Web Components: A Comparison between Web Services and Software Components

Luis Iribarne[*]

Abstract

The software engineering discipline is experiencing a quick consolidation in the applications development activities and the use of technologies and methodologies of web-based programming. In the web services arena some oriented and based practices for the construction of large scale software applications (i.e., distributed information systems) are beginning to appear. The software development practices based on web services composition (many of them elaborated by third parts) involve similar problems to those in the component-based development arena (more extended in the COTS components), like components' compatibility or interoperability. In the present work, it is carried out a comparative study between the traditional software components description and the web services description language (WSDL). The work also shows the limitations of the current directory service specification (UDDI) to develop distributed software applications by means of the composition of web services.

Keywords: *Web services, software components, WSDL, UDDI, semantic specification, protocol specification.*

Resumen

La disciplina de la ingeniería del software está experimentando una rápida consolidación de las actividades de desarrollo de aplicaciones software y del uso de tecnologías y metodologías de programación basadas en web. En el campo de los servicios web además están apareciendo prácticas (orientadas y basadas) para la construcción de aplicaciones software a gran escala (como por ejemplo los sistemas de información distribuidos). Las prácticas para el desarrollo de software basadas en la composición de servicios web (muchos de estos desarrollados por terceras partes) conllevan problemas similares a los que aparecen en el campo del desarrollo de software basado en componentes (más extendido en el campo de los componentes COTS), como la interoperabilidad o la compatibilidad de componentes. El presente trabajo lleva a cabo un estudio comparativo entre la descripción tradicional de los componentes software y el lenguaje para la descripción de servicio web (WSDL). El trabajo también identifica algunas de las limitaciones que presenta la actual especificación del servicio de directorio (UDDI) para desarrollar aplicaciones software distribuidas mediante la composición de servicios webs.

Palabras clave: *Servicios web, componentes software, WSDL, UDDI, especificación semántica, especificación de protocolos.*

1 Introduction

Nowadays, large information systems and software applications are based on multi-tier architectural client/server models making use of a great variety of technologies. These systems

[*]Department of Lenguajes y Computación, University of Almería (Spain), `Luis.Iribarne@ual.es`

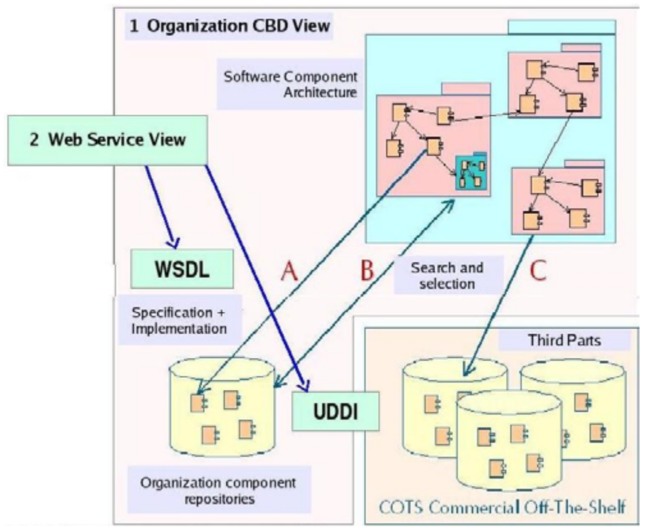

1 Organization CBD View
Software Component Architecture
2 Web Service View
WSDL
Specification + Implementation
A
B
Search and selection
C
Third Parts
UDDI
Organization component repositories
COTS Commercial Off-The-Shelf

Figure 1: A web service development view based on the traditional CBD view

are probably distributed and located in different geographical places, being communicated with distributed models like CORBA, EJB and/or DCOM, and making use of important rules, security methods and XML/XMI techniques for the intermediate representation of the software components information. The construction of these systems requires the use of traditional component-based development (CBD) practices and processes to build the systems from software components developed by third parts, known as commercial components or COTS components. These practices, used on both the construction of "oriented" and "based" systems in commercial components, are commonly identified by some important activities in CBD like the components description, or the publication and location of software components, usually carried out by trading services [19].

