Componentes web: La necesidad de tratar los servicios web como componentes software

Luis Iribarne*

Resumen

La disciplina de la ingeniería del software está experimentando una rápida consolidación en la aplicación de actividades de desarrollo y utilización de tecnologías y metodologías de programación basada en web. Este es el caso del campo de los servicios web, donde están empezando a aparecer prácticas orientadas y basadas en servicios web para el desarrollo de aplicaciones de software a gran escala (como los sistemas de información distribuidos). La consideración de prácticas de desarrollo de software basado en el ensamblaje de servicios web (muchos de ellos elaborados por terceras partes) conlleva la necesidad de plantearse problemas similares a los que aparecen en el desarrollo de software basado en componentes (más extendido en el campo de los componentes COTS), como la compatibilidad o la interoperabilidad. En este trabajo se realiza un estudio comparativo entre la descripción tradicional de los componentes software y el lenguaje de descripción de servicios web (WSDL), y se pone de manifiesto las limitaciones que presenta la actual especificación del servicio de directorios (UDDI) para desarrollar aplicaciones de software distribuidas mediante el ensamblaje de servicios web.

Palabras Claves: *servicios web, componentes software, WSDL, UDDI, especificación semántica, especificación de protocolos.*

1. Introducción

Los grandes sistemas de información y aplicaciones de software de hoy día están basados en modelos cliente/servidor con arquitecturas multicapa y que hacen uso de una gran variedad de tecnologías. La tendencia es que estos sistemas estén distribuidos y localizados en distintos lugares geográficos, comunicándose con modelos distribuidos como CORBA, EJB y/o DCOM, haciendo uso de normas y técnicas de seguridad importantes, y utilizando técnicas como XML/XMI para la representación intermedia de la información entre componentes software. Para la construcción de estos sistemas, los ingenieros suelen requerir la utilización de prácticas y procesos tradicionales del desarrollo de software basado en componentes (DSBC), que permiten, en muchos casos, construir sus sistemas a partir de componentes desarrollados por terceras partes, conocidos como componentes comerciales o componentes COTS. Estas prácticas, empleadas tanto para la construcción de sistemas "orientados" en componentes comerciales como para los "basados" en componentes comerciales, suelen estar identificadas por actividades o tareas importantes en DSBC, entre otras como la descripción de los componentes, o la publicación y localización de componentes, normalmente llevadas a cabo por servicios de mediación [20].

Por otro lado, en estos últimos tiempos se ha podido comprobar una paulatina consolidación de las técnicas empleadas en la programación web, como WSDL, SOAP y UDDI, con el aumento en el uso de los servicios web para construir sistemas de información basados —parcial o totalmente— en web. Aunque estas técnicas basadas en servicios web también

*Departamento de Lenguajes y Computación, Universidad de Almería, luis.iribarne@ual.es

permiten la descripción (WSDL), y la publicación y localización (UDDI) de servicios específicos que funcionan bajo recursos Internet, sin embargo no permiten la programación de complejos sistemas de información distribuidos basados en servicios web donde uno o más servicios puedan requerir la presencia de otros para funcionar, tal y como ocurre en DSBC con el ensamblaje de componentes software.

En el presente trabajo se analizan y comparan algunas de las actividades tradicionales del desarrollo de software basado en componentes —como la actividad de descripción o especificación y la actividad de publicación y localización— con aquellas equivalentes empleadas para el caso del desarrollo de software basado en servicios web, y se pone de manifiesto las limitaciones que estas últimas presentan para la construcción de sistemas siguiendo las prácticas tradicionales de DSBC.

En lo que sigue, el trabajo queda estructurado de la siguiente forma. En las secciones 2 y 3 se analizan la maneras tradicionales de describir los componentes software y los servicios web. En la sección 4 se realiza un estudio de la actual especificación del servicio de directorio UDDI para la publicación y localización de servicios web. En la sección 5 se analizan las limitaciones que presentan WSDL y UDDI para el desarrollo de aplicaciones basadas en servicios web. Se finaliza con unas conclusiones y trabajos futuros.

2. Descripción de componentes software

El desarrollo de software basado en componentes es un paradigma para desarrollar software, donde todos los artefactos, desde el código ejecutable hasta los modelos de especificación de interfaces, arquitecturas y modelos de negocio, pueden ser construidos ensamblando, adaptando e instalando todos ellos juntos y en una variedad de configuraciones. Sin embargo, esto no sería posible sin una descripción clara de los componentes.

Un componente software requiere de información de especificación para los usuarios y los implementadores del módulo. En el contexto de reutilización del software, la especificación ayuda a determinar si un módulo puede satisfacer las necesidades de un nuevo sistema. En el contexto de interoperación del módulo, la especificación se utiliza para determinar si dos módulos pueden interoperar.

Un componente software C puede quedar caracterizado como [18]: "C = (Atr + Oper + Even) + Comp + (Prot * Esc) + Prop". Los atributos (Atr), las operaciones o mtodos ($Oper$) y los eventos ($Even$) son parte de la interfaz de un componente, y representa su nivel sintáctico. El comportamiento ($Comp$) de estos operadores representa la parte semántica del componente. Los protocolos ($Prot$) determinan la interoperabilidad del componente con otros, es decir, la compatibilidad de las secuencias de los mensajes con otros componentes y el tipo de comportamiento que va a tener el componente en distintos "escenarios" (Esc) donde puede ejecutarse. Finalmente, el trmino $Prop$ se refiere a las "propiedades" extrafuncionales que puede tener el componente, como propiedades de seguridad, fiabilidad o rendimiento, entre otras. La descripción de un componente se resumen pues como:

a) **Descrición sintáctica.** Un componente software puede quedar identificado por una o más interfaces. Una interfaz contiene sólo información sintáctica de las signaturas de las "operaciones" entrantes y salientes de un componente con las que interactúa con otros. A este tipo de interacción se le conoce con el nombre de "control proactivo", esto es, las operaciones de un componente son activadas mediante las llamadas de otro. Además del control proactivo (la forma usual de llamar a una operación) está el "control reactivo" que se refiere a los "eventos" de un componente, como el que permite por ejemplo el modelo de componentes EJB (*Enterprise JavaBeans*). En el control reactivo, un componente puede generar eventos que se corresponden con una petición de llamada a operaciones; más tarde otros componentes del sistema recogen estas peticiones y se activa la llamada a la operación.

b) **Descripción semántica.** Sin embargo, no todas las secuencias de invocación de operaciones están permitidas. Existen restricciones operacionales que especifican los patrones de operación permisibles. A la hora de construir aplicaciones no basta con tener especificaciones que contengan sólo el nombre las signaturas y de los atributos del componente [36] [37]; es necesario incluir especificación semántica de las interfaces, para el significado de las operaciones y la especificación de su comportamiento [33]. La información a nivel "semántico" se puede describir con formalismos como las pre/post condiciones, como Larch [14], JML (*Java Modeling Language* o JavaLarch, `ftp://ftp.cs.iastate.edu/pub/leavens/JML`) [27] y OCL (*Object Constraints Language*) [35]. Lenguajes de programación como Eiffel [29] y SPARK [4] permiten escribir especificaciones de comportamiento utilizando pre/post condiciones dentro del código. Otros formalismos para la especificación semántica son las ecuaciones algebraicas [17], el cálculo de refinamiento [30], y otras extensiones de métodos formales orientados a objetos tradicionales que están siendo usados para la especificación de componentes software, como OOZE [2], VDM++ [15] y Object-Z [16].

