fase de preparación del protocolo de confirmación de dos fases, un participante puede emitir una respuesta de “sólo escritura”. Esta respuesta implica que el emisor no va a modificar el estado de sus recursos a lo largo de la transacción, por lo que no necesita ser notificado del resultado de la misma. Con esto se obtiene una mejora en tiempo ya que se ahorra el envío de algunos mensajes. Este comportamiento tiene su homólogo en el protocolo BTP, y ocurre cuando un participante de la transacción, estando ya preparado, decide que no le interesa el resultado de la misma.
 - Decisión autónoma de los participantes: En el protocolo clásico de dos fases, se alcanza el consenso de todos los participantes para que esperen la respuesta del coordinador mediante una estrategia de bloqueo, esto es, si el coordinador falla antes de terminar y comunicar el resultado, los participantes de la transacción permanecerán bloqueados sin modificar el estado de sus recursos. Sin embargo, en los sistemas de procesamiento transaccional actuales, el protocolo de dos fases se ha mejorado mediante heurísticas, de forma que los participantes puedan decidir por ellos mismos si finalizan la transacción actualizando el estado de los recursos o, por el contrario, retornar al estado anterior sin hacer ningún cambio. Este comportamiento puede llevar a situaciones en las que el comportamiento del grupo sea no atómico. En BTP, la estrategia es similar, consiste en dotar de una serie de

heurísticas para que los participantes puedan tomar decisiones unilaterales, con la diferencia de que los participantes dan previamente al coordinador cierta información acerca de su comportamiento frente a determinadas situaciones. De esta forma el coordinador puede optimizar el envío de mensajes durante la transacción.

4.2.1. Composición de Servicios

Cada una de estas especificaciones tiene un propósito bien definido en el contexto de la composición robusta de servicios:

- BPEL permite que un conjunto de SW sea combinado en un nuevo SW mediante construcciones de modelado de procesos bien definidas.
- *WS-Coordination* es un marco de trabajo general para la implementación de tipos de coordinación específicos donde la coordinación de los SW requiere un contexto compartido.
- *WS-Transaction* define dos tipos de coordinación, las transacciones atómicas (de corta duración) y las transacciones de negocio (de larga duración).

La combinación de estas tres especificaciones permite que el modelo de composición de BPEL sea extendido con capacidades para lo coordinación distribuida. Las relaciones de captura de fallos y compensación entre los ámbitos BPEL pueden ser expresados como un tipo de coordinación *WS-Coordination*. *WS-Coordination* define el contexto de coordinación para utilizar en entornos donde los ámbitos BPEL están distribuidos o abarcan implementaciones de distintos distribuidores.

4.2.2. WSCoordination y WS-Transaction

La *WS-Coordination* (Coordinación de SW) y la *WS-Transaction* (Transacciones de SW) abordan el problema de la coordinación de las interacciones de los servicios de múltiples partes.

El fundamento de la *WS-Coordination* es proveer un mecanismo de coordinación genérico que puede ser extendido para otros protocolos de

coordinación específicos. Esta coordinación incluye la ejecución de transacciones de corta duración dentro de una organización (similares a transacciones distribuidas tradicionales) y transacciones de larga duración como las que se suceden a través de varias organizaciones.

La *WS-Transaction* define tales protocolos de coordinación específicos para transacciones atómicas (corta duración) y de negocio (larga duración).

WS-Coordination

Los modelos de transacción y coordinación específicos son representados como un tipo de coordinación que ofrece soporte a un conjunto de protocolos de coordinación. Un protocolo de coordinación está constituido por el conjunto de mensajes bien definidos que son intercambiados entre los SW participantes. Los protocolos de coordinación, tal y como son los protocolos de entrega y los protocolos de sincronización o protocolos de notificación del resultado de la operación, abordan el problema de la correcta ejecución de un conjunto de actividades distribuidas necesarias para alcanzar un resultado consistente y bien definido.

El *WS-Coordination* define tres elementos que son requeridos por los distintos modelos de coordinación:

- Contexto de Coordinación: Contexto compartido y extensible que representa la coordinación propagada a los participantes distribuidos.
- Servicio de Activación: Servicio utilizado por los clientes para crear un contexto de coordinación.
- Servicio de Registro: Es el servicio utilizado por los participantes para registrar los recursos para los protocolos de coordinación específicos.

El Servicio de Activación y el Servicio de Registro son servicios genéricos, pero ambos junto con los servicios que representan los protocolos de coordinación específicos para un tipo de coordinación dado, componen el Servicio de Coordinación.

La coordinación de un conjunto de SW estaría compuesta por los siguientes pasos:

1. Los clientes de la coordinación inician la coordinación enviando un mensaje de petición al Servicio de Activación a fin de seleccionar un coordinador.
2. El Servicio de Activación crea un Contexto de Coordinación que posee un identificador global, una fecha de expiración, un puerto de referencia para el servicio de Registro y, además, puede ser extendido para agregar otro tipo de información útil para los protocolos de coordinación específicos. El puerto de referencia es una descripción WSDL, que consiste en una URI del puerto y otros tipos de información.

WS-Transaction

WS-Transaction influencia a la *WS-Coordination* mediante la definición de dos tipos de coordinación particulares: *Transacción Atómica* (TA) y *Actividad de Negocio* (AN). La TA modela transacciones atómicas de corta duración, mientras que la AN modela transacciones de negocio que pueden llegar a tener una duración considerable.

La TA puede compararse con las transacciones tradicionales distribuidas., por lo que este tipo de coordinación soporta la propiedad de la *atomicidad* (es decir, o todo o nada, con respecto a la ejecución de operaciones distribuidas de SW) basándose en la premisa de que los datos que son manipulados a lo largo de la operación pueden ser recuperados en caso de que la operación no llegue a completarse.

El tipo de coordinación AN soporta la coordinación de transacciones que, potencialmente, puedan tener una larga duración. Este tipo de coordinación no requiere que los recursos accedidos sean guardados, pero si que es necesaria una determinada lógica de negocio para capturar las excepciones. Los participantes en la transacción son vistos como tareas de negocio, que son considerados como hijos de la AN en la que están registrados. La lista de participantes en la coordinación es dinámica, es decir, que un participante puede libremente dejar la transacción, además los participantes tienen un bajo acoplamiento entre sí.

5. ENTORNOS TECNOLÓGICOS

5. ENTORNOS TECNOLÓGICOS

5.1. Introducción

En los apartados anteriores, se ha tratado la especificación de los SW y sus principales estándares. Las especificaciones sólo sientan las bases y las reglas de esta arquitectura de componentes.

Mediante estas especificaciones, se establecen una serie de convenios gracias a los cuales, los desarrolladores de las distintas plataformas consiguen que sus SW sean accesibles por usuarios de cualquier plataforma, aunque no sea la de desarrollo del servicio. En el caso de que este desarrollador sea una empresa que centre sus negocios en los SW, lo que consigue es un abanico más amplio de clientes.

De la misma forma, los usuarios potenciales de SW, no tienen que restringir su búsqueda de servicios para sus negocios solamente dentro de los desarrollos realizados para su propia plataforma de trabajo, sino que gracias a esta interoperabilidad multiplataforma, lo único que tendrán que hacer será respetar las reglas de comunicación para poder utilizar cualquier SW, sea cual sea su plataforma de origen.

Para el diseño y creación de SW, cada plataforma dispone de uno o varios entornos tecnológicos que, mediante sus propios lenguajes y aplicaciones de desarrollo, permiten crear SW de acuerdo a las especificaciones de una manera más o menos automatizada. Las distintas plataformas, siguiendo su propia filosofía de trabajo, diseñan implementaciones de los SW, aunque a efectos prácticos lo realmente importante es que estas implementaciones tengan en cuenta y respeten los estándares básicos de la Web sobre los que se sustentan los SW, a fin de que exista uniformidad de cara a los usuarios.

Actualmente, algunas de las plataformas que se han apuntado a la carrera del desarrollo y explotación de los SW son Sun Microsystems [22] y Microsoft. [16].

5.2. Microsoft .NET

5.2.1. Infraestructura de .NET para los Servicios Web

5.2.1.1. Espacios de nombres de Microsoft .NET

Particularmente, .NET tiene su propio soporte para los SW. Este soporte esta constituido por una serie de clases distribuidas en distintos espacios de nombres, todos ellos pertenecientes al paquete *System* (Figura 46).

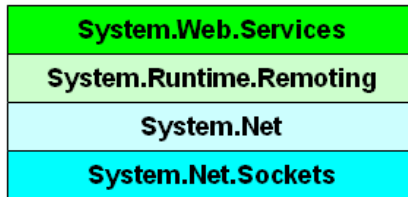


Figura 46. Pila tecnológica de Microsoft.NET

El espacio de nombre *System.Net.Sockets* contiene las clases que ofrecen el soporte para la comunicación mediante el protocolo TCP/IP. Y en este espacio de nombres están también las clases necesarias para incluir soporte para mantener la sesión de una comunicación mediante *sockets*.

Es muy común realizar la comunicación mediante el protocolo de transporte HTTP, ya que es el protocolo más utilizado para comunicarse con los servidores Web. HTTP también permite la comunicación con otras aplicaciones a través de *firewalls* corporativos. Todo esto viene soportado por las clases contenidas por el espacio de nombres *System.Net*.

Podemos implementar aplicaciones basadas en objetos remotos. En este caso, hay muchos inconvenientes relacionados con la identidad que tiene el objeto y el formato de representación del mismo. El formato del objeto remoto puede ser binario, o quizá una representación serializada del objeto en XML. El *framework* de .NET ofrece soporte a este tipo de aplicaciones mediante las clases contenidas en *System.Runtime.Remoting*.

5.2.1.2. Proveedores de Servicios Web en .NET

La infraestructura necesaria para soportar a los SW viene en .NET de la mano del uso de *Microsoft Internet Information Server* (IIS) y *Microsoft*

ASP.NET, en una máquina en la que tengamos instalado *Microsoft Windows*, y que sea nuestro proveedor de SW. Este proveedor de servicios, entre otras cosas dispondrá de un protocolo de escucha. Para los SW desarrollados bajo la plataforma .NET, el protocolo deberá ser HTTP.

Un proveedor de SW, debe tener la posibilidad de albergar varios SW, y por ende, redireccionar las peticiones que reciba al servicio en concreto al que vayan dirigidas. De la misma forma, debe poder ofrecer algo de seguridad a nivel de protocolo en los accesos para evitar que usuarios o consumidor de los servicios desconocido tenga acceso a los mismos.

El servidor Web IIS, provee todas estas propiedades necesarias para los SW mediante las siguientes características:

- IIS es un “*HTTP listener*”.
- IIS actúa como una pasarela para la implementación de varios SW que podrá albergar mediante su arquitectura basada en *Internet Server Application Programming Interface* (ISAPI).
- IIS provee una infraestructura significativa para la seguridad.

El IIS invoca a los SW en representación del cliente mediante el uso de diferentes opciones. Un servidor Web puede iniciar una aplicación *Common Gateway Interface* (CGI); iniciar un interprete de script, tal y como se hace con *Microsoft Active Server Pages* (ASP), o invocar una aplicación ISAPI.

Cuando el IIS trabaja junto con el CLR (*Common Runtime Language*), usa un filtro ISAPI para interceptar las peticiones dirigidas las páginas con extensión *.asmx*, e inicializar así el *run-time* del servidor, que ejecuta el código del SW implementado por *.NET Framework*.

5.2.2. Servicios Web en .NET

5.2.2.1. Publicación, Descubrimiento e Integración de SW en .NET

Como ya se comentó, mediante el estándar UDDI, podemos buscar los SW que necesitemos para nuestros negocios, podemos publicar los SW

que desarrollemos en el registro UDDI, y actualizar esos servicios que publiquemos para adaptarlos a nuevas versiones que satisfagan las necesidades de mercado, o por el contrario, eliminarlos por haber quedado totalmente obsoleta su funcionalidad, y haber perdido por ello todo el interés que en su día pudieron despertar.

Microsoft .NET hace su propuesta particular en este sentido. En UDDI, la forma de invocar las todas las operaciones de las APIs, es mediante documentos en XML. Las funciones están representadas por etiquetas mediante las cuales expresamos nuestra intención de emprender alguna acción. Dentro de estos documentos en XML, incluiremos también los argumentos necesarios para que se lleve a cabo la ejecución de estas operaciones. En muchas ocasiones generar estos documentos puede ser bastante complejo, y puede suponer una labor casi titánica. Es por esto que en .NET se ha implementado una serie de operaciones, para que podamos invocar a esas funciones de una manera sencilla y clásica, mediante los lenguajes de programación. De esta forma, cuando queramos realizar accesos a los registros UDDI, podremos crear esos mensajes SOAP en XML mediante una serie de funciones de una manera muy sencilla.

Como podíamos ver en la Figura 34, la información que hay en los registros UDDI se almacena mediante una serie de estructuras que tratan de organizar la información según su naturaleza, importancia o nivel de detalle. La manipulación de estas estructuras pueden ser también bastante compleja, ya que al igual que pasaba con las funciones, también se representan mediante etiquetas y atributos. Para hacer más sencilla la tarea de crear estas estructuras a fin de enviarlas a los registros UDDI, o de extraer la información contenida en una estructura que el registro nos mande como respuesta a una petición, Microsoft ha creado una serie de contenedores, que residen dentro del espacio de nombres *Microsoft.UDDI*. Las estructuras de datos de UDDI se representan mediante clases de .NET con sus correspondientes propiedades.

En general, la propuesta de Microsoft para UDDI consiste en el *Microsoft UDDI SDK*, que dispone de una serie de espacios de nombre (Tabla 10) que son los que implementan toda la lógica que da soporte a UDDI.