On the other hand, there has been a gradual consolidation of the web programming techniques (like WSDL, SOAP and UDDI) due to the increase in the use of web services to build (partial or totally) web-based information systems. Although these web services techniques also enable the description (WSDL), publication and location (UDDI) of specific services across Internet, they do not enable the programming of complex distributed information systems based on web services, in which one or more services require the presence of some other services to work with the composition of software components, just as it occurs in CBD.

As Fig. 1 shows, software development practices based on web services composition could involve similar problems to those present in COTS component-based development arena: software architecture descriptions, search and selection processes (i.e., web services' compatibility or interoperability), trading processes (more directly related to the publication and location of objects activities), or software (web services — components) reuse descriptions, among other practices.

The present work focuses on the analysis and comparison between the software description (specification) activity as well as the publication and location activity of the traditional component-based software development activities with those similar activities used in web services-based software development. This work also shows some limitations of these web services activities when building software systems following traditional CBD practices.

The work is finally structured as follows. Sections 2 and 3 analyze the traditional ways of describing software components and web services. Section 4 contains a study of the current UDDI directory service specification for location and publication of web services. Section 5 analyzes some WSDL and UDDI shortcomings for the development of applications based on web services. This work ends up with some conclusions and future works.

2 Software Components Description

The component-based development (CBD) is a paradigm to develop software, where all the appliances, from the source code to the interface specification models, architectures and business models, can be built by composing, adapting and installing together and following a variety of configurations. Nevertheless, this would not be possible without a clear description of the software components.

A software component requires some specification information for the users and developers of the module. In the software reuse context, a specification helps to determine whether a module fulfils the needs of a new system. In the interoperation context, the specification is used to determine if two modules are able to interoperate.

A software component C can be characterized as: "C = (Atr + Oper + Even) + Beh + (Prot * Esc) + Prop". Attributes (*Atr*), operations or methods (*Oper*) and events (*Even*) are a part of the component interface, and they represent their syntactical level. The behavior (*Beh*) of these operations represents the semantic level of the component. The protocols (*Prot*) determine the interoperability of the component with other components and the type of behavior that the component is going to have in different "settings" (*Esc*) where it expects to be executed. Finally, *Prop* refers to non-functional "properties" of the component, i.e., security, reliability or performance properties, among others. A component description is:

a) **Syntactic description.** A software component can be identified by one or more interfaces. An interface just contains syntactical information of a component's input and output operations with which interacts with other components. This kind of interaction is known as "proactive control", that is, the operations of a component are activated by means of another component's calls. Apart from the proactive control (the usual way of calling an operation) there is the "reactive control", which refers to the component's events, as the EJB (*Enterprise JavaBeans*) component model uses. In a reactive control, a component can generate events to reply to requested operations; afterwards other system components collect these requests and the call to the operation is activated.

b) **Semantic description.** Nevertheless, not all the operations invocation sequences are permitted. There exist operational restrictions that specify the possible operation patterns. The construction of applications does not involve the use of the traditional interface specifications. They only contain the name of the component signatures and attributes [35, 36]. It is necessary to include a semantic specification of the interfaces for the meaning of the operations and the description of their behavior [32]. The information at the "semantic" level can be described by using formalisms like the pre/post conditions, Larch [15], JML (*Java Modeling Language*) or JavaLarch (`ftp://ftp.cs.iastate.edu/pub/leavens/JML`) [26], and OCL (*Object Constraints Language*) [34]. Programming languages as Eiffel [28] and SPARK [4] enable to write specifications of behavior using pre/post conditions inside the code. Other formalisms for the semantic specification are the algebraic equations [17], the refinement calculus

[29], and other formal extensions of traditional object-oriented methods that are being used on software components specification, like OOZE [2], VDM++ [14] and Object-Z [16].