c) **Descripción de protocolos (coreografía).** Además de la información anterior, también es necesario otro tipo de información semántica que concierne al orden en el que deben ser llamadas las operaciones de las interfaces de un componente. A esta información semántica se la conoce con el nombre de "protocolos" de interacción (también llamada "coreografía"). Los protocolos de interacción pueden variar dependiendo del tipo de componente con el que interacciona y del "escenario" donde ésta se lleva a cabo. Para especificar información de protocolos existen diferentes propuestas —dependiendo del formalismo— como redes de Petri [6], máquinas de estados finitas [36], lógica temporal [19] [26] o el π-cálculo [11] [28]. Existen otros lenguajes para la sincronización de componentes (otra forma de referirse a los protocolos), como Object Calculus [25], Piccola [1] o ASDL (*Architectural Style Description Language*) [31].

d) **Descripción de la información de calidad y extra-funcional (NFR).** Otro aspecto importante es la información de calidad de un componente, recogido en la literatura como atributos de calidad. Un atributo de calidad se refiere tanto a información de calidad de servicio —p.e., el tiempo máximo de respuesta, la respuesta media y la precisión— como a atributos relacionados con la funcionalidad y la extra-funcionalidad de un componente, como por ejemplo la interoperabilidad y la seguridad para el caso de los atributos funcionales, o la portabilidad y la eficiencia para el caso de los atributos extra-funcionales, entre otros muchos [21]; aunque la mayor parte de los atributos de calidad están relacionados con la información extra-funcional. La información extra-funcional también viene recogida con otros términos en la literatura tradicional, como información extra-funcional, restricciones extra-funcionales NFR (*Non-Functional Restrictions*), o *ilities* [5, 32] (para referirse a términos como *reliability*", "*portability*", "*usability*", o "*predictability*"). En la literatura existen dos referencias clásicas en el concepto de "*ilities*": [3] y [13]. El primero es un estándar ISO 9126, y el segundo recoge y clasifica más de 100 *ilities*.

3. Descripción de servicios web

Los servicios web son pequeñas aplicaciones que pueden ser publicadas, localizadas e invocadas desde cualquier sitio web o red local basados en estándares abiertos de Internet. Combinan los mejores aspectos de la programación basada en componentes y la programación web, y son independientes del lenguaje en el que fueron implementados, del sistema operativo donde funcionan, y del modelo de componentes utilizado en su creación.

Una definición algo más formal y extendida es la que ofrece el grupo de trabajo de Servicios Web del W3C [34] que dice lo siguiente: "Un servicio web en un sistema software identificado por un URI (un identificador de recursos uniforme) [8], cuyas interfaces públicas y enlaces están definidos y descritos en XML, y su definición puede ser localizada por otros sistemas de software que pueden luego interactuar con el servicio web en la manera preestablecida por su definición, utilizando mensajes basados en XML transmitidos por protocolos de Internet".

Desde su aparición, los estándares XML y SOAP empezaron a ser utilizados en la programación de aplicaciones web. Sin embargo, las dos compañías pioneras en la creación y soporte del estándar SOAP (IBM y Microsoft), también vieron la necesidad de definir los servicios web en notación XML. En los primeros meses del año 2000, IBM desarrolló NASSL (*Network Accessibility Service Specification Language*), un lenguaje basado en XML para la definición de interfaces de servicios web, y que utilizaba el XML-Schemas del W3C como lenguaje para definir los tipos de datos de los mensajes SOAP. Por otro lado, en el mes de abril de ese mismo año, Microsoft lanzó al mercado SCL (*Service Contract Language*), y que también utilizaba XML para definir interfaces de servicios web, y el XDR (*XML Data-Reduced*) como lenguaje para la definición de tipos de datos. Pocos meses después, Microsoft ofrece SDL (*Services Description Language*) incorporado a Visual Studio .NET.

Afortunadamente, y como objetivo común a ambas compañías, IBM y Microsoft —junto con Ariba— aunaron sus esfuerzos y crearon un grupo de trabajo para la elaboración de un lenguaje estandarizado. Como resultado se genera WSDL (*Web Services Definition Language*), el primer lenguaje para la definición de servicios web. WSDL fue aceptado como recomendación por el W3C en marzo de 2001, y su especificación se puede encontrar en `http://www.w3.org/TR/wsdl`. Finalmente, este lenguaje utiliza la definición de tipos de datos base del W3C, pudiendo ser extendidos. IBM ofrece soporte WSDL en `http://alphaworks.ibm.com/tech/webservicestoolkit`. Microsoft ofrece soporte WSDL en `http://msdn.microsoft.com/xml` y en `http://msdn.microsoft.com/net`.

Por tanto, un servicio web definido en WSDL permite que cualquier cliente web pueda llamarlo mediante programación sin tener que conocer nada acerca de los detalles de implementación del servicio web o sobre qué plataforma o sistema operativo está funcionando.

Un documento WSDL está compuesto por siete elementos XML, aunque no necesariamente todos ellos son obligatorios para definir un servicio web. Estos elementos son: (1) los tipos (`<types>`), que se definen utilizando un esquema del W3C (aunque también se pueden utilizar otros tipos particulares accesibles desde un espacio de nombres); (2) los mensajes de entrada y salida (`<message>`); (3) los tipos de puerto (`<portType>`), donde se definen las interfaces del servicio; (4) las operaciones de una interfaz (`<operation>`); (5) los enlaces (`<binding>`); (6) los puertos (`<port>`); y el (7) servicio (`<service>`). Todos estos elementos (y algunos más) están definidos dentro del esquema WSDL `http://schemas.xmlsoap.org/wsdl/`.

En la figura 1 mostramos una definición de un servicio web definido utilizando el lenguaje WSDL. El servicio acepta como dato de entrada un valor de DNI que lo utiliza para buscar la información de usuario correspondiente en su base de datos asociada. Como respuesta, el servicio web devuelve el nombre y el primer apellido del usuario identificado.

El contenido de un documento WSDL se "escribe" de forma modular, definiendo primero cada una de sus partes en las primeras secciones del documento, y luego componiéndolas en el resto del documento. En primer lugar se definen los tipos de datos utilizados por el servicio web (tipos de entrada y de salida). Luego se define la forma de los mensajes y los tipos de puertos que acepta el servicio (las interfaces). Seguidamente se establece el punto de conexión para el servicio web. Y finalmente se establece el servicio web, con sus puertos y la forma de acceso por Internet.

Tal y como podemos apreciar en el documento WSDL del servicio de identificación mostrado en la figura 1, un servicio web queda delimitado por el elemento `definitions`,

```
1:  <?xml version="1.0"?>
2:  <wsdl:definitions name="servicioIdentificacion"
3:       targetNamespace="http://sitiofictitio.es/servicioIdentificacion.wsdl"
4:       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
5:       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
6:     <wsdl:types>
7:       <xsd:schema targetNamespace="http://sitiofictitio.es/identificacion.xsd"
8:                xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema">
9:         <xsd:element name="peticionIdentificacion">
10:          <xsd:complexType>
11:            <xsd:element name="DNI" type="xsd:float"/>
12:          </xsd:complexType>
13:        </xsd:element>
14:        <xsd:element name="devolucionIdentificacion">
15:          <xsd:complexType>
16:            <xsd:element name="Nombre" type="xsd:string"/>
17:            <xsd:element name="ApellidoA" type="xsd:string"/>
18:          </xsd:complexType>
19:        </xsd:element>
20:      </xsd:schema>
21:    </wsdl:types>
22:    <wsdl:message name="datosEntrada">
23:      <wsdl:part name="body" element="peticionIdentificacion"/>
24:    </wsdl:message>
25:    <wsdl:message name="datosSalida">
26:      <part name="body" element="devolucionIdentificacion"/>
27:    </wsdl:message>
28:    <wsdl:portType name="identificacionTipoPuerto">
29:      <wsdl:operation name="identificar">
30:        <wsdl:input message="datosEntrada"/>
31:        <wsdl:output message="datosSalida"/>
32:      </wsdl:operation>
33:    </wsdl:portType>
34:    <wsdl:binding name="identificacionEnlace" type="identificacionTipoPuerto">
35:      <wsdl:operation name="Identificar">
36:        <soap:operation soapAction="http://sitiofictitio.es/identificar"/>
37:        <wsdl:input> <soap:Body use="literal"/> </wsdl:input>
38:        <wsdl:output> <soap:Body use="literal"/> </wsdl:output>
39:      </wsdl:operation>
40:    </wsdl:binding>
41:    <wsdl:service name="servicioIdentificacion">
42:      <wsdl:documentation>Un servicio de identificación</wsdl:documentation>
43:      <wsdl:port name="identificacion" binding="identificacionEnlace">
44:        <soap:address location="http://sitiofictitio.es/servlet/control.Id"/>
45:      </wsdl:port>
46:    </wsdl:service>
47: </wsdl:definitions>
```