Espacio de Nombre	Descripción
Microsoft.UDDI	Contiene las clases que representan a las APIs de SOAP para UDDI
Microsoft.UDDI.Api	Contiene las clases básicas y de utilidades para otros espacios de nombres
Microsoft.UDDI.Authentication	Contiene una clase que representa al token de autenticación, además de clases para conseguirlo del registro y desecharlo cuando ya no sea necesario
Microsoft.UDDI.Binding	Contiene clases que representan elementos que son plantillas para asociaciones
Microsoft.UDDI.Business	Contiene clases que representan a las entidades de negocio
Microsoft.UDDI.Service	Contiene clases que representan a los servicios de negocio
Microsoft.UDDI.ServiceType	Contiene clases que representan a los elementos tModel, tModelInstanceDetail, y tModelInstanceCollection.

Tabla 10. Espacios de nombres del Microsoft UDDI SDK

5.2.2.2. Publicación de un Servicio Web

Cuando queremos publicar un SW en el registro UDDI, tenemos que seguir los pasos que dicta UDDI. El proceso de publicación se divide en una serie de pasos. En el primero de ellos tenemos que conseguir el *token* de autenticación que nos permitirá invocar a todas las funciones de la API de publicación. La función que nos permite obtener el *token* es *get_authToken*. La información de autenticación se provee mediante la clase **Publish**(Figura 47).

```
Publish.Url = strPublishUrl;
Publish.User = strUID;
Publish.Password = strPassword;
```

Figura 47. Clase que soporta la información de autenticación.

En caso de que se intente realizar alguna acción de publicación sin haber obtenido previamente el *token* de autenticación, las clases contenedoras realizarán esta petición de forma automática. Tras obtener el *token*, tenemos que publicar los *tModels* que soporta el SW mediante la función de .NET *save_tModel* (Figura 48).

```
TModel tModel = new TModel();
tModel.Name = "Mi tModel";
tModel.Descriptions.Add("tModel para un SW de prueba");
tModel.OverviewDoc.OverviewURL =
"http://www.miWeb.es/overview.htm";
tModel.OverviewDoc.Descriptions.Add("El tModel de mi SW de ejemplo");
SaveTModel saveTModel = new SaveTModel();
saveTModel.TModels.Add(tModel);
TModelDetail tmd = saveTModel.Send();
```

Figura 48. Salvando un tModel de nuestro SW.

Antes de publicar el SW, necesitamos publicar una estructura *businessEntity* a la que asociaremos el servicio (Figura 49).

```
Contact contact = new Contact();
contact.PersonName = "Perico Palotes";
contact.Emails.Add("Peri.Palo@miCorreo.es");
contact.Phones.Add("92622222");
contact.Descriptions.Add("en", "Administrador del sitio Web");
BusinessEntity businessEntity = new BusinessEntity();
businessEntity.Name = "Servicios Web a Domicilio";
businessEntity.Descriptions.Add("en", "Distribuidora de Servicios");
businessEntity.Contacts.Add(contact);
SaveBusiness saveBusiness = new SaveBusiness();
saveBusiness.BusinessEntities.Add(businessEntity);
BusinessDetail bd = saveBusiness.Send();
```

Figura 49. Registro de la información para el businessEntity.

El siguiente paso, consiste en publicar una estructura *businessService*, que permite a los desarrolladores asociar el *businessEntity* al servicio que deseamos publicar (Figura 50).

```
BindingTemplate bindingTemplate = new BindingTemplate();
...
// información de asociación
...
BusinessService businessService = new BusinessService();
businessService.BusinessKey = businessKey;
businessService.Name = "ServicioDeEjemplo";
businessService.BindingTemplates.Add(bindingTemplate);
SaveService saveService = new SaveService();
saveService.AuthInfo = "udditest";
saveService.BusinessServices.Add(businessService);
saveService.BusinessServices[0].BindingTemplates.Add(bindingTemplate);
ServiceDetail sd = saveService.Send();
```

Figura 50. Publicación de un businessService.

El último paso es asociar el *tModel* con el *businessService*, dando la información necesaria para la asociación. Para tal fin tenemos que publicar un *bindingTemplate* (Figura 51).

```
BindingTemplate bindingTemplate = new BindingTemplate();
bindingTemplate.ServiceKey = serviceKey;
bindingTemplate.Descriptions.Add("Plantilla de asociación");
AccessPoint accessPoint = new AccessPoint (URLTypeEnum.Http,
    "http://www.miWeb.es/servicios/MiServicio.asmx");
bindingTemplate.AccessPoint = accessPoint;
TModelInstanceInfo tModelInstanceInfo = new TModelInstanceInfo();
tModelInstanceInfo.TModelKey = tModelKey;
tModelInstanceInfo.Descriptions.Add("Información del tModel");
bindingTemplate.TModelInstanceDetail.TModelInstanceInfos.Add(tModelInstanceInfo);
SaveBinding saveBinding = new SaveBinding();
saveBinding.BindingTemplates.Add(bindingTemplate);
BindingDetail bd = saveBinding.Send();
```

Figura 51. Creando una plantilla businessTemplate para asociar el tModel y el businessService.

5.2.2.3. Búsqueda de Servicios Web

Una vez que hemos desarrollado un SW y lo hemos publicado sólo falta que los posibles clientes lo encuentren. De la misma forma, en las empresas surgen necesidades en distintos procesos que encuentran solución en los SW. Ambas partes tienen problemas distintos que encuentran su solución en UDDI. Este estándar, además de soportar la tarea de publicación de los servicios, también nos permite localizar servicios en el registro de UDDI atendiendo a distintos criterios de búsqueda.

A continuación, describimos como podemos localizar los SW en .NET. En primer lugar, cuando deseamos buscar un SW, localizaremos el negocio o área de negocio al que pertenece el servicio o sobre el que gira la prestación que deseamos obtener (Figura 52).

```
FindBusiness fb = new FindBusiness();
fb.Name = "Servicios Web a Domicilio";
BusinessList bl = fb.Send();
```

Figura 52. Búsqueda del negocio.

Una vez que ya tenemos la lista de los negocios que satisfacen nuestros criterios de búsqueda, ya podemos navegar por aquellos servicios cuyas

entidades de negocio nos permiten recuperar información de asociación. Una vez conseguida la plantilla de asociación que deseamos, conseguir la URL del SW es una operación trivial (Figura 53).

```
foreach (BusinessInfo bi in bl.BusinessInfos)
{
    foreach(ServiceInfo si in bi.ServiceInfos)
    {
        FindBinding fb = new FindBinding();
        fb.ServiceKey = si.ServiceKey;
        BindingDetail bindDetails = fb.Send();
        foreach (BindingTemplate bt in bindDetails.BindingTemplates)
        {
            if(bt.TModelInstanceDetail.TModelInstanceInfos[0].TModelKey == tModelKey)
            {
                strURL = bt.AccessPoint.Text;
                goto found;
            }
        }
    }
}
found:
...
```

Figura 53. Obtención información de los servicios.

Cuando disponemos de la plantilla de asociación, lo único que nos queda es invocar al SW, ya que disponemos de la URL que nos permite el acceso (Figura 54).

```
NorthwindEFTService eft = new NorthwindEFTService();
eft.Url = strURL;
XmlNode balances = eft.GetCurrentBalances("1XF99-S");
```

Figura 54. Recuperamos la información de vinculación para invocar al SW.

5.2.2.4. Métodos como Servicios

Cuando desarrollamos un SW, quizá no nos interese que todos los métodos que incluimos en nuestras clases (nos referimos a las clases que implementan la lógica del SW) sean métodos públicos. Para ello, Microsoft.NET, ha desarrollado un mecanismo mediante el cual, decimos explícitamente que métodos queremos publicar, estos serán los llamados métodos Web.

La forma en la que distinguiremos estos métodos será poniendo sobre la cabecera del método el atributo *WebMethod*. Con esto le decimos al

motor en tiempo de ejecución de ASP.NET que ponga el método como accesible públicamente. El atributo *WebMethod* tiene una serie de propiedades (Tabla 11) con las que podemos configurar los métodos Web para que aporten más información, o incluso para que utilicen alguna característica especial.

Además del atributo *WebMethod*, el marco de páginas ASP.NET proporciona otro atributo, el *WebService*, que se utiliza para modificar el SW como un todo. Cualquier cambio que se realice en sobre este atributo, se reflejará en el documento WSDL. Al igual que ocurría con *WebMethod*, *WebService* también dispone de una serie de propiedades (Tabla 12) para configurar el servicio.

Propiedad	Descripcion
<i>BufferResponse</i>	Especifica si se debe guardar en un buffer la respuesta del cliente
<i>CacheDuration</i>	Especifica en segundos, que se puede mantener una respuesta en memoria en el servidor Web para una determinada petición. Por defecto vale 0.
<i>Description</i>	Especifica el valor del elemento <i>description</i> en cada uno de los elementos <i>operation</i> en cada definición del tipo en el documento WSDL generado por ASP.NET
<i>EnableSession</i>	Indica si estarán disponibles los servicios de estado de sesión de ASP.NET para la implementación del método
<i>MessageName</i>	Especifica el nombre del método que expone el SW. Concretamente, indica el nombre del elemento en el cuerpo del mensaje de SOAP que contiene los parámetros así como el sufijo de la acción SOAP. También especifica el prefijo de los nombres de los elementos <i>message</i> , <i>input</i> y <i>output</i> del documento WSDL generado por ASP.NET
<i>TransactionOption</i>	Especifica el soporte transaccional que debería proporcionar para la implementación del método. El método puede servir sólo como raíz de una transacción y puede no participar en la transacción de quien llama.

Tabla 11. Propiedades del atributo *WebMethod*.

Propiedad	Descripción
<i>Description</i>	Especifica el elemento <i>description</i> bajo el elemento <i>service</i> del documento WSDL generado por ASP.NET.
<i>Name</i>	Especifica el nombre del elemento <i>service</i> en el documento WSDL generado por ASP.NET. También especifica el prefijo de los nombres de los elementos <i>portType</i> , <i>binding</i> y <i>port</i> .
<i>Namespace</i>	Especifica los nombres del espacio objetivo del documento WSDL, así como el documento esquema que define las estructuras para la codificación de los parámetros en el cuerpo de un mensaje SOAP. También especifica el prefijo del espacio de nombres para el esquema que contiene cualquier tipo predefinido en el SW y el valor de la acción de SOAP.

Tabla 12. Propiedades del atributo *WebService*.

5.2.2.5. Proxy para un Servicio Web

Un usuario o consumidor de SW, debe ser capaz de crear mensajes SOAP para realizar las peticiones a los SW, y de la misma forma, debe poder analizar los mensajes que le lleguen de estos. Diseñar un código que realice estas tareas puede ser complejo, tedioso y por la dificultad de la tarea, es susceptible de incluir errores. Por eso, toda la lógica que comprende la generación de mensajes para el SW, y el análisis de estos se encapsula dentro de una clase que hará de *proxy* para el servicio.

Por definición, un *proxy* es una entidad que actúa de intermediario para otras entidades. A efectos de nuestro problema, para nosotros, como clientes o consumidores de un SW, un *proxy* tendrá la misma interfaz que la del SW al que queremos acceder.

Una de las ventajas que nos ofrece el uso de este tipo de elementos, es que los desarrolladores pueden trabajar con interfaces fuertemente tipadas en lugar de manejar directamente mensajes de texto, que es lo que son los mensajes SOAP en XML que viajan entre el cliente y el SW. Podemos generar de manera sencilla un *proxy* a partir de la descripción WSDL de un servicio, que es un archivo en XML muy fácilmente procesable.

Visual Studio .NET aporta una herramienta para la generación de *proxies* a partir de la descripción WSDL de un SW. Esta herramienta que automatiza y facilita la tarea de generación de un *proxy* no es otra que el programa *wSDL.exe*. Como se puede observar en la Figura 55, la sintaxis de este programa está basada en opciones y una URL. La aplicación *wSDL.exe* admite para la generación del *proxy* tanto especificaciones en

WSDL, como descripciones en XSD y archivos *DISCO*. El parámetro URL del comando (Figura 55) correspondería a uno de los tipos de descripciones que acabamos de citar.

```
wSDL [opciones] {URL/Path}
```

Figura 55. Sintaxis de wsdl.exe

Las opciones son las siguientes:

- */d:dominio* → Nombre de dominio para la autenticación al conectarse.
- */extendednaming* → Permite el uso de nombres extendidos cuando generamos clases de conjuntos de datos.
- */l:lenguaje* → Especifica el lenguaje en el que se desea generar el *proxy*. Los posibles valores son: CS para C#, VB para Visual Basic o JS para Microsoft JScript.
- */n:espacio de nombres* → Espacio de nombres por defecto para el *proxy*.
- */outnombre de fichero* → Fichero que contendrá el código.
- */u:nombre de usuario* → Nombre de usuario requerido para la conexión a un servidor que requiera autenticación.
- */p:clave* → Clave requerida para la conexión a un servidor que requiera autenticación.
- */protocol:protocolo* → Protocolo que implementará. Puede ser SOAP, HttpGet, HttpPost o cualquier otro protocolo especificado en el archivo de configuración *Web.config*.
- */server* → Genera una clase abstracta para el SW basada en los contratos. Por defecto genera las clases para el cliente *proxy*.

5.2.3. Seguridad en los Servicios Web

Dentro de la seguridad en los SW, los tres aspectos que se tratan más en profundidad en Microsoft son la autenticación, la autorización y las comunicaciones seguras.