c) **Protocol description (choreography).** Besides the previous information, it is also necessary another kind of semantic information concerning with the order in which the component interface operations should be called. This semantic information is known as interaction "protocols" (also named "choreography"). Depending on the formalism, there exist different proposals to specify information of protocols: Petri Nets [6], temporal logic [18, 25], finite state machines [35] or the π-calculus [11, 27]. There exist some other languages for the synchronization of components (another way of referring to the protocols), like Object Calculus [24], Piccola [1] or ASDL (Architectural Style Description Language) [30].

d) **Quality and non-functional description.** Another important issue is the quality information of a component. This information is known in literature as attributes of quality. An attribute of quality refers both to the service quality information (i.e., the response maximum time, the average response and the precision) and the attributes related to the functionality and non-functionality of a component: for example the interoperability and the security for functional attributes, or the portability and the efficiency for non-functional attributes, among others [20]. However, most of the quality attributes are related to the non-functional information. The non-functional information is also called in the literature as non-functional restrictions (NFR) or "*ilities*" [5, 31] (to refer to "*reliability*", "*portability*", "*usability*", or "*predictability*" terms). In the component literature there exist two classical references for the "*ilities*": [3] and [13]. The first one is an ISO 9126 standard, and the second one collects and classifies more than 100 *ilities*.

3 Web Services Description

A web service is an application published, located and invoked from a web place or local network, which is based on Internet standards. It combines the better aspects of the web and component-based programming, independently of the language, operating system, and the component model in which it was implemented.

A more formal and extended definition is the one offered by the W3C Web Services working group [33]: "A web service is a software system identified by a URI (a uniform resources identifier) [8], whose public interfaces and links are defined and described in XML. Its definition can be located by other software systems that can interact with the web service in the pre-established way by its definition, using messages based on XML transmitted by protocols of Internet".

The XML and SOAP standards began to be used in web applications programming. Nevertheless, IBM and Microsoft also began to define web services in XML notation. In the early 2000, IBM developed the NASSL (*Network Accessibility Service Specification Language*), an XML-based language for the web services interfaces definition that used the W3C XML-Schemas notation to define the data types of the SOAP messages. On the other hand, in April of 2000, Microsoft launched to the market the SCL (*Service Contract Language*) and the XDR (*XML Dates-Reduced*). The first one is a language that also uses XML to define web service interfaces, and the second one is a language for data types definition. Few months later, Microsoft offered the SDL (*Services Description Language*) incorporated to the Visual Studio .NET.

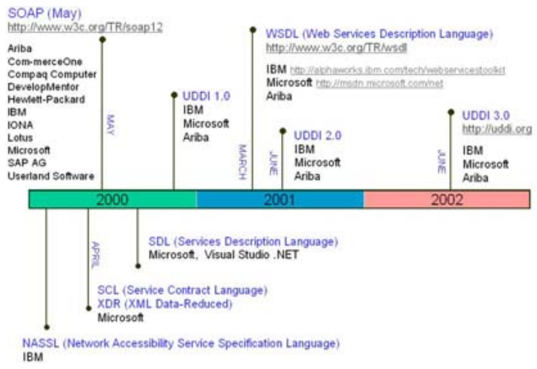
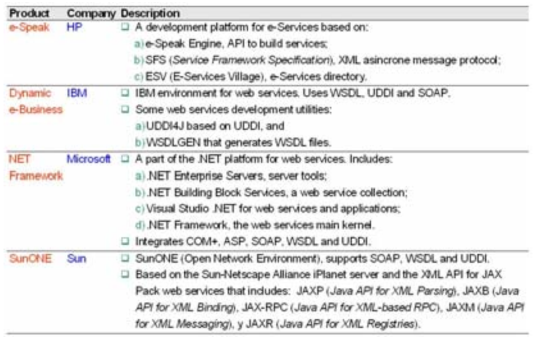