Figura 1: Ejemplo de un servicio web definido en WSDL

junto con un nombre de servicio al que encapsula (llamado `servicioIdentificacion` en el ejemplo). Los espacios de nombres (`xlmns` y `targetNamespace`) son necesarios para establecer el sitio de Internet de nuestro servicio web (`targetNamespace="http://sitiofictitio.es/servicioIdentificacion.wsdl"`) y para establecer los sitios Internet donde residen los esquemas base de SOAP y los esquemas base para documentar WSDL. Estos dos espacios de nombres residen por omisión en `http://schemas.xmlsoap.org/wsdl/soap/` y en `http://schemas.xmlsoap.org/wsdl/`, respectivamente, y por tanto, pueden ser omitidos en la definición del documento WSDL.

Los tipos de datos utilizados por las operaciones del servicio (sólo una, la operación `identificar`) se definen dentro de la sección XML `<types>` (entre la línea 6 y la 21). Como se ha adelantado antes, los tipos de datos se definen utilizando un documento de esquema del W3C (la sección `<schema>` entre la línea 7 a la 20). Como sucede para el ejemplo presentado, los tipos se pueden declarar directamente dentro del documento (en la sección `<types`), pero también pueden ser declarados en un documento externo y luego utilizar un puntero hacia el documento. Esto se hace con la cláusula de esquema WSDL `<import>`, por ejemplo `<import location="http://sitiofictitio.es/identificacionTipos.xsd">`.

```
<wsdl:binding name="identificacionEnlace" type="identificacionTipoPuerto">
  <wsdl:operation name="identificar">
    <soap:operation soapAction="http://sitiofictcio.es/identificar">
    <wsdl:input> <soap:Body use="literal"/> </wsdl:input>
    <wsdl:output>
      <mime:part>
        <mime:content part="datosSalida" type="text/html"/>
      </mime:part>
  </wsdl:operation>
</wsdl:binding>
```

Figura 2: Ejemplo de un protocolo de transporte MIME usado en un documento WSDL

Los mensajes que envía y acepta el servicio se definen primero dentro de dos secciones `<message>` (líneas 22 y 25). En el ejemplo se han definido los mensajes `datosEntrada` y `datosSalida`. Estos mensajes son luego utilizados por la operación `Identificar` (líneas 30 y 31), definida dentro de un tipo de puerto `identificacionTipoPuerto` (línea 28). Un tipo de puerto equivale a una interfaz de un componente. Por tanto, si nuestro servicio tuviera más de una interfaz, habría que definir tantos tipos de puertos como interfaces existentes. De forma parecida, el tipo de puerto que se muestra en nuestro ejemplo (`identificacionTipoPuerto`) tan solo contiene una sola operación, pero en caso de existir más de una habría que declarar tantas secciones `<operation>` como operaciones tenga el tipo de puerto (la interfaz).

La sección `<binding>` (entre las líneas 34 y 40) está relacionada con los protocolos del servicio. Los protocolos están relacionados con la forma en la que los mensajes son enviados en una petición y/o respuesta SOAP[1]. La forma de estos mensajes de entrada y de salida se establecen en la sección `<soap:Body>`. Aquí, el atributo `use` puede tomar el valor `"literal"` o el valor `"encoded"`. Generalmente se utiliza el primero de ellos para indicar que los mensajes se envían/reciben (`<input>`, `<output>`) respetando "literalmente" su definición, establecida en las secciones de `<message>`. En nuestro ejemplo esto quiere decir que para un mensaje SOAP de petición el valor de DNI irá literalmente dentro del mensaje XML SOAP. Cuando el valor es `"encoded"` entonces se refiere a que la forma en la que se envían/reciben los mensajes siguen un protocolo de transporte (HTTP, SMTP, MIME, FTP). Por ejemplo, en la figura 2 se muestra de nuevo la sección `<binding>` del ejemplo del servicio de identificación, pero ahora suponiendo que las salidas (sección `<output>`) se envían en una página HTML. Para ello, como vemos en la figura, se utiliza el protocolo MIME especificando el tipo "mime" de una página web (que es `text/html`).

El documento WSDL concluye con la descripción del servicio web (líneas 41 a la 46). Un servicio queda identificado por un nombre, por uno o más puertos, y por su ubicación en la red donde puede ser llamado. En nuestro ejemplo el servicio está compuesto por un solo puerto llamado `identificacion` (línea 43), y se activa mediante una llamada a una "servlet" en la ubicación `http://sitiofictcio.es/servlet/control.Identificacion` (línea 44).

Para mayor información sobre la especificación WSDL puede consultar en la página web del W3C (`http://www.w3c.org/TR/wsdl`) o también en [12] o en [23] (por citar tan sólo tres fuentes).

[1]Es necesario aclarar que el concepto de protocolo aquí no coincide con el concepto de protocolo en el área de los componentes software. En este caso nos estamos refiriendo al protocolo de transporte de los datos. Para el caso de los componentes software, un protocolos se refiere al orden en el que los mensajes son enviados y llamados.

4. Publicación y consulta de servicios

Al mismo tiempo que IBM y Microsoft se unieron para desarrollar una especificación de lenguaje para definir servicios web (el lenguaje WSDL), ambas compañías (junto con Ariba) también crearon otro grupo de trabajo con la idea de desarrollar una especificación de una función de localización y almacenamiento de servicios web. Al grupo de trabajo lo llamaron "Grupo del Proyecto UDDI" (`http://UDDI.org`). En septiembre del año 2000 este grupo crea la primera especificación UDDI del mercado, y nueve meses después, en junio de 2001, crea la versión 2.0. En la actualidad existe la versión 3.0, publicada a finales de junio del año 2002. El grupo de trabajo es mantenido por la organización OASIS en su enlace `http://www.uddi.org` y en el cual colaboran más de 200 organizaciones.

Aunque rápidamente aparecieron en el mercado implementaciones de la especificación UDDI, como UDDI Services .NET Server de Microsoft o CapeConnect de CapeSoftware (entre otros), no ha sido tan rápida la aparición de uno o varios sitios web conocidos, importantes y con la infraestructura necesaria para dar soporte a repositorios de servicios web que sigan la especificación UDDI. Esta laguna la aprovecharon otras compañías, como SalCentral (`http://www.salcentral.com`), BindingPoint (`http://www.bindingpoint.com`) y XMethods (`http://www.xmethods.com`), para crear importantes repositorios de servicios web accesibles por Internet. El inconveniente es que estos repositorios no siguen el modelo de especificación UDDI, y simplemente se limitan a hospedaje de documentos WSDL, utilizando modelos de almacenamiento y consulta propios a cada organización. Hoy día estos repositorios de servicios web siguen siendo un punto de referencia muy importante para los desarrolladores de aplicaciones basadas en componentes web. El más importante quizás sea SalCentral (`http://www.salcentral.com`), donde en los últimos meses se ha notado un aumento en servicios desarrollados por importantes compañías.