La autenticación de los clientes o consumidores de los SW, se lleva a cabo mediante el sistema operativo Windows de Microsoft y el *Microsoft Internet Information Server* (IIS). El *Microsoft .NET Framework* y el *common language runtime* asisten en la tarea de la autorización. La seguridad en las comunicaciones se lleva a cabo mediante el cifrado total o parcial de los mensajes que se intercambian entre los consumidores de los SW y los propios SW.

Autenticación

La autenticación es uno de los primeros pasos que hay que dar en la implementación de la seguridad. Los mecanismos que Microsoft aporta para ello son:

- *Autenticación IIS*. El IIS ofrece los siguientes mecanismos de autenticación:
 - Autenticación *básica*.
 - Autenticación *asimilada*.
 - Autenticación *integrada en Windows*.
- *Autenticación basada en ASP.NET*. ASP.NET tiene dos mecanismos de autenticación que no existían en la anterior versión del lenguaje:
 - Autenticación basada en *Formularios*.
 - Autenticación basada en *“Passport”*.
- *Autenticación basada en Formularios*. En este mecanismo, las peticiones no autenticadas son redirigidas a un formulario HTML mediante HTTP. Este formulario es rellenado por el usuario con las credenciales y devuelto, de forma que si la aplicación Web autentica esta información, se emite un formulario con las credenciales o una clave. Las siguientes peticiones que se envíen a la aplicación Web, incluirán este formulario en la cabecera, de forma que estas quede autenticadas automáticamente.

- *Autenticación basada en "Passport"*. Este es un mecanismo de autenticación centralizado. Tanto la autenticación basada en formularios como la basada en "Passport", no son métodos fáciles de utilizar en los SW, ya que la forma de autenticación que tienen es mediante una pantalla dirigida al usuario final mediante la cual se identifica, y como la invocación a los SW se hace automáticamente y no manualmente por el usuario final, el control de este método de autenticación se hace difícil. Por esta razón no es recomendable utilizar este método de autenticación.
- *Autenticación mediante cabeceras SOAP específicas*. En caso de que no queramos utilizar los mecanismos de seguridad incluidos, podemos implementar nosotros mismos nuestro propio mecanismo. O si no queremos pasar las credenciales como atributos en los métodos del SW, tendremos que utilizar otro mecanismo para pasar las credenciales de autenticación. Una forma de realizar esto, son las cabeceras de los mensajes SOAP, ya que para el consumidor no es tarea difícil incluir esa información en las cabeceras, y para el propio SW tampoco es complejo recuperar esa información para autenticar al cliente.

Autorización

El entorno ASP.NET, el *.NET Framework* y la plataforma Windows proveen diferentes técnicas para el acceso autorizado a los recursos del sistema. Los recursos accesibles serán la intersección de los recursos autorizados por los siguientes mecanismos:

- *Sistema de seguridad de Windows NT*. Las características de seguridad de Windows están basadas en la identidad del usuario para prevenir los accesos no autorizados a los recursos del sistema. Esas características de seguridad son la autenticación del usuario y el control de acceso basado en objetos. Los administradores del sistema, pueden crear listas de control de acceso discrecional o listas DACL, que controlan el acceso a los recursos u objetos de una red.
- *Seguridad basada en roles*. En este modelo de seguridad, la identidad del usuario no es importante. Lo interesante son los

roles que el usuario puede asumir. El sistema de seguridad basado en roles, utiliza los roles asociados al usuario para tomar decisiones sobre las autorizaciones para la seguridad.

- *Código para la seguridad en el acceso.* Este mecanismo de seguridad es similar a la seguridad basada en roles, ya que el código para la seguridad en el acceso necesita que el usuario sea autenticado en primer lugar para que el código para la seguridad en el acceso pueda operar.

Las aplicaciones ASP.NET, están diseñadas para que se ejecuten en una carpeta determinada. Para acceder a esta aplicación, usaremos el *Identificador Universal del Recurso* o URI.

Métodos de Cifrado

Hasta ahora, la autenticación y la autorización nos permiten prevenirnos de los accesos no autorizados por parte de los usuarios a los recursos del sistema. Pero ninguno de estos mecanismos evita que los datos intercambiados entre un SW y el consumidor del mismo sean interceptados. Mediante el cifrado de los datos, evitamos que la información contenida en los mensajes pueda ser interceptada.

En el cifrado, tenemos que seguir algún criterio para decidir que datos serán codificados, ya que cifrar es un proceso costoso, y por ello tenemos que seleccionar cuidadosamente las comunicaciones que tienen que ser cifradas. Podemos considerar las siguientes alternativas:

- Cifrar el mensaje completo. Esto es muy sencillo de hacer, pero decae mucho el rendimiento, ya que es poco probable que todas las comunicaciones tengan que ser privadas.
- Cifrar solamente el cuerpo de los mensajes. Esta opción es menos costosa, computacionalmente hablando, pero aun así, puede haber mucha información que no tenga que ser cifrada.
- Cifrar solamente las cabeceras de los mensajes. En los SW, la autenticación del usuario que envía el mensaje, va en la cabecera de los mensajes SOAP. Teniendo en cuenta que esta información

es poco deseable que sea descubierta, resulta una buena opción el cifrar solamente la cabecera de los mensajes.

- Cifrar solamente los mensajes seleccionados. El mayor esfuerzo que se realiza aquí es por parte del desarrollador, que busca un equilibrio entre seguridad y rendimiento. En esta opción, sólo se cifran los mensajes que deben ser privados.
- No cifrar nada. Si los datos que van a intervenir en las comunicaciones no tienen por que ser privados, no se cifra nada para que el rendimiento no se vea afectado.
- Particionar el SW. La idea es separar la interfaz del servicio en grupos, uno en los que los mensajes tengan que ser cifrados y otro grupo en el que no se requiera que los mensajes sean cifrados. De esta manera, tendríamos dos SW, uno en el que siempre cifraríamos los mensajes y otro en el que nunca los cifraríamos. De esta forma nos aseguramos de que sólo codificaremos lo que no debe ser interceptado, evitando así pérdidas innecesarias en el rendimiento.

Algunas de las maneras de cifrar las comunicaciones son las siguientes:

- *Secure Socket Layer (SSL)*. Es una forma sencilla de codificar todas las comunicaciones entre el SW y el usuario que lo invoca.
- *Extensiones SOAP específicas*. Para un control más detallado, se pueden implementar extensiones SOAP que nos permitan cifrar sólo los datos más críticos. De esta forma, sólo y únicamente cifraremos los datos que realmente sean privados, aumentando de esta forma notablemente el rendimiento del protocolo SSL.

5.3. J2EE

J2EE es la plataforma de Sun Microsystems [22] para Java [9] en la que se integran un conjunto de potentes tecnologías para el desarrollo Web. La plataforma J2EE dispone de un conjunto de APIs importantes para el desarrollo de SW:

- JAXP

- JAXB
- JAX-RPC
- JAXM
- JAXR

5.3.1. JAXP

Esta API se encarga del procesamiento de documentos SOAP mediante SAX y DOM.

SAX (*Simple API for XML Parsing*)

SAX lee el documento de principio a fin, y cuando encuentra una construcción sintáctica, lo notifica a la aplicación que lo ejecuta mediante la interfaz *ContentHandler*, por lo que se dice que el funcionamiento de esta API está dirigido por eventos.

El *DefaultHandler* es la implementación por defecto del *ContentHandler*, y sus métodos no tienen efecto alguno. Por esto, para hacer uso de SAX, tendremos que codificar una especificación de *DefaultHandler* en la que implementemos los métodos *startElement(...)*, *characters(...)* y *endElement(...)* entre otros.

Para procesar un documento mediante SAX, necesitaremos un parser SAX que obtendremos mediante la *SAXParserFactory*.

DOM (*Document Object Model*)

Representa el documento XML mediante una estructura arborescente (Figura 56), y permite su manipulación con clases y operaciones que manipulan este árbol conceptual.

XML Namespaces

Los nombres de esquema utilizados en un documento XML son únicos a fin de evitar la ambigüedad en el significado de las etiquetas. Sin embargo, si un documento XML hace referencia a múltiples esquemas, existe la posibilidad de que dos o más de estos contengan el mismo nombre, para lo que distinguimos en el documento un esquema de otro asignándoles un *namespace* a cada uno.

Podemos decirle a un parser SAX o DOM que utilice esquemas mediante el método `setNamespaceAware(bolean)` de la instancia de la `ParserFactory` que estemos utilizando.

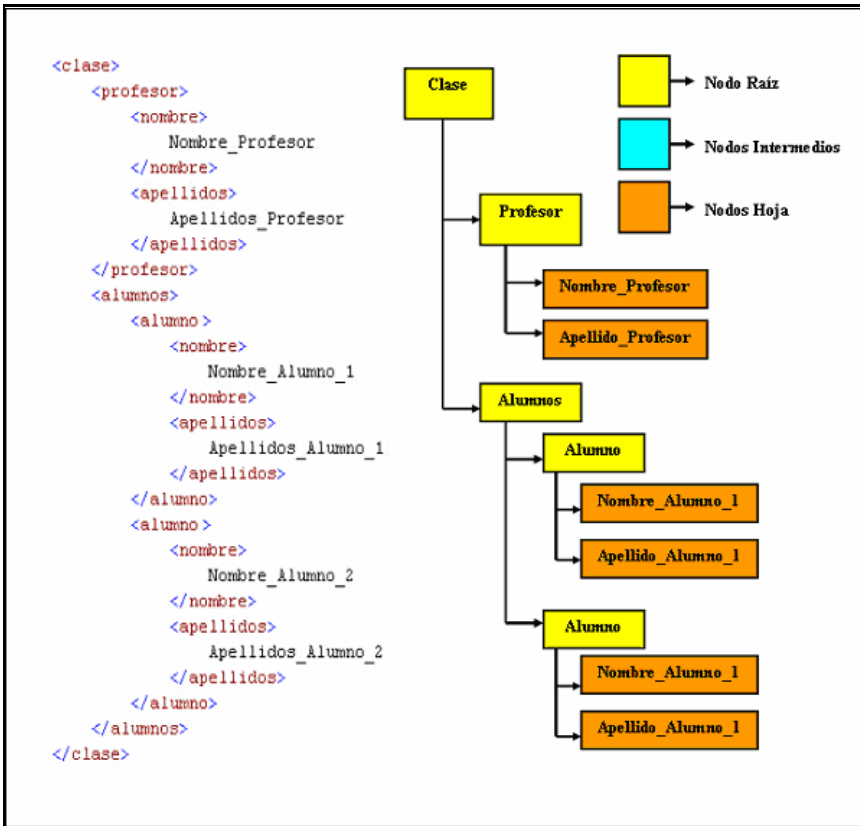


Figura 56. Documento XML (izquierda) y su representación arborescente (derecha)

XSLT (*XSL Transformations*)

JAXP utiliza XSLT, un *sub-lenguaje* de XML a través del paquete `javax.xml.transform`, que permite usar un transformador XSLT para aplicarlo sobre un documento XML y obtener otro tipo de documento.

Los transformadores son instancias de `TransformerFactory`, que devuelve una instancia del transformador mediante el método `newTransformer`.

Si tenemos que obtener un documento HTML a partir de un documento XML, necesitaremos en primer lugar el propio documento XML y la página de estilo XSL, creando por último el correspondiente transformador. El transformador volcaría el documento resultante en un archivo de salida como se puede observar en la Figura 57.

```
TransformerFactory tFactory = TransformerFactory.newInstance();
FileOutputStream salida = new FileOutputStream("fichero.html");
Transformer transformador = tFactory.newTransformer(new StreamSource("hojaEstilo.xsl"));
transformador.transform(new StreamSource("fichero.xml", new StreamResult(salida));
```

Figura 57. Transformación de un fichero en XML de acuerdo a una hoja de estilo XSL

5.3.2. JAXB (*Java Architecture for Java Binding*)

Permite generar clases Java a partir de esquemas XML. Como parte de este proceso, JAXB proporciona métodos para transformar (*unmarshal*) una instancia de un documento XML en un árbol de objetos Java, y viceversa, transformar el árbol de objetos en un instancia de documento XML.

JAXB ofrece un método rápido para ligar un esquema XML a una representación de código Java, facilitando la incorporación de datos y funciones procedentes de Java a las aplicaciones XML sin necesidad de tener grandes nociones de XML.

El proceso de ligado (*binding*) en JAXB consta de los siguientes pasos:

1. Generar clases de un esquema XML, y compilar las clases generadas.
2. Transformar los documentos XML obtenidos de acuerdo al esquema, de modo que se genere un árbol de objetos Java que represente la estructura y contenido de los documentos XML originales.
3. La transformación del paso anterior puede hacer que se validen los documentos XML antes de generar el árbol de objetos. Este proceso se conoce como "*unmarshall*"

4. La aplicación cliente puede modificar los datos XML representados por un árbol de contenidos por medio de interfaces generadas por el compilador de enlaces (*binding compiler*).
5. El árbol de objetos es transformado a uno o más documentos XML. A este proceso se le llama “*marshall*”

5.3.3. JAX-RPC (*Java API for XML-based Remote Procedure Call*)

Esta es la API de Java utilizada para usar y desarrollar SW. Un SW basado en RPC (*RPC-based Web Service*) es una colección de procedimientos a los que un cliente puede llamar a través de Internet. El SW se publica (*deploy*) en un contenedor, como Tomcat o el que provee la plataforma J2EE.

Un SW puede ponerse a disposición de sus clientes potenciales describiéndose a si mismo mediante el documentos WSDL, que como ya se ha mencionado anteriormente es un documento XML que incluye el nombre del servicio, sus operaciones, parámetros y la URL donde el SW lee las peticiones (*requests*).