SOAP (May)
http://www.w3c.org/TR/soap12
Ariba
Com-merceOne
Compaq Computer
DevelopMentor
Hewlett-Packard
IBM
IONA
Lotus
Microsoft
SAP AG
Userland Software
MAY
UDDI 1.0
IBM
Microsoft
Ariba
MARCH
WSDL (Web Services Description Language)
http://www.w3c.org/TR/wsdl
IBM http://alphaworks.ibm.com/tech/webservicestoolkit
Microsoft http://msdn.microsoft.com/net
Ariba
JUNE
UDDI 2.0
IBM
Microsoft
Ariba
JUNE
UDDI 3.0
http://uddi.org
IBM
Microsoft
Ariba
2000
2001
2002
APRIL
SDL (Services Description Language)
Microsoft, Visual Studio .NET
SCL (Service Contract Language)
XDR (XML Data-Reduced)
Microsoft
NASSL (Network Accessibility Service Specification Language)
IBM

Figure 2: A roadmap through the web services



Product	Company	Description
e-Speak	HP	❑ A development platform for e-Services based on: a) e-Speak Engine, API to build services; b) SFS (*Service Framework Specification*), XML asincrone message protocol; c) ESV (E-Services Village), e-Services directory.
Dynamic e-Business	IBM	❑ IBM environment for web services. Uses WSDL, UDDI and SOAP. ❑ Some web services development utilities: a) UDDI4J based on UDDI, and b) WSDLGEN that generates WSDL files.
NET Framework	Microsoft	❑ A part of the .NET platform for web services. Includes: a) .NET Enterprise Servers, server tools; b) .NET Building Block Services, a web service collection; c) Visual Studio .NET for web services and applications; d) .NET Framework, the web services main kernel. ❑ Integrates COM+, ASP, SOAP, WSDL and UDDI.
SunONE	Sun	❑ SunONE (Open Network Environment), supports SOAP, WSDL and UDDI. ❑ Based on the Sun-Netscape Alliance iPlanet server and the XML API for JAX Pack web services that includes: JAXP (*Java API for XML Parsing*), JAXB (*Java API for XML Binding*), JAX-RPC (*Java API for XML-based RPC*), JAXM (*Java API for XML Messaging*), y JAXR (*Java API for XML Registries*).

Figure 3: Platforms and tools to work on web services environments

Fortunately, IBM and Microsoft —together with Ariba— combined their efforts and created a working group to elaborate a standard language, as a common objective for both companies. As a result, the WSDL (*Web Services Definition Language*) is generated: the first language for web services definition. In March of 2001, WSDL was accepted as a recommendation by the W3C. The WSDL specification can be found in `http://www.w3.org/TR/wsdl`. Finally, this language uses the W3C base-data types definition. IBM offers WSDL support at `http://alphaworks.ibm.com/tech/webservicestoolkit`. Microsoft offers support to WSDL at `http://msdn.microsoft.com/xml` and at `http://msdn.microsoft.com/net`. As a summary, Fig. 3 shows some of the more important platforms and tools used to build web services environments.

Fig. 4 shows some of the forthcoming extensions of the web services on security, QoS or transactions expecting to appear soon in the literature. However, this section will only focus on the WSDL description.

A WSDL definition allows any web client to call the web service by means of programming, without knowing anythig about the implementation details of the web service or what platform or operating system it is working.

A WSDL document is composed of seven XML elements, although not all of them are obligatory to define a web service. These elements are: (1) the types (`<types>`), which are defined with a W3C's schema (though other accessible private types can also be used