Pero a finales del año 2002 se ha presenciado la consolidación de las primeras implementaciones y mantenimiento de repositorios web basados en el modelo UDDI. A la infraestructura que soporta y mantiene un repositorio de servicios web UDDI se la conoce con el nombre de UBR (*UDDI Business Registry*). Los repositorios UBR más importantes son precisamente aquellos mantenidos por las compañías pioneras en el modelo UDDI: IBM y Microsoft. Aquellas compañías que mantienen y ofrecen un punto de entrada por Internet a repositorios UBR reciben el nombre de "operadores UDDI" (*UDDI operator node*). En la tabla 1 se ha recopilado algunos de los operadores UDDI más importantes.

Operador	URL de publicación	URL de búsqueda
HP	`https://uddi.hp.com/publish`	`http://uddi.hp.com/inquire`
IBM	`https://uddi.ibm.com/ubr/publishapi`	`http://uddi.ibm.com/ubr/inquiryapi`
Microsoft	`https://uddi.microsoft.com/publish`	`http://uddi.microsoft.com/inquiry`
NTT	`https://www.uddi.ne.jp/ubr/publishapi`	`http://www.uddi.ne.jp/br/inquiryapi`
SAP	`https://uddi.sap.com/uddi/api/publish`	`http://uddi.sap.com/uddi/api/inquire`
Systinet	`https://www.systinet.com/wasp/uddi/publish`	`http://www.systinet.com/wasp/uddi/inquiry`
Algunos operadores de repositorios UDDI de pruebas		
IBM	`https://uddi.ibm.com/testregistry/publish`	`http://uddi.ibm.com/testregistry/inquiry`
Microsoft	`https://test.uddi.microsoft.com/publish`	`http://test.uddi.microsoft.com/inquiry`
SAP	`https://udditest.sap.com/UDDI/api/publish`	`http://udditest.sap.com/UDDI/api/inquire`

Cuadro 1: Algunos puntos de acceso a operadores UDDI

Un UBR puede ser mantenido de forma común por uno o mas operadores, cada uno de ellos con su propio repositorio. UDDI permite hacer duplicados de un servicio publicado en los repositorios afiliados para mantener así la consistencia del UBR. Por tanto, un servicio publicado en el operador A mas tarde puede ser consultado en el operador B, ya que su repositorio asociado se actualiza con un duplicado automático del servicio publicado en el operador A. Los duplicados no son automáticos, siendo el tiempo mínimo de actualización de 12 horas. Como ejemplo, IBM y Microsoft mantienen de forma conjunta un UBR.

UDDI se basa en un modelo "suscribir/publicar/preguntar" para registrar y localizar objetos. Para la publicación de servicios web, UDDI requiere que el proveedor de servicios tenga que estar suscrito previamente en el operador. Para la consulta no es necesario estar suscrito. En la tabla 1 se ha incluido dos columnas: el punto de entrada por Internet para hacer publicaciones, y el de entrada para hacer consultas. Además, como se puede observar, los puntos de entrada de publicación de los operadores UDDI requieren una conexión segura `https` con una identificación del proveedor. En los puntos de entrada de consulta, directamente se pueden hacer búsquedas en el repositorio.

En las últimas filas de la tabla 1 se ha incluido algunos puntos de entrada a operadores UDDI que ofrecen repositorios de pruebas, denominados "repositorios UDDI de desarrollo". Esta clase de repositorio es muy útil para los desarrolladores de servicios web para labores de construcción y validación. Los desarrolladores pueden registrar sus prototipos de servicio en los repositorios UDDI de pruebas para validar sus servicios, siempre de forma privada en su cuenta particular. Cuando estos servicios han sido suficientemente probados y validados, el propio desarrollador los publica luego en otro repositorio denominado "repositorio UDDI de producción". Los servicios publicados en esta clase de repositorio pueden ser consultados por otros desarrolladores o clientes de servicios web.

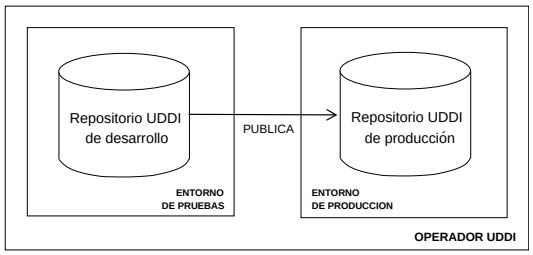

Figura 3: Entorno de pruebas y de producción de un operador UDDI

El modelo de información de un repositorio UDDI básicamente almacena información de aquellas organizaciones que publican servicios web en el repositorio. Los documentos WSDL no se almacenan en el repositorio UDDI; estos permanecen en un sitio web al que apunta el registro publicado. Conceptualmente, una empresa proveedora puede publicar o consultar su registro de información de forma similar a los tres tipos de un listín telefónico:

a) *Páginas blancas*. Contiene información básica de contacto, como el nombre de la empresa, direcciones postales y de correo electrónico, contactos personales, números de teléfono, y un identificador único. Los identificadores son valores alfabéticos o numéricos que distinguen de forma única a una empresa proveedora. Un identificador se asigna cuando la empresa se suscribe en el operador UDDI.

b) *Páginas amarillas*. Agrupa a las empresas registradas por categorías, en función del identificador asignado y de los tipos de servicios ofertados. Esta opción usa un catálogo de servicios web durante el proceso de almacenamiento y de consulta. Esta clase es muy útil para hacer búsquedas selectivas de servicios web dentro de una o varias categorías seleccionadas, reduciendo el tiempo de respuesta.

c) *Páginas verdes*. Contienen información técnica sobre los servicios que una empresa proporciona. Esta información es muy útil para que un objeto cliente de servicio pueda conocer detalles de conexión y de programación de comunicación con el servicio web, pues esta información define cómo invocar al servicio (como vimos en la sección anterior). Las páginas verdes normalmente incluyen referencias a los documentos WSDL, que contienen información sobre cómo interactuar con el servicio web.

Como ya se ha adelantado antes, la especificación UDDI requiere que la empresa proveedora de servicios web esté suscrita en el operador donde desea acceder. Cuando se accede por primera vez, UDDI suscribe al proveedor, solicitando su dirección de correo electrónico y una contraseña. Internamente, UDDI genera una clave única siguiendo algún modelo para la generación de claves, y crea un catálogo (un espacio) para el proveedor, con derechos para modificar, dar de alta y de baja a sus servicios web. Al catálogo creado (una única vez) para una empresa proveedora se le denomina documento `businessEntity`. Dentro, la empresa proveedora podrá incluir nuevos servicios web cuando ésta lo desee, estableciendo previamente una conexión segura (`https`), figura 1.