Interoperabilidad

Para que los SW desarrollados en Java contengan las características de interoperabilidad y multiplataforma, API JAX-RPC soporta los protocolos SOAP y WSDL.

Características Avanzadas

Con JAX-RPC se pueden enviar documentos completos o fragmentos, soporta manejadores para enviar una amplia variedad de tipos de mensajes, pudiendo enviar mensajes en un solo sentido (en vez de la forma *petición/respuesta*). Esto último es útil en los SW, ya que estos según el estándar, deben soportar varios tipos de mensajes o modos de comunicación (ver Sección 3.1.2.4.).

Uso de JAX-RPC

Mediante *wscompile* generamos un stub a partir del documento WSDL del SW, y *wsdeploy* genera un *proxy*. El stub lo usa el cliente, mientras que el tie lo utilizará el servidor.

Durante la ejecución, un sistema JAX-RPC convierte las llamadas a métodos que hace el cliente en mensajes SOAP que serán enviados al SW a través del *stub*. Aunque ya se explicó, los mensajes SOAP son secuencias de caracteres que conforman un documento en XML donde, siguiendo unas reglas de notación, (1) un cliente especifica la petición que desea realizar al SW, o bien (2) el SW retorna al cliente el resultado de la ejecución de un servicio solicitado. Y estas llamadas convertidas en mensajes SOAP son enviadas al servicio mediante un protocolo de transporte como HTTP, en el otro lado, JAX-RPC recibe el mensaje SOAP con la solicitud, lo traduce al método y lo invoca. Acto seguido, el SW devuelve el resultado al cliente siguiendo un proceso similar.

Creación de un Servicio Web

Un SW, constará básicamente de dos ficheros: una interfaz, que declara los procedimientos remotos del servicio y una clase que los implementará. Una vez desarrollada la interfaz y su implementación se generaran el *stub* y el *proxy* (de los que hablamos antes) junto con el resto de clases que pudieran ser necesarias, mediante una herramienta que automatiza el proyecto.

Los pasos finales son el empaquetamiento del SW en un fichero WAR su instalación (*deployment*) en el sistema para exponerlo a los clientes.

Escritura del código del cliente

El cliente usará el *stub* que se haya generado a partir de la clase *X*. El propósito del *stub* acceder a instancias de la interfaz de la clase *X* mencionada en el apartado anterior, y lanzar llamadas a los métodos remotos, que estarán contenidos en el fichero WSDL asociado.

Llamada de un método remoto

El JAX-RPC determina la ubicación del destinatario de la llamada (es decir, la ubicación del SW) a partir del documento WSDL.

5.3.4. JAXM (*Java API for XML Messaging*)

Esta API proporciona un mecanismo estándar para mandar documentos XML a través de Internet. Se basa en la especificación 1.1 de SOAP y en

SOAP con adjuntos. A veces es preferible utilizar JAXM en vez de JAX-RPC. Las características de JAXM que no tiene JAX-RPC son:

- Mensajeado asíncrono (en una sola dirección).
- Enrutamiento de un mensaje a más de un destinatario.
- Mensajeado fiable, como la garantía de entrega

La clase que encapsula un mensaje de este tipo es la *SOAPMessage*, que tiene siempre una parte SOAP (para cumplir con el estándar) y puede tener además una o más partes correspondientes a adjuntos:

- *SOAPEnvelope* es la parte SOAP, que debe tener un *SOAPBody* y puede tener un *SOAPHeader*, que puede contener una o varias cabeceras.
- El *SOAPBody* puede tener fragmentos XML (por ejemplo, el contenido del mensaje) o contenidos en otro formato debidamente referenciado.

5.3.4.1. Obtención de una conexión

El cliente JAXM necesita obtener bien una *SOAPConnection* o una *ProviderConnection*.

5.3.4.2. Obtención de una conexión punto a punto

La *SOAPConnection* se obtiene de la *SOAPConnectionFactory* con su método *createConnection()*. Normalmente, el código necesario para establecer la conexión estará incluido en el stub, que se habrá generado de forma automática o semiautomática con anterioridad.

5.3.4.3. Obtención de una conexión al proveedor de mensajes

La conexión con un proveedor de mensajes se realiza mediante una *ProviderConnection*, que es una conexión al proveedor de mensajes en vez de al destinatario especificado.

Esta *ProviderConnection* se obtiene con la clase *ProviderConnectionFactory* a través de su método *createConnection()*.

A su vez, la *ProviderConnectionFactory* se puede crear del contexto de ejecución, para lo que debería usarse JNDI, que nos permite conectarnos a la *Factory* preestablecida. El nombre JNDI deberá estar dado de alta en el servidor que se esté utilizando.

5.3.4.4. Creación de mensajes

La creación de los mensajes es muy sencilla, basta con utilizar la *MessageFactory*. Mediante el método *createMessage()* de la factoría obtenemos un *SOAPMessage*.

5.3.4.5. Adición de adjuntos

Tal y como se comentó, todo *SOAPMessage* necesita una parte SOAP (*SOAPPart*) que tiene un *SOAPEnvelope* con, a su vez, un *SOAPHeader* y un *SOAPBody*.

Para añadir la *SOAPPart* a un mensaje se debe crear un *SOAPHeaderElement* y un *SOAPBodyElement* al que se añade el XML (Figura 58).

```
SOAPPart sp = m.getSOAPPart(); //m es un SOAPMessage
SOAPEnvelope env = sp.getSOAPEnvelope();
SOAPBody body = env.getSOAPBody();
SOAPBodyElement be = body.addBodyElement
    (env.createName("text", "hotitems", "http://..."));
be.addTextNode("some-xml-text");
```

Figura 58. Código necesario para la adición de contenido al mensaje SOAP

Otra forma es añadir contenido pasándole un *javax.xml.transform.source*, que puede ser un *SAXSource* o un *StreamSource* (Figura 59).

```
SOAPPart sp = m.getSOAPPart(); //m es un SOAPMessage
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse("file://...xml");
DOMSource ds = new DOMSource(doc);
sp.setContent(sp);
```

Figura 59. Código necesario para la adición de contenido al mensaje SOAP

5.3.4.6. Adición de contenido a la parte *Attachment*

Un mensaje puede no tener adjuntos, pero si tiene alguno y no esta en formato XML, debe colocarse en la sección de adjuntos. Un adjunto podrá contener cualquier tipo de elemento.

Los adjuntos se manipulan con objetos de la clase *Attachment* como podemos ver en la Figura 60.

```
URL url = new URL("http://...");
DataHandler dh = new DataHandler(url);
AttachmentPart ap = m.createAttachment(dh); //m es un SOAPMessage
m.addAttachmentPart(ap);
```

Figura 60. Adición de un elemento como *Attachment* a un mensaje

Por supuesto, los *AttachmentPart* tienen más métodos para especificar su contenido.

5.3.4.7. Envío de mensajes

En conexiones punto a punto se usa el método *call* sobre la *SOAPConnection*:

```
SOAPMessage m = SOAPConnection.call(m, endpoint); //endpoint es una URL
```

Figura 61. Envío de un mensaje

Si se usan proveedores de mensajes se usa el método *send* sobre el *ProviderConnection* (ver Sección 4.3.4.3.).

```
pc.send(m); //pc es un proveedor de conexión
```

Figura 62. Envío de un mensaje a través de un proveedor de conexión

5.3.5. JAXR (*Java API for XML Registries*)

JAXR permite acceder a registros de negocios en Internet que, como ya decíamos con UDDI, vienen a ser como páginas amarillas.

5.3.6. Nociones avanzadas de JAXB

Tal y como se resumió en el principio, JAXB es la tecnología que permite generar clases Java a partir de esquemas XML. También permite obtener árboles de objetos a partir de documentos XML y viceversa.

Arquitectura de JAXB

Tal y como podemos ver en la Figura 63, JAXB se integra con otros elementos para realizar su función:

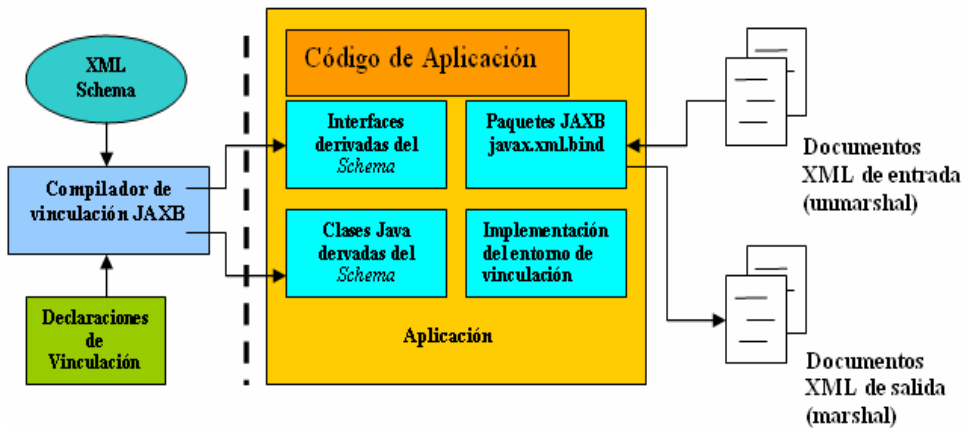


Figura 63. Esquema de funcionamiento del JAXB

A continuación, detallaremos algunos de los elementos mostrados en la Figura 63:

- *XML Schema*: Describe en XML las relaciones entre elementos, atributos y entidades de un documento XML. Sirven para definir una clase de documentos XML que se adhieran a un conjunto específico de reglas estructurales y restricciones de datos. En JAXB, el documento XML que contiene los datos y que está restringido por un *XML Schema* se le llama “instancia del documento”; los datos y la estructura dentro de una “instancia del documento” se conocen como “árbol de contenidos” o *content tree*.

- *Binding declarations*: El *JAXB binding compiler* enlaza clases y paquetes Java a código XML según un conjunto de reglas predefinido. JAXB permite sin embargo la modificación de estas reglas.
- *Binding compiler*: Transforma un esquema XML a un conjunto de clases Java.
- *Binding Framework Implementation*: API que da interfaces para realizar el *marshall*, el *unmarshall* y la validación de los documentos XML. Las clases e interfaces que realizan estas operaciones tan importantes se encuentran en el paquete *javax.xml.bind*. El punto principal de entrada al *binding framework* es la clase *JAXBContext*:

```
JAXBContext jc = JAXBContext.newInstance(contextPath);
```

Figura 64. Objeto JAXBContext

El *contextPath* contiene una lista de nombres de paquetes que contienen interfaces derivadas del esquema (realmente son las interfaces generadas por el *JAXB binding compiler*).

- *Schema-Derived Classes*: Clases que generan el *JAXB compiler*.
- *Application Code*: Aplicación Java cliente que usa el binding framework para hacer unmarshal de datos XML, validar y modificar objetos en XML. Esta aplicación es la aplicación, que hará esto y muchas otras cosas más.
- *XML Input Documents*: Proceso de *unmarshalled*.
- *XML Output Documents*: Proceso de *marshalled out*.

Con todo esto, el proceso de enlazado con JAXB estaría compuesto por los siguiente pasos:

1. Generar clases a partir de un esquema XML.
2. Compilar las clases.

3. Hacer *unmarshal*, obteniendo los objetos.
4. Generar el árbol de contenidos (árbol de objetos).
5. Validación (opcional) del XML antes de obtener el árbol.
6. Procesar contenido, modificando el XML a través de los objetos.
7. Hacer *marshal* obteniendo XML desde los objetos.

5.3.6.2. Esquemas XML

Los esquemas XML se utilizan para describir los elementos, atributos, entidades y relaciones posibles en un documento XML. Un documento XML con datos que esté restringido por un esquema se llama “instancia de documento”, mientras que la estructura y datos del documento constituyen su árbol de contenido.

5.4. IBM WebSphere

5.4.1. ¿Qué es WebSphere?

WebSphere de IBM constituye una completa plataforma para aplicaciones de comercio electrónico. WebSphere es un servidor de aplicaciones que ejecuta aplicaciones de negocio, ofreciendo un completo soporte para la plataforma J2EE y los estándares de los SW. Ofrece un lugar donde ubicar la lógica de presentación y de negocio de las aplicaciones, integrándose también con la infraestructura computacional existente en las empresas.

De acuerdo con la Figura 65, el servidor de aplicaciones es realmente el eje de toda la infraestructura computacional del comercio electrónico. En casi todas las topologías, el servidor Web se sitúa entre el navegador y el servidor de aplicaciones. El contenido estático se suele obtener generalmente del mismo servidor Web, mientras que las peticiones de contenido dinámico son directamente redirigidas al servidor de aplicaciones. Tal y como se observa, el contenido puede ser almacenado de manera temporal para reducir tanto el tráfico en la red como el tiempo de carga. Para llevar a cabo su trabajo, los servidores de aplicaciones pueden implementar determinadas interacciones con otros elementos tales como bases de datos o sistemas heredados. De hecho, la presencia de estos sistemas heredados es totalmente necesaria para implementar parcial o totalmente las soluciones desarrolladas.