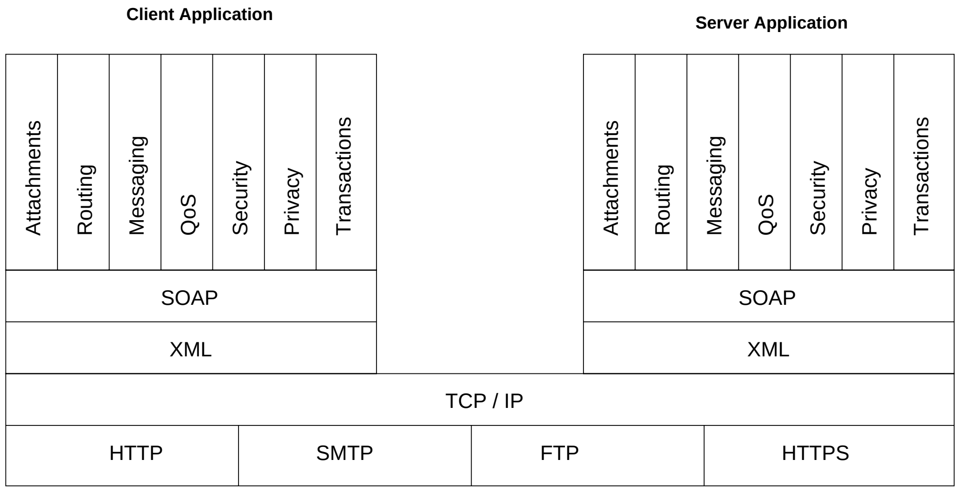


Figure 4: Forthcoming extensions of the web services

from a name space); (2) the input and output messages (`<message>`); (3) the types of port (`<portType>`), where the service interfaces are defined; (4) the interface operations (`<operation>`); (5) the bindings (`<binding>`); (6) the ports (`<port>`); and (7) the service (`<service>`). All these elements (and some more) are defined in the WSDL schema `http://schemas.xmlsoap.org/wsdl/`.

Fig. 5 shows a web service definition in the WSDL language. In this example, the web service accepts an identifier (ID) to look for the user information in its associated database. As a reply, the web service returns the name and the first name of the identified user.

The content of a WSDL document is represented in a structural way. Firstly, the web service data types are defined (the input and output types). After that, the web service messages and ports types are described. Then, the web service connection point is established. Finally, the web service with its ports and an Internet access way is established.

A web service is delimited by the `definitions` element and a name. Fig. 5 shows the WSDL of the web service `identificationService`. The `xlmns` and `targetNamespace` namespaces establish the Internet place of the web service (`targetNamespace="http://acme.solutions.es/identificationService.wsdl"`) and the Internet place of the SOAP and WSDL document schemas. By default, these two namespaces reside respectively in `http://schemas.xmlsoap.org/wsdl/soap/` and `http://schemas.xmlsoap.org/wsdl/`.

The data types used by the service operations (only one, the operation `identification`) are defined inside the XML `<types>` section (lines 6 to 21). Data types are defined in a W3C schema document (`<schema>` section, lines 7 to 20). Types can be declared either directly in the document (`<types>` section) or in a linked external document, for instance ⟨`import location="http://acme.solutions.es/typesIdentification.xsd"`⟩.

Firstly, the service messages are defined in two `<message>` sections (lines 22 and 25). These messages are then used by the `Identification` operation (lines 30 and 31) in a port type: `portTypeIdentification` (line 28). Port types are the component interfaces. Therefore, if the service had more than one interface, we would have to define as many port types as existing interfaces. The port type shown in the example (`portTypeIdentification`) only contains a single operation, but in case there was more than one operation, we would have to declare as many `<operation>` sections as operations the port type (the interface) had.

```
<?xml version="1.0"?>
<wsdl:definitions name="identificationService"
      targetNamespace="http://acme.solutions.es/identificationService.wsdl"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:types>
      <xsd:schema targetNamespace="http://acme.solutions.es/identification.xsd"
                 xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema">
         <xsd:element name="identificationRequest">
            <xsd:complexType>
               <xsd:element name="ID" type="xsd:float"/>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="identificationResponse">
            <xsd:complexType>
               <xsd:element name="Name" type="xsd:string"/>
               <xsd:element name="Firstname" type="xsd:string"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="inputData">
      <wsdl:part name="body" element="identificationRequest"/>
   </wsdl:message>
   <wsdl:message name="outputData">
      <part name="body" element="identificationResponse"/>
   </wsdl:message>
   <wsdl:portType name="portTypeIdentification">
      <wsdl:operation name="identification">
         <wsdl:input message="inputData"/>
         <wsdl:output message="outputData"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="bindingIdentification" type="portTypeIdentification">
      <wsdl:operation name="Identification">
         <soap:operation soapAction="http://acme.solutions.es/identification"/>
         <wsdl:input> <soap:Body use="literal"/> </wsdl:input>
         <wsdl:output> <soap:Body use="literal"/> </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="identificationService">
      <wsdl:documentation>An identification service</wsdl:documentation>
      <wsdl:port name="identification" binding="bindingIdentification">
         <soap:address location="http://acme.solutions.es/servlet/control.Id"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Figure 5: A WSDL web service definition example