UDDI está compuesto básicamente por cinco tipos de información: (a) `businessEntity`, información acerca de la empresa; (b) `businessService`, información acerca de los servicios que la empresa ofrece; (c) `bindingTemplate`, información técnica de un servicio; (d) `tModel`, información sobre cómo interactuar con el servicio web; (e) `publisherAssertion`, información acerca de la relación de la empresa con otras. Un registro UDDI es un documento XML que incluye estos cinco tipos de información mencionados. En la figura 4 mostramos un meta-modelo de información que se ha creado a partir de la especificación de la estructura de datos de UDDI 3.0 [7]. La sección `businessEntity` encapsula a los demás tipos, salvo `publisherAssertion` y las definiciones WSDL de los servicios web. Cuando la empresa proveedora se suscribe en el operador UDDI, se crea un documento `businessEntity` para ella, con cierta información de empresa (como su nombre o dirección de contacto).

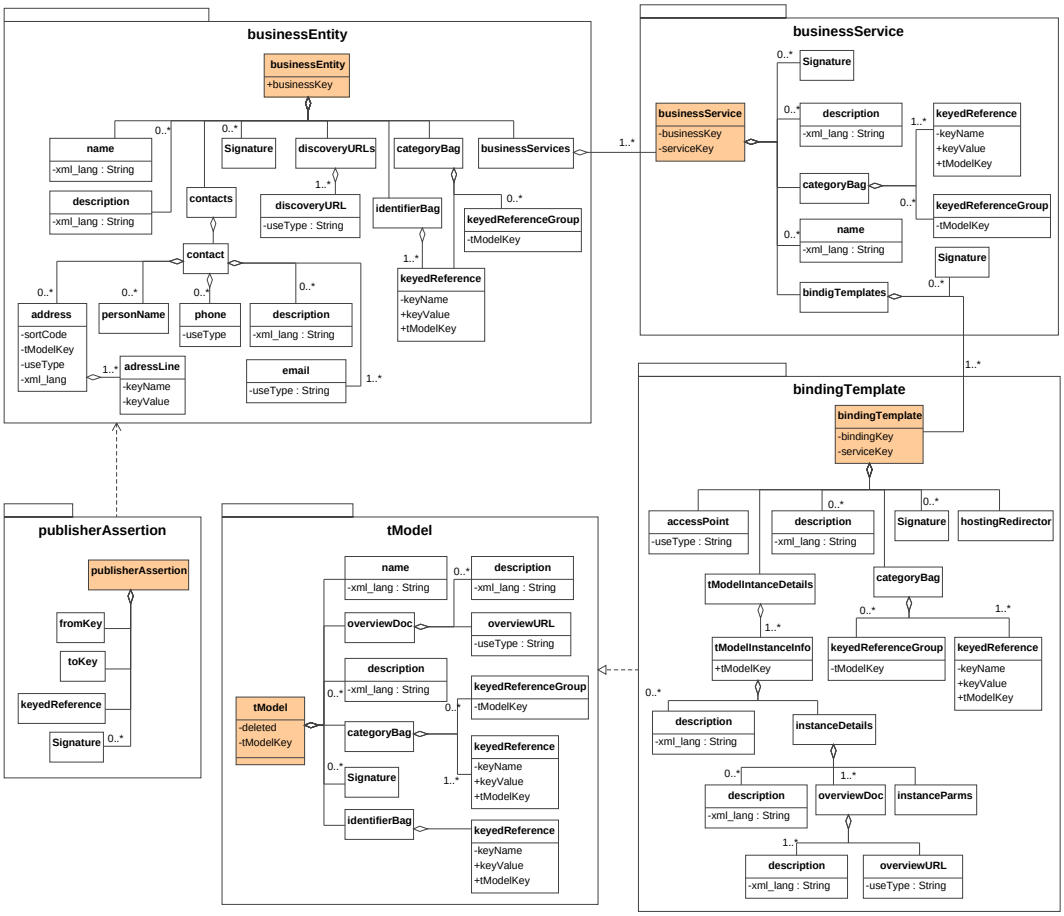

Figura 4: Arquitectura interna de un registro UDDI

Cuando la empresa proveedora realiza su primer registro de servicio web en UDDI, dentro de `businessEntity` se crea una sección `businessServices`, y dentro de ésta se crea una sección `businessService` por cada servicio web publicado.

Cuando se lleva a cabo el almacenamiento del servicio web en el repositorio UDDI, la información de servicio se desglosa en las secciones técnicas `bindingTemplate` y en un documento XML externo `tModel`. Un documento `tModel` contiene un enlace hacia la definición WSDL del servicio web que describe cómo se hacen las conexiones con las operaciones del servicio. Un documento `tModel` es considerado como un tipo de servicio y que puede ser reutilizado por la empresa proveedora para publicar otros servicios web.

Para analizar mejor este meta-modelo y algunos conceptos mencionados anteriormente acerca del modelo de información UDDI, en la figura 5 mostramos un ejemplo de documento UDDI en XML que se corresponde con una instancia del meta-modelo de la figura 4. Para analizar el modelo de información de UDDI haremos referencia de forma conjunta a estas dos figuras (4 y 5).

```
1:  <businessEntity businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
2:    <name>SitioFicticio S.A.</name>
3:    <description xml:lang="es">Empresa de servicios para correo electrónico</description>
4:    <contacts>
5:      <contact>
6:        <personName>Director técnico</personName>
7:        <phone>+34-950-123456</phone>
8:        <email>manager@sitioficticio.es</email>
9:        <address> <addressLine>Ctra. Sacramento s/n 04120</addressLine> </address>
10:     </contact>
11:   </contacts>
12:   <businessServices>
13:     <businessService businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
14:                      serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
15:       <name>Servicio de Identificacion</name>
16:       <description xml:lang="es">Un servicio de identificacion Internet</description>
17:       <categoryBag>
18:         <keyedReference keyName="NAICS: e-commerce" keyValue="..." tModelKey="... "/>
19:         <keyedReference keyName="NAICS: Database soft." keyValue="..." tModelKey="... "/>
20:         ...
21:       </categoryBag>
22:       <bindingTemplates>
23:         <bindingTemplate bindingKey="6D8F8DF0-3AAF-11D5-80DC-002035229C64"
24:                          serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
25:           <accessPoint>http://sitioficticio.es/servlet/control.Identificacion</accessPoint>
26:           <tModelInstanceDetails>
27:             <tModelInstanceInfo tModelKey="564B5113-1BAE-21A3-906E-122035229C75"/>
28:           </tModelInstanceDetails>
29:         </bindingTemplate>
30:         ...
31:       </bindingTemplates>
32:     </businessService>
33:     ...
34:   </businessServices>
35: </businessEntity>
```

Figura 5: Un documento XML como registro UDDI

En el diagrama se ven las partes de un documento XML `businessEntity`. Cada clase UML modela una etiqueta del documento XML. Se ha utilizado el apartado de atributos de clase para modelar los atributos de una etiqueta. El signo "`-`" significa que el atributo es opcional, y "`+`" significa que el atributo es obligatorio. Por ejemplo, la clase `businessEntity` tiene un atributo requerido llamado `businessKey`, que representa la clave única de la empresa proveedora, asignada cuando realizó su suscripción en el operador UDDI. Esta clase modela la etiqueta `businessEntity` de un documento UDDI (línea 1, figura 5).

Como información general, el documento `businessEntity` contiene: el nombre de la

empresa proveedora (línea 2), una descripción de la empresa (línea 3) y datos de contacto (líneas 4 a la 11). Esta información se corresponde con la de "páginas blancas".

Los servicios web se registran dentro de la sección `businessServices` (entre las líneas 12 y 34). Dentro de ésta, se utiliza una sección `businessService` para definir cada servicio web publicado por la empresa proveedora (líneas 13 a la 32). Para cada servicio se utilizan dos atributos: uno representa la clave única de la empresa proveedora, y el otro representa la clave única del servicio web publicado. Para este último, la asignación de claves para los servicios publicados siguen el mismo criterio que para la asignación de claves para las empresas proveedoras (la clave `businessKey`).