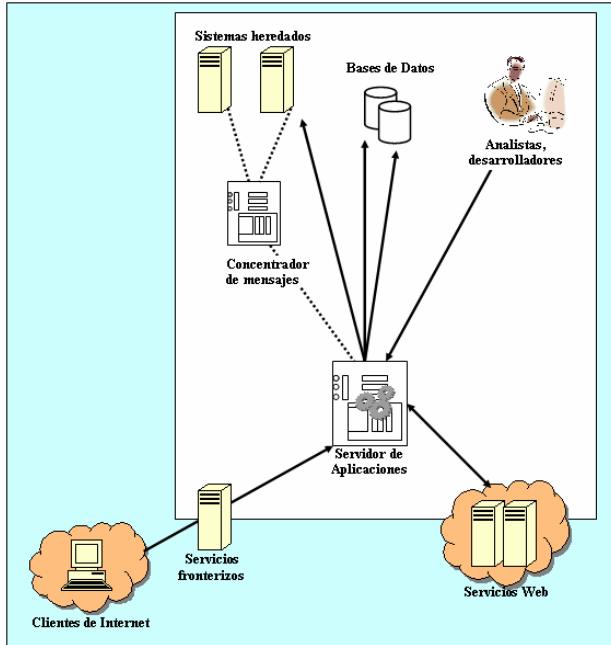


Figura 65. Integración del Servidor de Aplicaciones

Tanto desarrolladores como analistas, necesitan acceder a las aplicaciones, tanto para su desarrollo y mantenimiento como para el análisis de los procesos de negocio y la gestión de las políticas que se están ejecutando.

Como puede observarse, el centro de todas estas tareas es el servidor de aplicaciones. El servidor de aplicaciones utiliza como mecanismo de comunicación el servicio de mensajería de Java (*Java Messaging Service, JMS*) además de otras características propias de WebSphere 5.0. Con esto, se pueden abordar los modelos de programación síncronos y asíncronos, tan necesarios para la construcción de las aplicaciones de próxima generación.

5.4.2. WebSphere: Una familia de productos

La plataforma WebSphere está constituida por un conjunto de productos que ofrecen una amplia gama de soluciones que nos permiten transformar los negocios tradicionales en negocios electrónicos, habiendo sus miras a un nuevo mercado.

IBM WebSphere divide sus productos en tres grupos: *Foundation and tools*, *Reach and user experience* y *Business integration* (Figura 66). El primer grupo, *Foundation and tools*, consta de los productos que forman la infraestructura básica de la plataforma (como *WebSphere Application Server*), incluyendo además herramientas de desarrollo (*WebSphere Studio*). En el segundo grupo, *Reach and user experience*, se incluyen productos más específicos, pensados para que los usuarios de los mismos puedan llegar a un rango más amplio de clientes. Nos encontramos entre ellos *WebSphere Portal* y *WebSphere Everyplace*. Permiten migrar los negocios a medios de distribución como Internet. Por último, el grupo *Business integration* incluye productos que permiten la interconexión de sistemas. En estos productos se utiliza plenamente la arquitectura basada en mensajes (*WebSphere MQ Integrator Broker* y *WebSphere Business Integrator*).

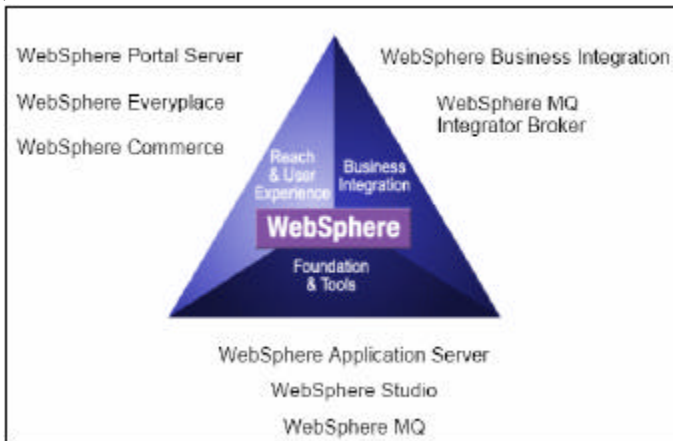


Figura 66. Grupos y productos de WebSphere [30]

Los productos incluidos en el primer grupo (*Foundation and tools*) están directamente relacionados con la tecnología de SW. El *WebSphere Application Server* constituye la base, el *WebSphere Studio* provee un entorno de desarrollo y *WebSphere MQ* aporta el soporte para JMS, además de ser una alternativa para el intercambio de mensajes SOAP. Por otro lado, algunos productos de otros grupos permiten la extensión de los SW a entornos empresariales y usuarios más avanzados [30].

5.4.3. SDK de WebSphere para el desarrollo de Servicios Web

Dado que existen múltiples versiones del *WebSphere Application Server*, todas ellas con distintos paquetes para el desarrollo de SW, nos

centraremos únicamente en la última versión disponible, concretamente la 5.1. Esta versión, incluye dos paquetes con el soporte necesario en el desarrollo de aplicaciones basadas en SW: el *WebSphere SDK for Web Services* (WSDK) y el *Emerging Technologies Toolkit* (ETTK).

El más importante de los paquetes (y en el que nos centraremos) es el WSDK. Este paquete incluye un conjunto de herramientas y un entorno de ejecución para el diseño y ejecución de aplicaciones basadas en SW. Para poder utilizar este *runtime*, los SW desarrollados deben seguir los estándares de los SW además del estándar para la interoperabilidad definido por la *Web Services Interoperability Organization* (WS-I).

Como habíamos dicho, el WSDK incluye un conjunto de aplicaciones y un entorno de ejecución para el diseño, ejecución y prueba de aplicaciones basadas en SW. La funcionalidad del WSDK esta basada en especificaciones abiertas como SOAP WSDL y UDDI. La distribución actual de WSDK esta disponible tanto para Linux como para Windows. El WSDK permite el desarrollo y prueba de SW basados en Java como un *kit* de desarrollo básico.

5.4.3.1. Elementos del WSDK

El WSDK contiene los siguientes elementos:

- Versión embebida de la versión 5.0.2 del *WebSphere Application Server Express* con soporte adicional para ORBs y EJBs.
 - Soporte para SOAP 1.1, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0 (JSR 101), EJB 2.0, *Enterprise Web Services* 1.0 (JSR 109), WSDL4J, UDDI4J y *WS-Security*.
 - Registro privado de UDDI v.2.
 - Base de datos básica con una implementación de JDBC.
 - IBM SDK para Java, versión 1.3.1.
 - Soporte para JMS y los *Message Driven Beans* (MDB).
- *Plug-ins* para la versión 2.1.1. de Eclipse.
- Herramientas para la generación de SW a partir de JavaBeans y *stateless session EJBs*, generación de SW a partir de definiciones WSDL y para la publicación y eliminación de SW del registro UDDI.
- Documentación y ejemplos.

WebSphere Application Server Express

Este servidor incluye un registro UDDI privado para la publicación y descubrimiento de SW mediante los protocolos UDDI. El propio registro se ejecuta como una *enterprise application* (EAR) en el servidor de aplicaciones. Este servidor está basado en la plataforma J2EE, haciendo las veces de servidor Web HTTP, motor de servlets y contenedor EJB.

El servidor incluye también capacidades para los SW, ya que está basado en los estándares definidos en la J2EE, concretamente JAX-RPC (JSR 101) y la integración de los SW con el servidor J2EE (JSR 109).

Herramientas

Se incluyen un conjunto de herramientas que permiten habilitar, utilizar y publicar aplicaciones J2EE. Estas herramientas son:

- *Bean2WebService*: Crea un SW completo a partir de una clase Java y, opcionalmente lo pone en funcionamiento en el servidor de aplicaciones.
- *EJB2WebService*: Crea un SW completo a partir de un *stateless session EJB* contenido en un módulo EJB (contenido en un fichero EAR) y, opcionalmente lo pone en funcionamiento en el servidor de aplicaciones.
- *WSDL2WebService*: Crea un SW completo a partir de uno o más documentos WSDL y, opcionalmente lo pone en funcionamiento en el servidor de aplicaciones.
- *WSDL2Client*: Genera un cliente de SW completo a partir de uno o más documentos WSDL y, opcionalmente lo pone en funcionamiento en el servidor.
- *UDDIPublish*: Publica una entidad o un servicio de negocio en un registro UDDI público o privado.
- *UDDIUnpublish*: Elimina una entidad o un servicio de negocio de un registro UDDI público o privado.

Además el WSDK incluye algunas herramientas de administración como:

- *tcpmon*: Permite observar los mensajes intercambiados entre los clientes de los SW y el servidor de SW.
- *Appserver*: Incluye algunas funciones de administración para el servidor de aplicaciones.

- *setProxy*: Configuración de la información del proxy HTTP para las herramientas del WSDK y el *runtime server*.

Documentación y Ejemplos

La documentación del WSDK se divide en cinco secciones: (1) documentación básica de WSDK y los SW, (2) conceptos específicos de los SW, (3) tareas a realizar para implementar aplicaciones basadas en SW, crear clientes, hacer seguros los SW, etc..., (4) documentación sobre las APIs, guías de problemas, material de referencia para las herramientas del WSDK y (5) ejemplos completos.

5.5. CalculadoraWeb. Implementación de un Servicio Web Sencillo

5.5.1. Descripción genérica del servicio

En esta sección, vamos a poner en práctica los contenidos teóricos estudiados en las secciones anteriores, pero aplicándolos a la plataformas tecnológicas abordadas en la sección 4, JAVA de Sun Microsystems y Microsoft.NET.

La forma de poner en práctica todos los contenidos teóricos acerca de los SW, será el desarrollo de un SW en las dos plataformas estudiadas. Este SW hará las veces de una sencilla y particular calculadora Web. Mediante este ejemplo, pretendemos mostrar como se implementa un SW, como se hacen públicos los métodos que deseamos que los clientes o usuarios utilicen del mismo, y como se solicitan los servicios del mismo mediante una aplicación que lo invoca.

De forma genérica, esta calculadora, ofrecerá dos servicios, uno de ellos que permite realizar operaciones sencillas (tales y como son la suma, resta, división, multiplicación, resto y potencia) y otro, que sin ser excesivamente complejo, resuelve ecuaciones de segundo grado cuyas soluciones sean no imaginarias.

Para complicar un poco el ejemplo, en lugar de utilizar tipos simples para representar los argumentos de los métodos del SW, utilizaremos estructuras que contengan a los argumentos. De esta forma, además de practicar esta faceta no tan trivial como es el envío y la recepción de datos simples (enteros, cadenas,...), también comprobamos como el SW

indica al cliente mediante el documento de especificación del servicio (documento WSDL) como son los tipos que espera recibir y que enviará como respuesta a las peticiones.

De modo que comprendamos la estructura de nuestra calculadora Web, proponemos en la Figura 67, mediante un diagrama de clases, como implementaremos las funcionalidades que nos ofrecerá nuestra calculadora.

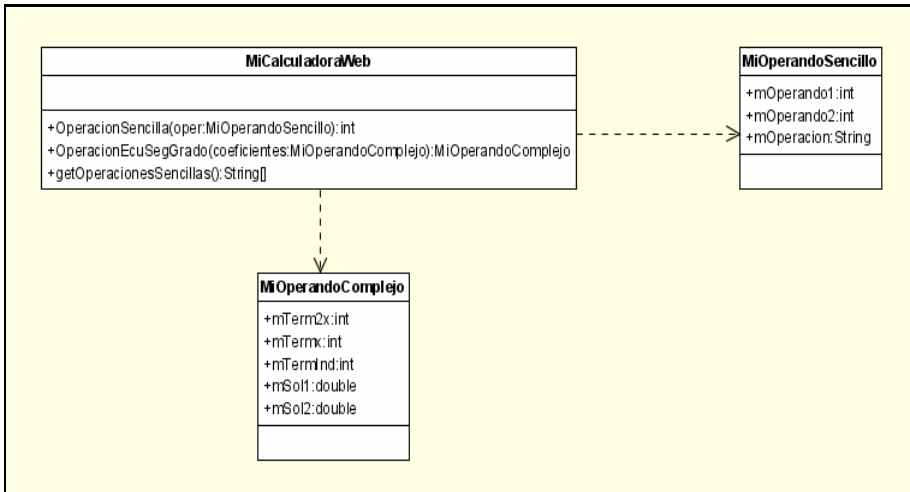


Figura 67. Diagrama de clases de *MiCalculadoraWeb*

La clase principal, que será la que implemente el SW, será *MiCalculadoraWeb* (Figura 68). Ésta tiene tres operaciones públicas que son los servicios que ofrece.

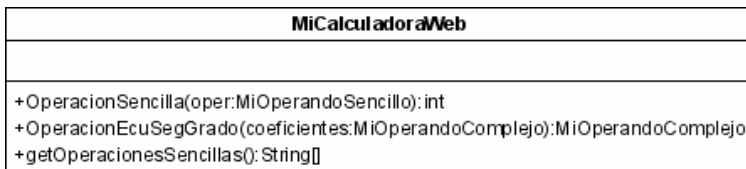


Figura 68. Clase *MiCalculadoraWeb*

La operación *OperacionSencilla*, como su nombre nos hace intuir, realiza operaciones sencillas como la suma, resta, división, producto, resto y potencia, todas ellas de dos operandos. Este método admite sólo un argumento *oper* del tipo *MiOperandoSencillo* (Figura 69), que tiene a su vez tres propiedades, que son la información necesaria para realizar la operación. Estas propiedades son *mOperando1*, que contiene al primer

operando, *mOperando2* que contiene al segundo operando, y *mOperacion* que contiene la operación que deseamos realizar sobre los dos operandos. El método *OperacionSencilla* devuelve un entero como resultado a la operación que solicitamos que realice.

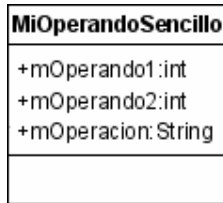


Figura 69. Clase *MiOperandoSencillo*

El método público *getOperacionesSencillas*, simplemente devuelve una matriz con las operaciones que el método *OperacionSencilla* puede realizar. Si al método *OperacionSencilla* le pasáramos una operación que no esté en el conjunto devuelto por el método *getOperacionesSencillas*, lanzará una excepción SOAP, indicando que el fallo es debido a una acción incorrecta del cliente.