```
<wsdl:binding name="bindingIdentificacion" type="portTypeIdentification">
  <wsdl:operation name="identification">
    <soap:operation soapAction="http://acme.solutions.es/identification">
    <wsdl:input> <soap:Body use="literal"/> </wsdl:input>
    <wsdl:output> <mime:part>
        <mime:content part="outputData" type="text/html"/> </mime:part>
  </wsdl:operation>
</wsdl:binding>
```

Figure 6: A MIME protocol example for a WSDL document

The `<binding>` section (lines 34 to 40) is related to the service protocols. Protocols are related to the way in which the messages are sent in a SOAP, request and reply[1]. The input and output structure of these messages are established in the `<soap:Body>` section. Here, the `use` attribute takes the values `"literal"` or `"encoded"`. Generally, the first one is used to indicate that the messages (`<input>`, `<output>`) respect "literally" its definition, established in the `<message>` sections. If the value is `"encoded"`, it refers to the transportation protocol to send or receive the messages (HTTP, SMTP, FTP). For example, Fig. 6 rewrites the `<binding>` section for the "Identification service", but now the outputs (`<output>` sections) are supposed to be sent in a HTML page. A protocol MIME is used in the example specifying the "mime" web page type (i.e., `text/html`).

The WSDL document concludes with the web service description (lines 41 to 46). A service is defined by a name, one or more ports, and its location in the network where it can be called. In the example, the service contains a single port called `identification` (line 43) and it is activated by a call to a "servlet" at `http://acme.solutions.es/servlet/control.Identification`.

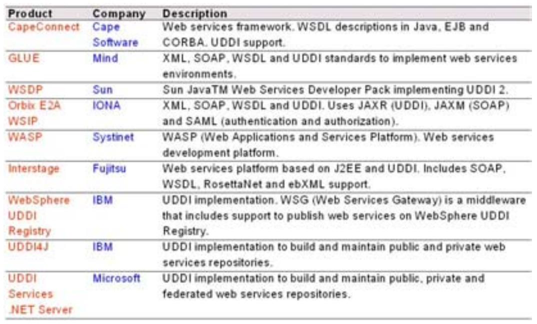
For further information about the WSDL specification, you can go through the W3C's web page (`http://www.w3c.org/TR/wsdl`) or in [12] and [22], among others.

4 Publising and Inquiring Services

Similar to the web service description language (the WSDL language), IBM and Microsoft (together with Ariba) also created another working group to develop a specification function for the publication and location of web services, UDDI (*Universal Description, Discovery and Integration*). In September, 2000, this group created the first UDDI specification, and nine months later, in June, 2001, created the version 2.0. Currently there exists the version 3.0, published at the end of June, 2002. The working group is maintained by the OASIS organization at `http://www.uddi.org`, in which more than 200 organizations collaborate.

UDDI specification implementations quickly appeared in the market (see Fig. 7), but it has not been so quick the appearance of one or more important known web places which have the necessary infrastructure to support UDDI-based web services repositories. Other companies, as SalCentral (`http://www.salcentral.com`), BindingPoint (`http://www.bindingpoint.com`) or XMethods (`http://www.xmethods.com`), took advantage of this gap creating important Internet-based web services repositories. However, these repositories do not respect the UDDI model, and they are simply limited to lodging WSDL documents. Nowadays, these web service repositories are a very important reference point