Para cada servicio definido en la sección `businessService`, se indica: un nombre (línea 15), una descripción del servicio (línea 16) e información de catalogación (entre líneas 17 y 21). Toda esta información se corresponde con la de "páginas amarillas".

Además, una sección de servicio `businessService` puede contener información técnica, recogida en la sección `bindingTemplates` (entre las líneas 22 a la 31). Esta información se corresponde con información de "páginas verdes". La sección `bindingTemplates` contempla información de conexión (línea 25) y un enlace al tipo del servicio `tModel` que está almacenado en otro documento XML dentro del repositorio del operador UDDI (26-28). La etiqueta `tModelInstanceInfo` contiene un atributo con la clave del tipo de servicio `tModel`.

Dentro de un documento `tModel` existe un puntero hacia la definición WSDL (un archivo "`.wsdl`") que contiene la descripción acerca de cómo se debe invocar las operaciones del servicio, y cómo estas devuelven los datos.

UDDI además cuenta con diferentes interfaces funcionales que permiten la interacción con los distintos objetos cliente: (a) el objeto que publica servicios web (*publisher*), y (b) el objeto que consulta servicios (*inquiry*). Las llamadas y las respuestas a todas las operaciones UDDI se realizan utilizando documentos XML. En la tabla 2 mostramos los tipos de mensajes de petición y respuesta para las operaciones de las interfaces de publicación y consulta (`Publisher` y `Inquiry`).

Publicación		Consulta	
Petición	Respuesta	Petición	Respuesta
<add_publisherAssertion>	<dispositionReport>	<find_binding>	<bindingDetail>
<delete_binding>	<dispositionReport>	<find_business>	<businessList>
<delete_business>	<dispositionReport>	<find_relatedBusinesses>	<relatedBusinessesList>
<delete_publisherAssertions>	<dispositionReport>	<find_service>	<serviceList>
<delete_service>	<dispositionReport>	<find_tModel>	<tModelList>
<delete_tModel>	<dispositionReport>	<get_bindingDetail>	<bindingDetail>
<discard_authToken>	<dispositionReport>	<get_businessDetail>	<businessDetail>
<get_assertionStatusReport>	<assertionStatusReport>	<get_serviceDetail>	<serviceDetail>
<get_authToken>	<authToken>	<get_tModel>	<tModelDetail>
<get_publisherAssertions>	<publisherAssertions>		
<get_registeredInfo>	<registeredInfo>		
<save_binding>	<bindingDetail>		
<save_business>	<businessDetail>		
<save_service>	<serviceDetail>		
<save_tModel>	<tModelDetail>		
<set_publisherAssertions>	<publisherAssertions>		

Cuadro 2: Ordenes de publicación y de consulta en el repositorio UDDI

Las operaciones de la interfaz de publicación utilizan unos documentos XML como mensajes para almacenar, modificar, borrar o eliminar información de páginas blancas, amarillas o verdes. Las operaciones de la interfaz de consulta también utilizan documentos XML, en este caso para interrogar en los tres tipos de páginas. Los documentos de consulta del estilo `<find_xxx>` extraen una colección de documentos que cumplen con las restricciones impuestas dentro del documento de consulta. La colección de documentos devueltos son encapsulados por una etiqueta global, mostradas en la cuarta columna de la tabla 2. Los documentos de consulta del estilo `<get_xxx>` extraen información detallada.

En la tabla 3 mostramos algunos ejemplos de uso de algunas operaciones de consulta, junto con los resultados que estas generan. Como vemos en estos ejemplos, podemos ir refinando la búsqueda deseada aplicando documentos de consulta sobre aquellos documentos devueltos en consultas anteriores, limitando cada vez más el espacio de la búsqueda. En los ejemplos de la tabla comenzamos buscando todos los servicios publicados por la empresa proveedora "SitioFicticio S.A.". Luego, sobre el resultado devuelto, queremos conocer si dicha empresa ha publicado un "Servicio de Identificacion". UDDI devuelve sólo una referencia al servicio, para indicar que lo ha encontrado. Finalmente, en la última consulta pedimos a UDDI que obtenga una descripción más detallada.

Consulta 1: busca todos los servicios de una empresa proveedora

```
1:  <find_business">
2:     <name>SitioFicticio S.A.</name>
3:  </find_business>
```

Resultado 1: la lista de servicios

```
4:  <businessList>
5:     <businessInfos>
6:        <businessInfo businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
7:           <name>SitioFicticio S.A.</name>
8:           <description xml:lang="es">Empresa de servicios para correo electrónico</description>
9:           <serviceInfos>
10:             <serviceInfo businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
11:                          serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
12:                <name>Servicio de Identificacion</name>
13:             </serviceInfo>
14:             ...
15:          </serviceInfos>
16:       </businessInfo>
17:    <businessInfos>
18: </businessList>
```

Consulta 2: ahora busca un servicio concretro dentro de la lista de servicios

```
19: <find_service businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
20:    <name>Servicio de Identificacion</name>
21: </find_service>
```

Resultado 2: una referencia al servicio buscado

```
22: <serviceList>
23:    <serviceInfos>
24:       <serviceInfo businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
25:                    serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
26:          <name>Servicio de Identificacion</name>
27:       </serviceInfo>
28:    </serviceInfos>
29: </serviceList>
```

Consulta 3: busca información más detallada del servicio

```
30: <get_serviceDetail>
31:    <serviceKey>894B5100-3AAF-11D5-80DC-002035229C64</serviceKey>
32: </get_serviceDetail>
```

Resultado 3: la información de servicio

```
33: <serviceDetail>
34:    <businessService businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
35:       Véase lineas 13 a la 32 de la figura 5
36:    </businessService>
37: </serviceDetail>
```

Cuadro 3: Diferentes ejemplos de consulta en UDDI

Debido a la extensión del informe de especificación de UDDI 3.0 (el apartado de las estructuras de datos ocupa más de 300 páginas) aquí sólo se ha descrito aquellas etiquetas de documento más significativas. Una descripción bastante detallada del modelo de información se puede encontrar en [7] [9] [10].

5. Limitaciones de WSDL y UDDI

Las limitaciones han sido detectadas en base a la experiencia obtenida tras analizar la especificación UDDI 3.0 [7] y algunas de las implementaciones industriales del servicio UDDI, como la API UDDI4J de IBM (`http://www-124.ibm.com/developerworks/oss/uddi4j`), y los *UBR* de Microsoft (`http://uddi.ibm.com`), IBM (`http://uddi.ibm.com`) y HP (`http://uddi.hp.com`). Además, también se han analizado otros repositorios de servicios web que no siguen el modelo UDDI, como el repositorio de SalCentral (`http://www.salcentral.com`) y el repositorio de XMethods (`http://www.xmethods.com`).

Por su corto espacio de tiempo en el mercado, pensamos que la técnica UDDI para la actividad de publicación y localización en entornos de ensamblaje de servicios web está aún por madurar y que debe cubrir ciertas lagunas que a continuación vamos a discutir. Aunque también es cierto que algunas de estas limitaciones son debidas en parte al modelo de especificación de servicios web (WSDL) que utiliza UDDI a la hora de hacer las operaciones de registro y consulta. Veamos entonces primero estas limitaciones del modelo WSDL, y luego destacaremos cómo influyen éstas en el modelo UDDI.