El método público *OperacionEcuSegGrado* realiza la resolución de una ecuación de segundo grado del tipo $Ax^2 + Bx + C = 0$, donde la información que nosotros facilitaremos serán los coeficientes A, B y C. Esta operación tiene un argumento *coeficientes* del tipo *MiOperandoComplejo*. En este caso, *MiOperandoComplejo* tiene cinco propiedades; las propiedades, *mTerm2x*, *mTermx*, y *mTermInd*, de tipo entero, corresponden a los coeficientes de la ecuación A, B, y C relativamente. Las propiedades *mSol1* y *mSol2*, de tipo *double*, sirven para que la calculadora Web coloque ahí las dos soluciones que tendrá la ecuación, ya que el argumento *coeficientes* es devuelto al cliente una vez que se ha realizado el cálculo. Esta operación lanza una excepción SOAP si la ecuación tiene al menos una de sus soluciones complejas.

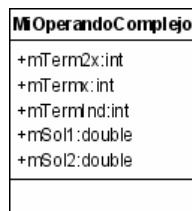


Figura 70. Clase *MiOperandoComplejo*

5.5.2. Implementación en Microsoft.NET

Para ilustrar proceso de desarrollo de un SW en .NET, vamos a utilizar el Visual Studio .NET, que es el entorno integrado de desarrollo que incluye Microsoft en su plataforma .NET. A continuación, vamos a mostrar paso a paso el proceso de desarrollo del un SW sencillo, la *CalculadoraWeb*, que ha sido especificado de manera genérica en el apartado anterior.

El primer paso, es lanzar el Visual Studio .NET, para lo cual, basta con hacer doble clic en el acceso directo (Figura 71).



Figura 71. Lanzar Visual Studio .NET

Una vez que tenemos el entorno abierto, el siguiente paso es seleccionar un lenguaje para desarrollar nuestro proyecto. Como ya sabemos, una de las principales características de la plataforma .NET es la interoperabilidad entre lenguajes, por ello, antes de iniciar cualquier tipo de proyecto, tendremos que seleccionar el lenguaje que deseamos utilizar (Figura 72).

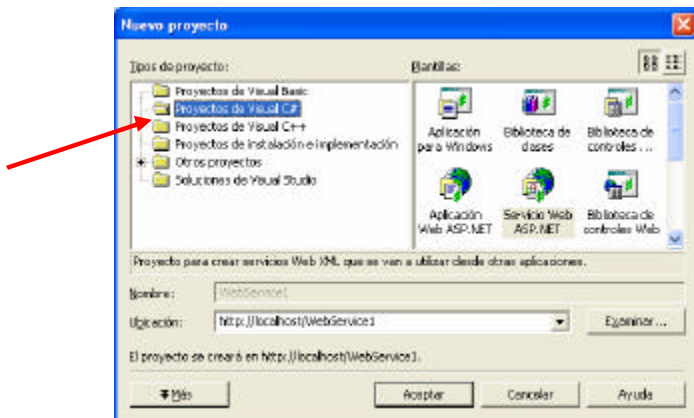


Figura 72. Selección del lenguaje de desarrollo.

Visual Studio .NET nos permite desarrollar, mediante el lenguaje de desarrollo que seleccionemos, diferentes tipos de proyecto. Algunos de estos tipos de proyecto son las aplicaciones para Windows, Bibliotecas de controles, Aplicaciones Web ASP.NET, etc., pero la que realmente nos atañe, son los SW ASP.NET. Así pues, seleccionamos este tipo de proyecto (Figura 73).

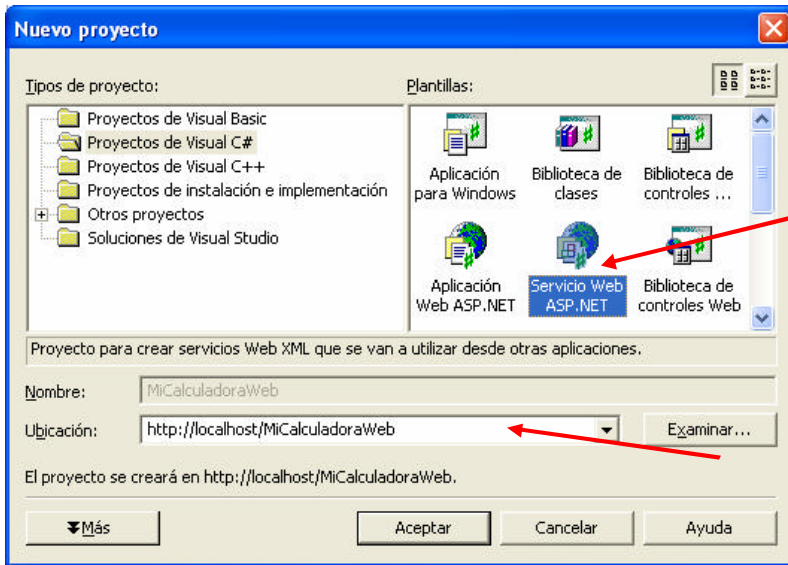


Figura 73. Selección del tipo de proyecto y nombre del mismo.

El siguiente paso, y antes de aceptar en la ventana, tenemos que dar nombre al SW, que de paso será también su URL de acceso (Figura 73). Tras esto, Visual Studio .NET realiza las operaciones necesarias para “dar de alta” al SW en el IIS (*Internet Information Server*), de forma que podamos probarlo, depurarlo, y ofrecerlo como servicio. Si estas operaciones se realizan correctamente, el entorno de desarrollo ya está listo para que comencemos la implementación de nuestro SW (Figura 74). El desarrollo de un SW es bastante similar al desarrollo de una clase en la que vamos a tener métodos públicos que podrán ser invocados.

Para comenzar el desarrollo de nuestra calculadora, podríamos recordar el diagrama de clases que habíamos realizado anteriormente (Figura 67). El primer paso que podemos realizar, es implementar los tipos de datos *MiOperandoSimple* y *MiOperandoComplejo*, que serán necesarios en las operaciones que contendrá la clase *MiCalculadoraWeb*, que por razones de claridad, llamaremos igual que al SW. En la Figura 75, ya hemos

implementado *MiOperandoSencillo*, que es una clase solamente con los atributos contendrán la información para realizar una operación sencilla. Tras esto, el siguiente paso sería implementar *MiOperandoComplejo* (Figura 76), que es igual de sencillo de desarrollar que *MiOperandoSimple*.

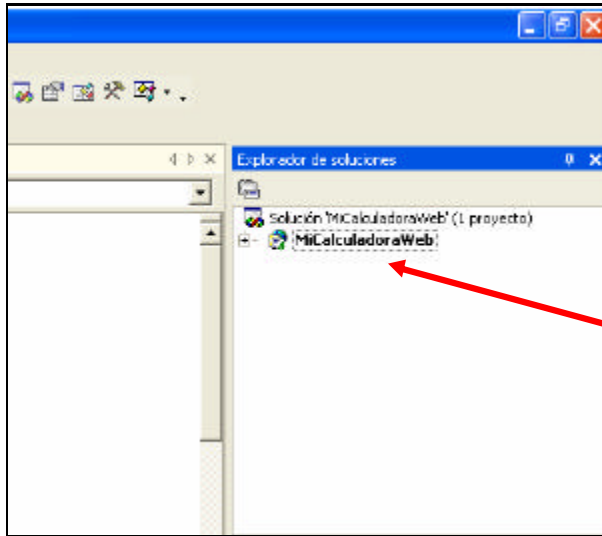


Figura 74. Entorno de desarrollo listo para comenzar.

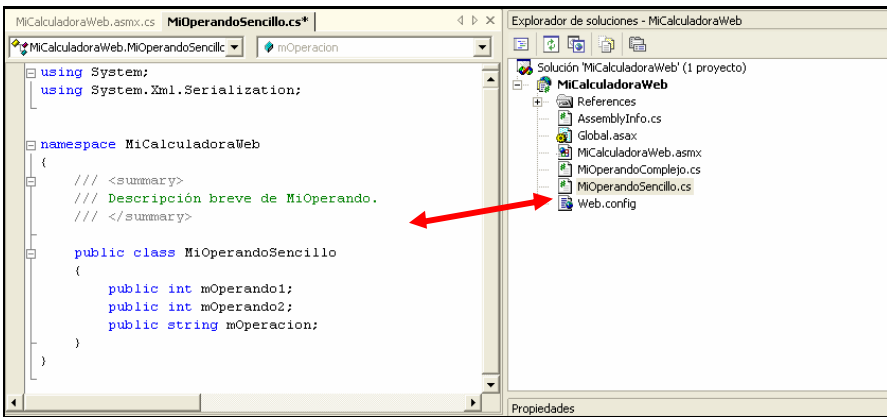


Figura 75. Implementamos *MiOperandoSencillo*.

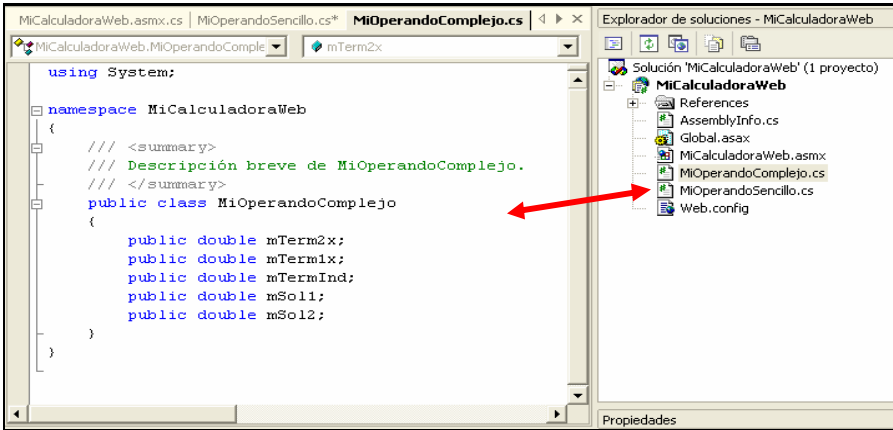


Figura 76. Implementamos *MiOperandoComplejo*

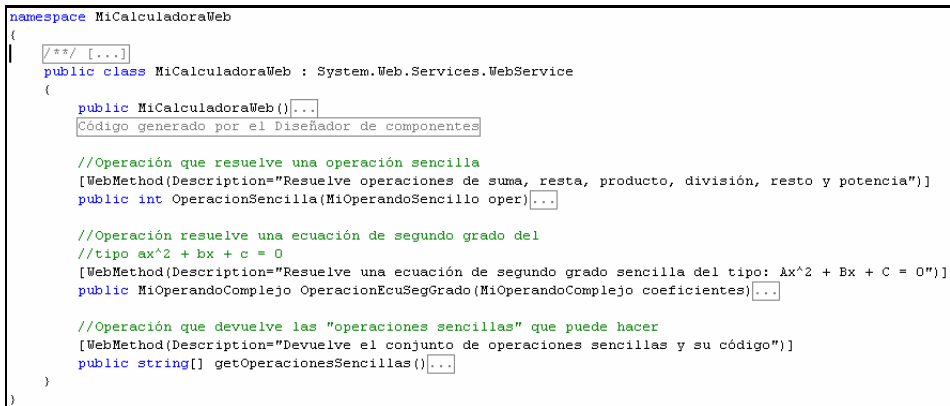


Figura 77. Operaciones que ofrecerá el SW.

El último paso (en lo que a desarrollo se refiere) para terminar nuestro SW, es implementar la clase *MCalculadoraWeb*. En esta clase vamos a situar los métodos que serán los servicios que exponga nuestro SW. En la Figura 77, podemos observar los tres métodos, que son los servicios que ofrece la calculadora. Comenzando por el tercero, *getOperacionesSencillos*, devuelve una matriz de cadenas, que son los nombres con los que se solicita la ejecución de la operación sencilla. Cada una de estas cadenas será el tercer atributo del tipo de dato *MiOperandoSencillo*. El primer método, *OperacionSencilla*, realiza las operaciones de suma, resta, multiplicación, división, resto y potencia con los operandos que le pasemos dentro de *MiOperandoSencillo*. La segunda operación, resuelve una ecuación de segundo grado cuyos coeficientes le pasamos mediante una instancia de la clase *MiOperandoComplejo*. La solución la devolveremos en la misma

instancia de clase que manda el cliente, mediante las propiedades de clase *mSol1* y *mSol2*. No entraremos en detalles de implementación, ya que es lo menos importante.

Una vez que el código está escrito, hay que decir de alguna manera, qué métodos deseamos que sean públicos, es decir, qué operaciones se ofrecerán al cliente. Para ellos, utilizamos unos atributos que Visual Studio .NET tiene *ex profeso*. Estos atributos son *WebMethod* y *WebService*, que ya presentamos en la Sección 4.2.2.4. A parte de personalizar los métodos que deseamos que se oferten como métodos Web, también podemos personalizar la información del servicio, y como hemos visto, esto se hace con el atributo *WebService*.

Llegados a este punto, ya tendríamos nuestro SW *MiCalculadoraWeb* listo, esperando a la consulta de algún cliente. Esto podemos comprobarlo accediendo al mismo mediante su URL a través de un navegador Web (Figura 78).