[1]It is necessary to clarify that the concept of protocol here does not coincide with the concept of protocol in the components software arena. In this case, we are referring to the data transportation protocol. In software components, a protocol refers to the order in which the messages are sent and called by other components



Product	Company	Description
CapeConnect	Cape Software	Web services framework. WSDL descriptions in Java, EJB and CORBA. UDDI support.
GLUE	Mind	XML, SOAP, WSDL and UDDI standards to implement web services environments.
WSDP	Sun	Sun JavaTM Web Services Developer Pack implementing UDDI 2.
Orbix E2A WSIP	IONA	XML, SOAP, WSDL and UDDI. Uses JAXR (UDDI), JAXM (SOAP) and SAML (authentication and authorization).
WASP	Systinet	WASP (Web Applications and Services Platform). Web services development platform.
Interstage	Fujitsu	Web services platform based on J2EE and UDDI. Includes SOAP, WSDL, RosettaNet and ebXML support.
WebSphere UDDI Registry	IBM	UDDI implementation. WSG (Web Services Gateway) is a middleware that includes support to publish web services on WebSphere UDDI Registry.
UDDI4J	IBM	UDDI implementation to build and maintain public and private web services repositories.
UDDI Services .NET Server	Microsoft	UDDI implementation to build and maintain public, private and federated web services repositories.

Figure 7: Some UDDI implementations

Operator	Publish	Query
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`
Some testing UDDI operators		
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`

Table 1: Some UDDI Internet entry points

for developers of web component-based applications. Perhaps the most important one is the SalCentral's repositoy (`http://www.salcentral.com`), which lodges an interesting collection of web services developed by important companies.

At the end of the year 2002, we witnessed the consolidation of earlier web service repositories implementations based on the UDDI model. The infrastructure that supports and maintains a web service UDDI repository is named UBR (*UDDI Business Registry*). The most important UBR repositories are exactly those maintained by the pioneering companies in the UDDI model: IBM and Microsoft. Those companies that maintain and offer an Internet entry point to UBR repositories are called "UDDI operator node" (UON). Table 1 shows some important UON.

A UBR can be commonly maintained by one or more operators, each one supporting its own repository. The UDDI model supports duplication of services to affiliated repositories to maintain the consistency of the UBR. Therefore, a service published in the operator A, can be searched later in the operator B, since its associated repository is actualized with an automatic duplicate of that service published in the operator A. The duplicates are not automatic, and the minimum updating time is 12 hours. As an example, IBM and Microsoft maintain together an UBR.

UDDI is based on a "suscribe/publish/inquire" model to register and locate objects. For web services publication, the UDDI model requires that the service supplier has to be subscribed previously in the operator. It is not necessary to be subscribed for searches. Table 1 contains two columns: the Internet entry point for publishing web services, and that for searching web services. Besides, the UDDI publication entry points require a secure connection `https` with an identification for the supplier. In the quering entry points, searches can be directly done in the repository.