WSDL es un lenguaje muy útil para la especificación de componentes simples que funcionan por Internet, y como vimos, a esta clase de componente se le denomina *servicio web*. Normalmente estos servicios han estado encapsulados en páginas web, desde donde los clientes acceden de forma usual utilizando un navegador de Internet. Con el tiempo, y debido al avance de las nuevas tecnologías web (XML y SOAP básicamente), surge la figura del programador de aplicaciones basadas en Internet. Este tipo de desarrollador de software necesita conocer ciertos detalles de implementación de un servicio web definidos en sus interfaces, como por ejemplo la forma en la que se llaman las operaciones, o la forma en la que se devuelven los datos, o las restricciones de uso del servicio, por citar solo algunos detalles de implementación de una interfaz.

Por esta razón, y dado que XML se impone en los últimos años como un estándar para el intercambio uniforme de datos en Internet (además de su utilidad en otras actividades y campos de la informática), surge WSDL como un lenguaje basado en XML para la especificación de servicios web. En la sección 3 se ha dedicado un apartado donde se describen algunos detalles del lenguaje WSDL.

Como vimos, un documento WSDL está compuesto por siete elementos XML que definen detalles de la interfaz (o interfaces) de un servicio, como las signaturas y tipos de datos de entrada y salida de las operaciones, o la forma en la que se llama las operaciones y su devolución, y detalles de conexión. Sin embargo, la especificación de WDSL no contempla los siguientes aspectos:

(a) *Protocolos*. WSDL utiliza el término protocolo para referirse al protocolo de transporte (HTTP, MIME, FTP, SMTP). Sin embargo, el esquema XML de WSDL no contempla una notación para referirse al concepto de protocolo (o coreografía) en el sentido de conocer el orden en el que se llaman a las operaciones de entrada y de salida. Por tanto, el lenguaje debe permitir incluir directamente (en XML) o indirectamente (un puntero externo) notación para la definición de protocolos: por ejemplo, un protocolo definido en SDL o π-cálculo.

(b) *Comportamiento*. Es necesario que el lenguaje permita definir también el comportamiento semántico de las operaciones (pre/post condiciones e invariantes), y no solo la definición sintáctica de las signaturas usando etiquetas XML. Por tanto, como antes, el lenguaje debe permitir incluir directamente (en XML) o indirectamente (un puntero externo) notación semántica: por ejemplo, una definición en Larch.

(c) *Información extra-funcional*. Por último, también es necesario que el lenguaje pueda utilizar una notación para definir propiedades extra-funcionales, además de la información funcional (sintaxis, semántica y protocolos de las interfaces) y la información

técnica del servicio. De nuevo, debe permitir incluir directa o indirectamente esta clase de información: por ejemplo, propiedades de la forma nombre, tipo, valor.

Como se ha adelantado antes, todas estas limitaciones del lenguaje WSDL, de alguna forma, también repercuten como limitaciones en el modelo UDDI, y básicamente afectan a las operaciones de búsqueda en el repositorio. Por lo tanto, una primera limitación de UDDI es que las operaciones de búsqueda están sujetas sólo a aspectos técnicos de una empresa proveedora de servicios web, a aspectos técnicos de los propios servicios web, y también aspectos de localización, conexión y comunicación de las operaciones de un servicio web. Sin embargo, estas operaciones no permiten búsquedas sobre: (a) los protocolos de interacción de las operaciones, (b) el comportamiento de las operaciones; y (c) información extra-funcional.

En segundo lugar, aunque la especificación UDDI permite el concepto de afiliación (`publisherAssertion`), esto no contempla realmente la posibilidad de federación de repositorios UDDI en la línea a como lo hace por ejemplo el servicio de mediación estándar de ODP [22]. Como vimos, la afiliación de operadores UDDI permite mantener la consistencia de la información de un único repositorio virtual UBR (*UDDI Business Registry*). Cada vez que se publica un servicio en el repositorio de un operador UDDI, éste (el servicio) se duplica en los repositorios afiliados al UBR. Sin embargo, un operador filial puede estar asociado a más de un UBR y esto puede acarrear ciertos problemas.

Por ejemplo, supongamos la existencia de dos UBRs (*UBR1* y *UBR2*). En *UBR1* están afiliados los operadores *A* y *B*, y en *UBR2* están afiliados los operadores *B* y *C*. Como vemos, el operador *B* está afiliado a los dos UBRs. Según esto, la publicación de un servicio en *UBR1* no implica que éste (el servicio) sea también publicado en el *UBR2* mediante un duplicado (y viceversa); aun estando el operador *B* como filial en las dos UBRs.

Además, tampoco está permitida la propagación de consultas entre los operadores de un UBR (la consulta se realiza sobre el operador y no sobre el UBR). Lo único que se propaga es un duplicado del servicio publicado en un operador hacia sus operadores afiliados. Siguiendo con el ejemplo, si suponemos que las publicaciones se llevan a cabo con más regularidad sobre el *UBR1*, un cliente *X* de servicios web podrá hacer consultas indistintamente desde el operador *A* o desde el operador *B*, aun desconociendo éste que los dos operadores son afiliados, y que (en principio) debería tener los mismos servicios registrados. Sin embargo, las consultas que realice otro cliente *Y* en el operador *C*, sólo se limitan al repositorio de dicho operador, y no se propagan a *B*, supuestamente con más información.

En este sentido, UDDI ofrece un útil y completo servicio de directorio, pero no un servicio de mediación. Un servicio de mediación puede ser considerado como un servicio de directorio avanzado que permite búsquedas basadas en atributos, como por ejemplo atributos de calidad [24].

6. Conclusiones y trabajo futuro

Al igual que los componentes software en sus inicios han sido definidos mediante una única interfaz, los servicios web actualmente encapsulan la funcionalidad de un objeto individual que se activa y responde por web. Sin embargo, al igual que los recientes modelos de componentes permiten la definición de múliples interfaces ofrecidas y requeridas por un componente y donde la comunidad de componentes parece que empieza a tener mas o menos claro qué tipo de información es necesaria para describir un componente software (sintáctica, semántica, protocolos y extra-funcional) para desarrollar sistemas mediante el acoplamiento de múltiples componentes, los servicios web también deberían contemplar esta clase de información, útil para los ingenieros cuando construyan sus sistemas a partir de colecciones de servicios web que supuestamente podrían también interoperar y/o estar desarrollados por terceras partes.

El problema básico es que, a diferencia de los componentes software, donde no parece estar extendida la forma sobre cómo recoger esa información del componente, en el caso de los servicios web sí que existe un lenguaje extendido basado en XML para la descripción de estos servicios, el lenguaje WSDL. Sin embargo, como se ha estudiado en este trabajo, tanto la actual notación para la descripción de servicios WSDL como la especificación del servicio de mediación UDDI, usados para las actividades de descripción, publicación y localización de servicios, no están preparadas para ser utilizados en procesos tradicionales del desarrollo de software basado en componentes, esto es, mediante ensamblaje de diferentes partes.

Aunque no se realiza ninguna propuesta en este sentido, en el trabajo presentado se realiza un estudio de las técnicas WSDL y UDDI, usadas en el desarrollo de software basado en servicios web, y se identifica un problema en la concepción de estas técnicas para el caso de la construcción de sistemas complejos a partir de múltiples servicios web que interoperan.

Como línea de trabajo futuro, se deberían extender las actuales notaciones WSDL y UDDI para que estas puedan realmente ser utilizadas en procesos DSBC. También sería interesante contemplar la federación entre repositorios WSDL que respetan la especificación UDDI y los que no, al estilo de los repositorios de Salcentral, que implementan repositorio WSDL particulares. Por último, debido a que presumiblemente puede aparecer, en un corto espacio de tiempo, un mercado de servicios web, también sería interesante relacionar los servicios web con los componentes COTS, permitiendo la conexión con servicios de mediación de componentes comerciales, como el servicio COTStrader [20].