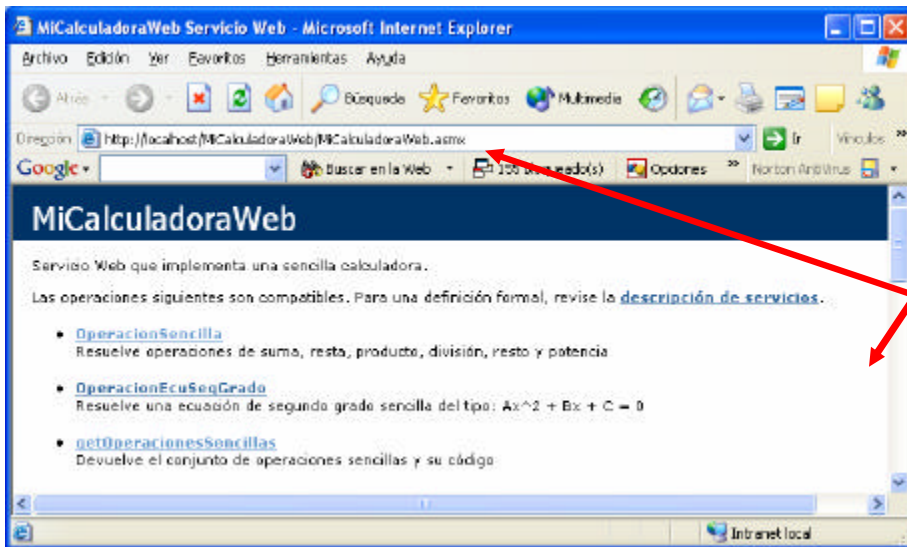


Figura 78. SW accedido a través del navegador.

Si pulsamos sobre el método Web *getOperaciónSencilla*, obtendríamos Figura 79 como respuesta. Tras pulsar en invocar, el resultado de la ejecución de la operación sería Figura 80

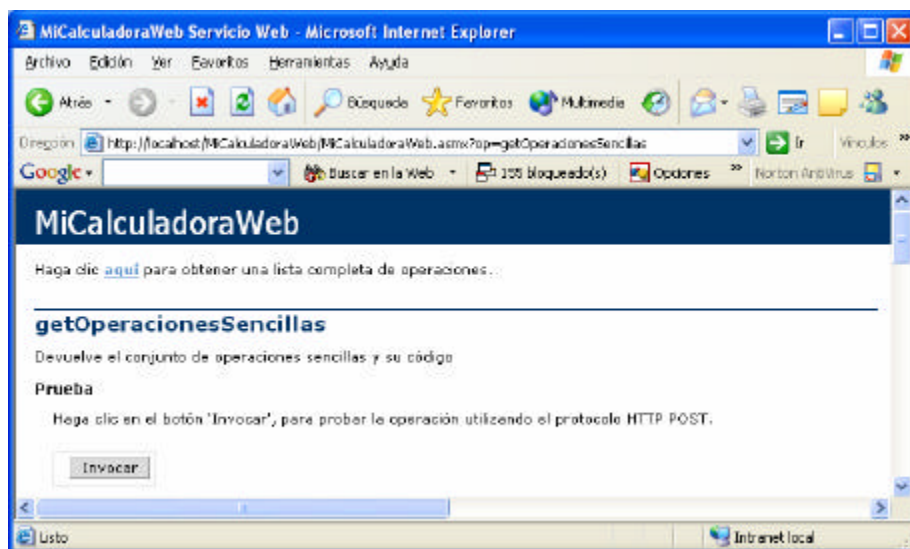


Figura 79. Petición de permiso para invocar la operación *getOperacionesSencillas*

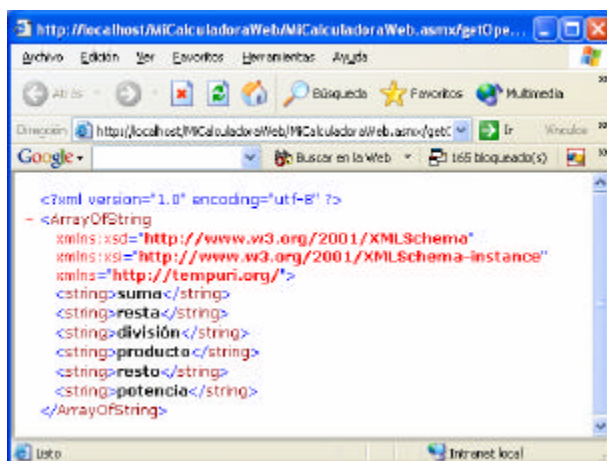


Figura 80. Respuesta con la lista de operaciones.

Para probar el resto de operaciones se ha creado una sencilla aplicación ASP.NET que incluye este servicio y lo utiliza para realizar las operaciones. La interfaz es la siguiente:

Calculadora Web

Operación Sencilla

Operando 1

Operando 2

Operación

Ecuación de Segundo Grado

x^2 + x^2 + = 0

Solución 1

Solución 2

Figura 81. Interfaz del “Invocador de Calculadora Web”

En la lista desplegable de Figura 81, se pueden ver las operaciones de las que dispone el SW, y no es por que se las hayamos puesto nosotros, sino por que al iniciarse la aplicación, se consulta al SW para conseguir la lista de operaciones mediante el método Web *getOperacionesSencillas*.

Para ilustrar el ejemplo vamos a realizar una operación y ver su respuesta:

Operación Sencilla

Operando 1

Operando 2

Operación

Operación Sencilla

Operando 1

Operando 2

Operación

Resultado




Figura 82. Invocación de una suma

Y por último, vamos a ver como resuelve una ecuación de segundo grado:

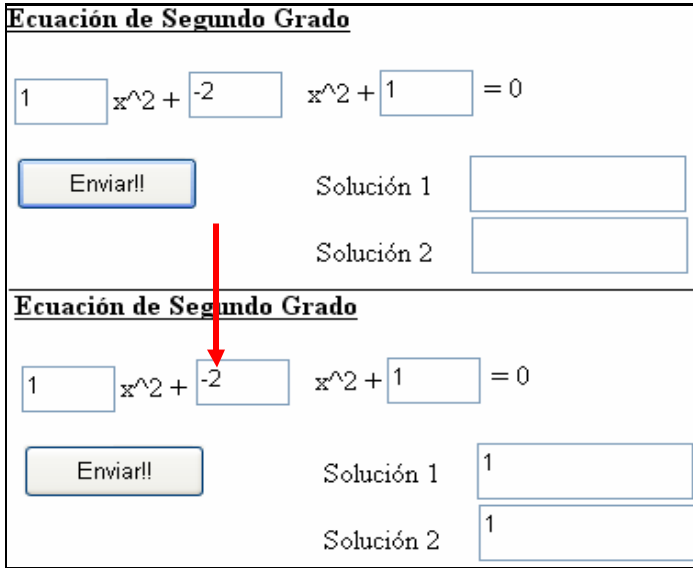


Figura 83. Invocación de la ecuación de segundo grado

5.5.3. Implementación en JAVA

De la misma manera que ilustrábamos la creación de un SW con la plataforma .NET, ahora ilustraremos cómo desarrollar el mismo SW mediante el lenguaje Java. Teniendo en cuenta que podemos utilizar múltiples entornos de desarrollo para implementar nuestros proyectos Java, nos dedicaremos a realizar nuestro SW con el que a día de hoy es uno de los entornos de programación más potentes para el lenguaje de Sun, el JDeveloper de Oracle y, más concretamente, la recientemente lanzada versión 10 del producto.

En primer lugar, crearemos un espacio de trabajo para la aplicación o *Application WorkSpace* donde introduciremos algunos datos como el nombre del espacio y la ruta (Figura 84).

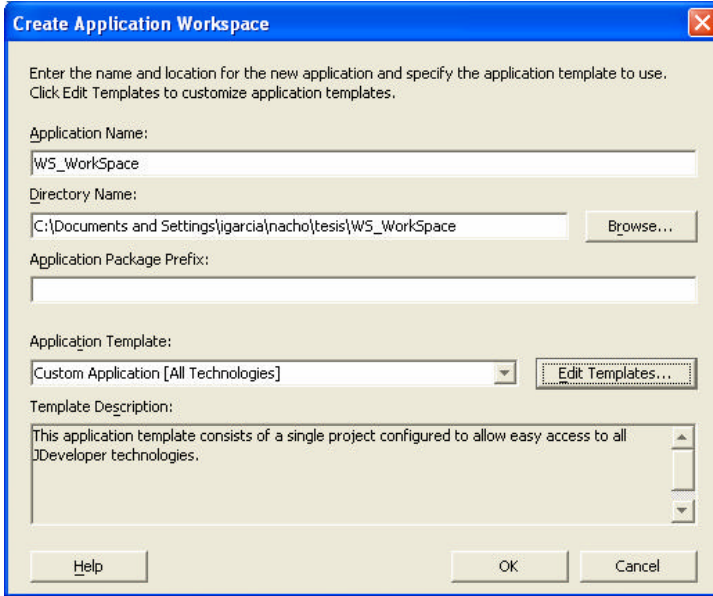


Figura 84. Creación de un espacio de trabajo

Acto seguido, pulsaremos el botón *Edit Templates* de la misma ventana, desplegándose el siguiente cuadro de diálogo:

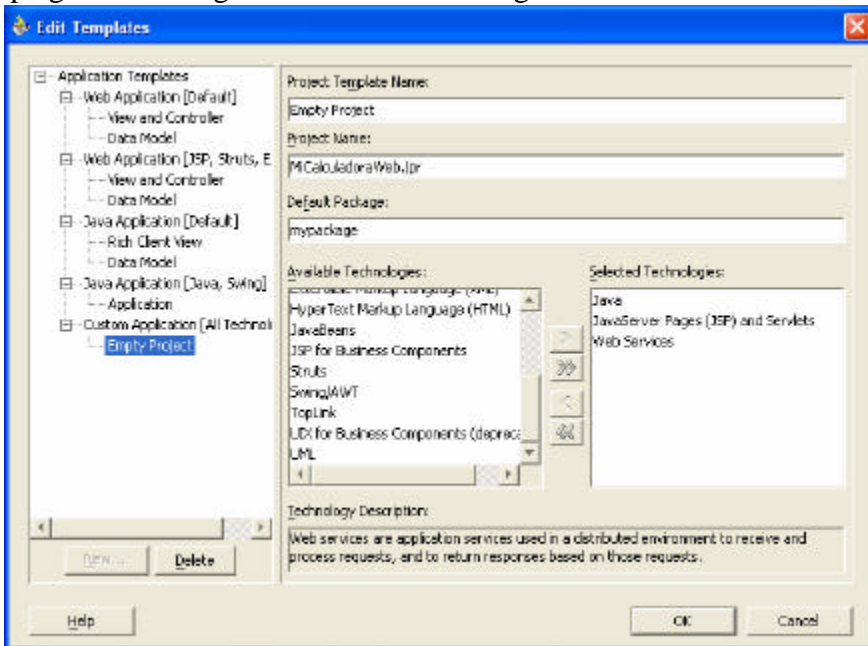


Figura 85. Dialogo de la plantilla del nuevo proyecto

Indicando en el árbol de los tipos de aplicación, que es un *Empty Project*, e indicando luego también que las tecnologías seleccionadas son *JavaServer Pages (JSP) and Servlets* y *Web Services*. Si se desea, se puede seleccionar el nombre del proyecto, aunque la JDeveloper ya pone uno por defecto. Pulsamos ‘OK’ y ya tenemos creado el espacio de trabajo y el proyecto para nuestro SW.

Una vez que tenemos el proyecto creado, el siguiente paso es desarrollar las clases que contendrán el código que implementará las funcionalidades del SW. En este sentido el código no tiene por que ser desarrollador de manera especial, simplemente codificaremos lo que queremos hacer y bastaría (salvo que por las características inherentes al servicio a prestar, tuviéramos que tener en cuenta alguna particularidad especial). Así pues, las clases de nuestro SW tendrían el siguiente aspecto:

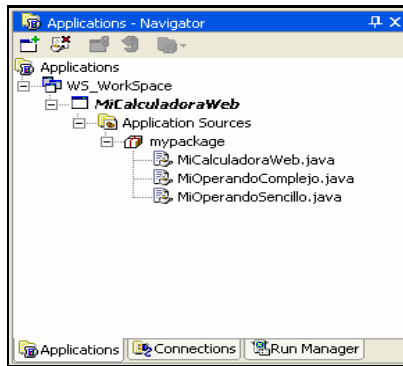


Figura 86. Clases del SW “MiCalculadoraWeb”

El siguiente paso es generar el SW, y para ello tomaremos como punto de inicio la clase que implementa las operaciones que serán ofertadas. Sobre esto y previo a la generación del SW, debemos asegurarnos que todos los tipos de datos utilizados en las operaciones ofertadas están bien especificados. Por ejemplo, en nuestro tipo de dato *MiOperandoSencillo*, tenemos que implementar métodos de acceso a las propiedades de la clase (Figura 87) (esto es, métodos “*get*” y “*set*”), ya que de lo contrario las operaciones que incluyan estos operando no serán incluidas como parte de la interfaz del SW, quedando ocultas para los clientes.