Referencias

[1] F. Acherman, M. Lumpe, J. Schneider, and O. Nierstrasz. Piccola—A Small Composition Language, 1999. `http://www.iam.unibe.ch/~scg/Research/Piccola/`.

[2] A. Alencar and J. Goguen. OOZE. In S. Stepney, R. Barden, and D. Cooper, editors, *Object Orientation in Z*, pages 158–183. Springer-Verlag: Cambridge CB2 1LQ, UK, 1992. Workshops in Computing.

[3] M. Azuma. Square the next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality. *ESCOM (European Software Control and Metrics conference)*, April 2001. `http://www.escom.co.uk/conference2001/papers/azuma.pdf`.

[4] J. Barnes. *High Integrity Ada: the SPARK Approach*. Addison-Wesley, 1997. ISBN: 0-20-11751-77.

[5] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998. ISBN: 0-201-19930-0.

[6] R. Bastide, O. Sy, and P. Palanque. Formal Specification and Prototyping of CORBA Systems. In *ECOOP'99*, number 1628 in LNCS, pages 474–494. Springer-Verlag, 1999.

[7] T. Bellwood, L. Clent, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI Version 3.0, 19 de Julio 2002. Publicacin de especificacin. `http://www.uddi.org/pubs/uddi-v3.00-published-20020719.pdf`.

[8] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. Technical Report RFC 2396, IETF, August 1998. `http://www.ietf.org/rfc/rfc2396.txt`.

[9] T. Boubez, M. Hondo, C. Kurt, J. Rodriguez, and D. Rogers. UDDI Data Structure Reference V1.0, 28 de Junio 2002. Publicacin de especificacin. `http://www.uddi.org/pubs/DataStructure-V1.00-Published-20020628.pdf`.

[10] T. Boubez, M. Hondo, C. Kurt, J. Rodriguez, and D. Rogers. UDDI Programmer's API 1.0, 28 de Junio 2002. Publicacin de especificacin. `http://www.uddi.org/pubs/ProgrammersAPI-V1.01-Published-20020628.pdf`.

[11] C. Canal, L. Fuentes, E. Pimentel, J. M. Troya, and A. Vallecillo. Adding roles to CORBA objects. *IEEE Trans. Softw. Eng.*, 29(3):242–260, Feb. 2003.

[12] P. Cauldwell, R. Chawla, V. Chopra, G. Damschen, C. Dix, T. Hong, F.Ñorton, U. Ogbuji, G. Olander, M. A. Richman, K. Saunders, and Z. Zaev. *Professional XML Web Services*. Wrox Press Ltd, 2001. ISBN: 1-861005-09-1.

[13] L. Chung, B.Ñixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 1999. ISBN: 07-923-86663.

[14] K. K. Dhara and G. T. Leavens. Forcing Behavioral Subtyping Through Specification Inheritance. In *18th International Conference on Software Engineering (ICSE-18)*, pages 258–267, Berlin, Germany, 1996. IEEE Press.

[15] E. Drr and N. Plat. VDM++ Language Reference Manual, 1994. Utrecht, The Netherlands: Cap Volmac.

[16] R. Duke, G. Rose, and G. Smith. Object-Z: A Specification Language Advocated for the Description of Standards. *Computer Standards and Interfaces*, 17:511–533, Sept. 1995.

[17] J. Goguen, D.Ñguyen, J. Meseguer, Luqi, D. Zhang, and V. Berzins. Software Component Search. *Journal of Systems Integration*, 6:93–134, Sept. 1996.

[18] J. Han. Characterization of Components. In *International Workshop on Component-Based Software Engineering (CBSE'98)*, pages 39–42, Kyoto, Japan, April 1998.

[19] J. Han. Semantic and Usage Packaging for Software Components. In A. Vallecillo, J. Hernández, and J. M. Troya, editors, *Object Interoperability. ECOOP'99 Workshop on Object Interoperability*, pages 25–34, Lisbon, Portugal, 1999.

[20] L. Iribarne, J. M. Troya, and A. Vallecillo. A Trading Service for COTS Components. *The Computer Journal*, 2003. Aceptado con cambios menores y actualmente en segunda revision.

[21] ISO/IEC-9126. Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use, 1991. International Standard ISO/IEC 9126, Geneve, Switzerland: International Standards Organization/International Electrochemical Commision.

[22] ISO/IEC-ITU/T. Information Technology – Open Distributed Processing – Trading function: Specification, August 1997. ISO/IEC 13235-1, UIT-T X.950.

[23] T. Jewell and D. Chappell. *Java Web Services*. O'Reilly, 2002. ISBN: 0-596-00269-6.

[24] L. Kutvonen. Achieving interoperability through odp trading function. In *Second International Symposium on Autonomous Decentralized systems (ISADS'95)*, pages 63–69, Arizona, Apr. 1995. IEEE Computer Society.

[25] K. Lano, J. Bicarregui, T. Maibaum, and J. Fiadeiro. Composition of Reactive Systems Components. In *Proceedings of the 1st Workshop on Component-Based Systems*, European Software Engineering Conference (ESEC) y el ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), 1997. `http://www.cs.iastate.edu/~leavens/FoCBS/FoCBS.html`.

[26] D. Lea and J. Marlowe. Interface-Based Protocol Specification of Open Systems Using PSL. In *Proc. of the 9th European Conference on Object-Oriented Programming (ECOOP'95)*, pages 374–398, Aarhus, Dänemark, 1995.

[27] G. T. Leavens, L. Baker, and C. Ruby. *Behavioral Specifications of Businesses and Systems*, chapter JML: A Notation for Detail Desing. Kluwer Academic, 1999. `http://www.cs.iastate.edu/~leavens/JML.html`.

[28] M. Lumpe, J. Schneider, O.Ñierstrasz, and F. Achermann. Towards a Formal Composition Language. In *Proceedings of the 1st Workshop on Component-Based Systems*, European Software Engineering Conference (ESEC) y el ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), 1997. `http://www.cs.iastate.edu/~leavens/FoCBS/FoCBS.html`.

[29] B. Meyer. *Eiffel: The Language*. Prentice Hall, 1992. ISBN: 0-13-247925-7.

[30] A. Mikhajlova. *Ensuring Correctness of Object and Component Systems*. PhD thesis, Åbo Akademi University, October 1999.

[31] R. Riemenschneider and V. Stavridou. The Role of Architecture Description Languages in Component-Based Development: The SRI Perspective, 1999. `http://www.sei.cmu.edu/cbs/icse99/papers/42/42.htm`.

[32] C. Thompson. Workshop on Compositional Software Architectures: Workshop Report. 1998. `http://www.objs.com/workshops/ws9801/report.html`.

[33] A. Vallecillo, J. Hernández, and J. M. Troya. Object Interoperability. In *Object-Oriented Technology: ECOOP'99 Workshop Reader*, number 1743 in LNCS, pages 1–21. Springer-Verlag, 1999.

[34] W3C-WebServices. Web Services Glosary, November 2002. W3C Working Draft. `http://www.w3.org/TR/2002/WD-ws-gloss-20021114/`.

[35] J. Warmer and A. Kleppe. *The Object Constraint Language — Precise Modeling with UML*. Addison-Wesley, 1998. ISBN: 0-201-37940-6.

[36] D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, Mar. 1997.

[37] A. M. Zaremski and J. M. Wing. Specification Matching of Software Components. *ACM Trans. on Software Engineering and Methodology*, 6(4):333–369, Oct. 1997.