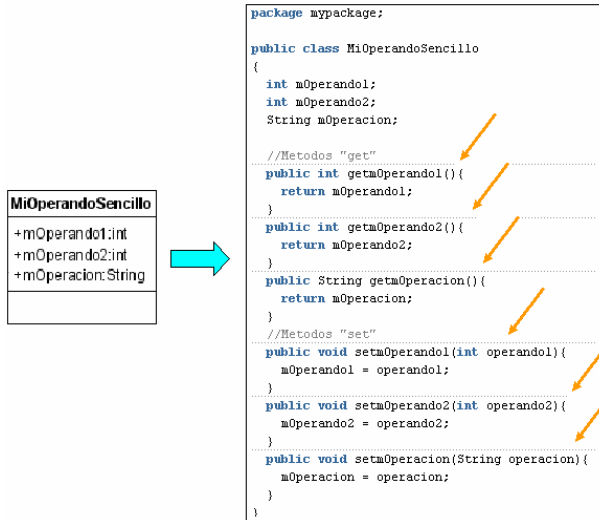


Figura 87. MiOperandoSencillo con sus métodos de acceso.

Una vez realizada esta tarea ya podemos comenzar el proceso automático de generación del SW. Para ello, seleccionamos la clase que contiene las operaciones que serán públicas en el SW, pulsamos el botón derecho sobre la misma, y seleccionamos la opción “*Generate Web Service From Class*”. En el mismo proyecto se genera un fichero llamado *MyWebServiceX*, siendo X un número asignado por el entorno. Lo siguiente es personalizar el SW con su ventana de configuración (pulsando el botón derecho sobre el fichero generado y seleccionando la opción *Edit*). En la pestaña *Class* de la ventana de dialogo configuramos el nombre (Figura 88). Tras esto, seleccionamos de los métodos disponibles en la clase a partir de la cual generamos el SW, cuales queremos que estén disponibles a los clientes (Figura 89) y el SW quedaría configurado. JDeveloper ofrece la posibilidad de configurar el SW con un nivel de granularidad más fino, no obstante la exploración de estas opciones adicionales escapa al alcance de esta sección.

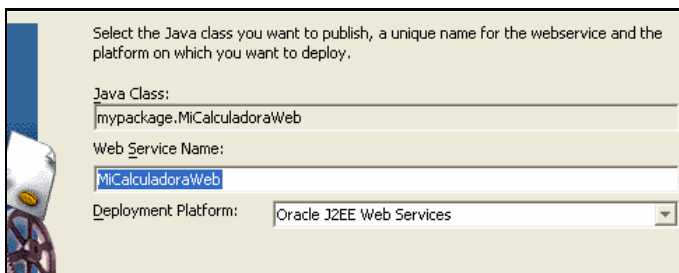


Figura 88. Primer paso en la configuración del SW.

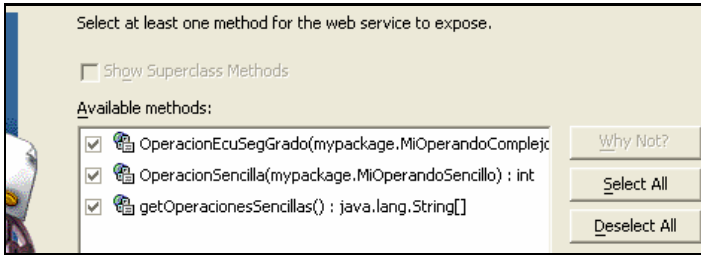


Figura 89. Segundo paso en la configuración del SW.

Por último, para probar el SW no hay más que lanzarlo desde el JDeveloper y acceder al mismo mediante cualquier navegador Web (Figura 90).



Figura 90. SW MiCalculadoraWeb

DICCIONARIO

DICCIONARIO

<u>INGLÉS</u>	<u>ESPAÑOL</u>
Atomic transaction	Transacción atómica
Binding	Vinculación
Business application	Aplicación de negocio
Business environment	Entornos de negocio
Business process management language	Lenguaje de ejecución de procesos de negocio
Business relationship	Relación de negocio
Business transaction	Transacción de negocio
Business transaction protocol	Protocolo de transacción de negocios
Compensation activities	Actividades de compensación
Endpoint	Localizador de red
Entity	Entidad
Legacy system	Sistema heredado
Long-running transaction	Transacción de larga duración
Message	Mensaje
Module	Módulo

Notification message	Mensaje de notificación
One-way message	Mensaje unidireccional
Party ¹	Parte
Port	Puerto
portType	Tipo de puerto
Provider	Proveedor
Request-response message	Mensaje de petición-respuesta
Requester	Solicitante
Resource	Recurso
REST-compliant	Conforme a REST
Transaction	Transacción
Transaction models	Modelos de transacción
Two-phase completion protocol	Protocolo de finalización de dos fases
Two-phase protocol	Protocolo de dos fases
Types	Tipos
Service aggregator	Agregador de servicio
Service composition	Composición de servicios
Short-duration transactions	Transacciones de corta duración
SOAP Envelope	Sobre SOAP

¹ En el contexto de las transacciones y la coordinación (ver apartado 4.2)

SOAP features	Características SOAP
SOAP Header	Cabecera SOAP
SOAP Node	Nodo SOAP
Solicit-response message	Mensaje de solicitud-respuesta
Web service choreography interface	Interfaz de coreografía de servicios web

REFERENCIAS

REFERENCIAS

1. Alonso, G., Casati, F., Kuno, H. y Machiraju, V., *Web Services. Concepts, Architectures and Applications*, Carey, M.J. y Ceri, S. (eds.). 2004, Berlin: Springer. Num. páginas: 354
2. Brickley, D., Froumentin, M., *SWAD-Europe Deliverable 4.1: Semantic Web and Web Services: RDF/XML and SOAP for Web Data Encoding*, Project Number: IST-2001-34732, [://www.w3.org/2001/sw/Europe/reports/xml_graph_serialization_report/](http://www.w3.org/2001/sw/Europe/reports/xml_graph_serialization_report/). Disponible el 01-06-2004.
3. Chung, J., Lin, K., Mathieu, R.G., *Web Services Computing: Advancing Software Interoperability*, IEEE Computer Society, October (2003), pp. 35-62.
4. *CORBA IIOP Specification*. http://www.omg.org/technology/documents/formal/corba_iiop.htm. Disponible el 20-10-2004.
5. Curbera, F., Khalaf, R., Mukhi, N., Tai, S. y Weerawarana, S., The Next Step in Web Services, in *Communications of the ACM*. 2003. pp. 29-34.
6. *Extensible Markup Language (XML) 1.0 (Second Edition)*. <http://www.w3.org/TR/2000/REC-xml-20001006>. Disponible el 10-02-2004.
7. García-Molina, H. y Salem, K., *Sagas*, ACM SIGMOD International Conference on the Management of Data. 1987: ACM.
8. Gray, J.N., The transaction concept: virtues and limitations. *Proceedings of the 7th VLDB Conference*, 1981: pp. 144-154.
9. <http://java.sun.com>, Disponible en 15-12-2003.
10. <http://www.W3.org/TR/2000/REC-xml-20001006>, Disponible el 15-12-2003.

11. Jacobs, I., *Architecture of the World Wide Web*, W3C Working Draft. <http://www.w3.org/TR/2003/wd-webarch-20030627/>. Disponible el 27-6-2003.
12. *Java Message Service (JMS)*. <http://java.sun.com/products/jms/>. Disponible el 10-02-2004.
13. Jewell, T., Chappell, D. *Java Web Services*, Marzo 2002. <http://www.oreilly.com/catalog/javawebse/vchapters/ch06.html>. Disponible el 10-2-2004.
14. Kendall, S.C., Waldo, J., Wollrath, A., Wyant, G., *A Note On Distributed Computing*, Sun Microsystems Laboratories, Inc., <http://research.sun.com/techrep/1994/smlr-tr-94-29.pdf>. Disponible el 10-02-2004
15. Little, M., Transactions and Web Services. Communications of the ACM, 2003. 46(10): pp. 49-54.
16. Microsoft Corporation. <http://www.Microsoft.com>. Disponible el 10-02-2004.
17. Nicoloudis, N., Mingins, C., *XML Web Services Automation: A Software Engineering Approach*, Proceedings of the 9th Asia-Pacific Software Engineering Conference (APSEC'02), IEEE Computer Society.
18. Papazoglou, M.P. y Georgakopoulos, D., Service - Oriented Computing, in Communications of the ACM. 2003. p. 25-28
19. Paul's REST Resources. <http://www.prescod.net/rest/>. Disponible el 10-02-2004
20. Peltz, C., *Web Services Orchestration and Choreography*, Computer, October 2003, Published by the IEEE Computer Society, pp.46-52
21. Short, S. *Creación de Servicios Web XML para la plataforma Microsoft® .NET*, Editorial Mc Graw Hill, Madrid, 2002.

22. Sun Microsystems. <http://www.sun.com>. Disponible el 10-02-2004.
23. *The Open System Interconnection (OSI) Architecture*, <http://cne.gmu.edu/modules/network/osi.html>. Disponible el 10-02-2004.
24. Turner, M., Budgen, D. y Brereton, P., Turning Software into a Service, in IEEE Computer Society. 2003. pp. 38-44
25. UDDI Technical White Paper. http://www.uddi.org/pubs/lru_uddi_technical_white_paper.pdf. Disponible el 10-2-2004.
26. W3C, SOAP version 1.2 Part 1. <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>. Disponible el 20-10-2004.
27. W3C, SOAP version 1.2. Part 2. <http://www.w3.org/TR/2003/REC-soap12-part2-20030624/>. Disponible el 20-10-2004
28. W3C, Web Services Architecture. <http://www.w3.org/TR/2003/WD-ws-arch-20030808/>. Disponible el 20-10-2004.
29. W3C ,Web Service Description Language. <http://www.w3.org/TR/2001/NOTE-wsd1-20010315>. Disponible el 20-10-2004.
30. Wahli, U., Garcia Ochoa, G., Cocasse, S. y Muetschard, M., *WebSphere Version 5.1. Application Developer 5.1.1. Web Services Handbook*. Segunda Edición. 2004: IBM.
31. World Wide Web Consortium. <http://www.w3c.org> Disponible el 10-2-2004.
32. Would, D., *Accessing CICS Transactions from EJBs*, CICS System Test, IBM, 24-6-2004. <http://www->

16.ibm.com/developerworks/eserver/articles/would_cics.html.
Disponible el 10-2-2004.

33. *XML Path Language (XPath)*. <http://www.w3.org/TR/xpath>.
Disponible el 10-2-2004.

34. *XML Schema*. <http://www.w3.org/XML/Schema>. Disponible el
10-02-2004.

ACRÓNIMOS

ACRÓNIMOS

API	<i>Application Program Interface</i>
ASP	<i>Active Service Pages</i>
BPEL	<i>Abreviatura de BPEL4WS</i>
BPEL4WS	<i>Business Process Execution Language for Web Services</i>
BPML	<i>Business Process Management Language</i>
CLR	<i>Common Language Runtime</i>
CORBA	<i>Common Object Request Broker Architecture</i>
CSS	<i>Cascade Style Sheet</i>
DAACL	<i>Discretionary Access Control List</i>
DOM	<i>Document Object Model</i>
DTD	<i>Document Type Definition</i>
EJB	<i>Enterprise JavaBean</i>
ETTK	<i>Emerging Technologies Toolkit</i>
FTP	<i>File Transfer Protocol</i>
HTTP	<i>Hyper Text Transport Protocol</i>
IIOB	<i>Internet Inter-ORB Protocol</i>
IIS	<i>Internet Information Server</i>
IP	<i>Internet Protocol</i>
ISAPI	<i>Internet Server Application Programming Interface</i>
J2EE	<i>Java 2 Platform, Enterprise Edition</i>
JAX-RPC	<i>Java API for XML-based Remote Procedure Call</i>
JAXB	<i>Java Architecture for Java Bindings</i>
JAXM	<i>Java API for XML Messaging</i>
JAXR	<i>Java API for XML Registries</i>
JMS	<i>Java Messaging Service</i>
JSR	<i>Java Specification Request</i>
LAN	<i>Local Area Network</i>
MDB	<i>Message Driven Bean</i>
MEP	<i>Message Exchange Patterns</i>
MIME	<i>Multipurpose Internet Mail Extensions</i>
MOM	<i>Message Object Model</i>
OMG	<i>Object Management Group</i>
ORB	<i>Object Request Broker</i>
OSI	<i>Open System Interconnection</i>
PC	<i>Personal Computer</i>
QName	<i>Qualified Name</i>

RMI	<i>Remote Method Invocation</i>
REST	<i>Representation State Transfer</i>
RPC	<i>Remote Procedure Call</i>
SAX	<i>Simple API for XML Parsing</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SNA	<i>Systems Network Architecture</i>
SOA	<i>Service Oriented Architecture</i>
SOAP	<i>Simple Object Access Protocol</i>
SOC	<i>Service Oriented Computing</i>
SSL	<i>Secure Socket Layer</i>
SW	<i>Servicio Web</i>
TCP	<i>Transport Control Protocol</i>
UBR	<i>UDDI Business Registry</i>
UDDI	<i>Universal, Description, Discovery and Integration</i>
URI	<i>Uniform Resource Identifier</i>
URL	<i>Uniform Resource Locator</i>
UUID	<i>Universally Unique Identifier</i>
WAN	<i>Wide Area Network</i>
WS-I	<i>Web Service Interoperability Organization</i>
WSA	<i>Web Service Architecture</i>
WSCI	<i>Web Service Choreography Interface</i>
WSD	<i>Web Service Definition</i>
WSDK	<i>Web Service Development Kit</i>
WSDL	<i>Web Service Description Language</i>
WSDL4J	<i>Web Service Description Language for Java Toolkit</i>
WSFL	<i>Web Service Flow Language</i>
WWW	<i>World Wide Web</i>
W3C	<i>World Wide Web Consortium</i>
XML	<i>Extender Mark up Language</i>
XPath	<i>XML Path Language</i>
XSL	<i>eXtensible Stylesheet Language</i>
XSLT	<i>XSL Transformation</i>