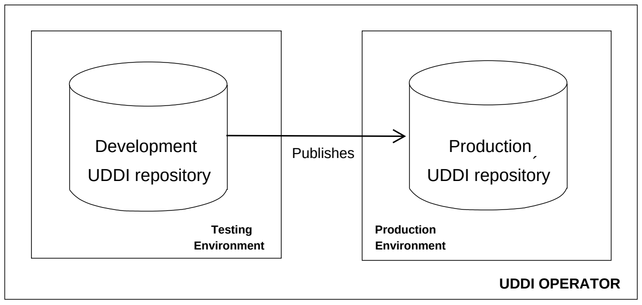


Figure 8: A development and production environment for a UDDI operator

Last rows in Table 1 show some entry points to UDDI operators repositories for testing activities, called "development UDDI repositories". This kind of repository helps the web services developers in the construction and validation activities. Developers can register their web service prototypes in the test repositories to validate their services, always in a private way. When these services have been sufficiently tested and validated, the developer publishes them in another repository called "production UDDI repository". The services published in that repository can be used by other developers or web services clients.

The UDDI repository's information model basically stores information of those organizations that publish web services in the repository. The WSDL documents are not stored in the UDDI repository; they remain in a web place pointed by the published service. Conceptually, a supplier company can publish or query services following some models:

a) *White pages model.* It contains basic contact information, for example the company's name, postal and e-mail addresses, personal contacts, telephone numbers, and a unique identifier. Identifiers are alphabetic or numerical values that distinguish supplier companies. The identifier is assigned to the supplier business when it is subscribed to the UDDI operator.

b) *Yellow pages model.* It gathers the companies registered by categories, using the assigned identifier and the offered service types. This option uses a web service catalogue when publishing and looking for services. This model is very useful for selective web services searches into one or more selected categories, reducing the response times.

c) *Green pages model.* It contains the services' technical information for supplier companies. This information is very useful for service clients to know the web service connection and communication programming details. Therefore, this information defines how to invoke the service (as we saw in the previous section). The green pages normally include references to the WSDL documents, which contain information about how to interact with the web service.

As it has already been advanced before, the UDDI specification requires that web services supplier should be subscribed in that operator where it desires agreed. At the first time, UDDI subscribes the supplier requesting its e-mail address and a password. Internally, UDDI generates a unique identifier based on some key generation model, and creates a catalogue (a space) for the supplier, with rights to modify, register and withdraw its web services. This supplier company catalogue is a `businessEntity` document in which the supplier will be able to include new web services, previously establishing a secure connection (`https`), Table 1.

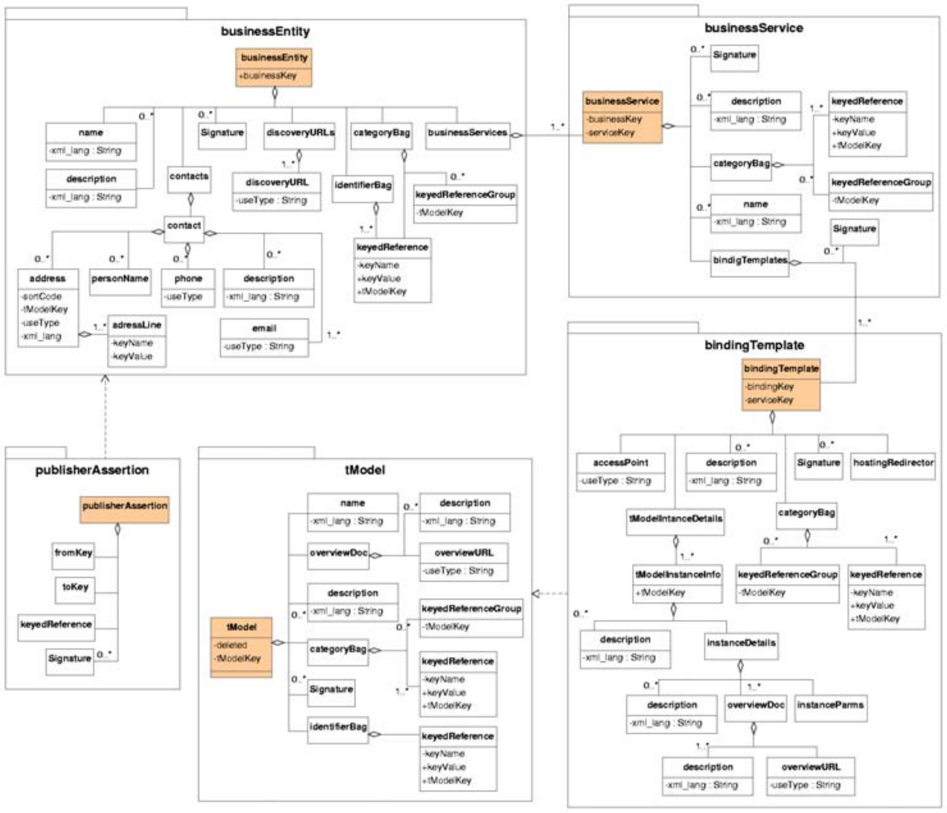


Figure 9: Modeling a UDDI register

```
<businessEntity businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
  <name>Acme solutions S.A.</name>
  <description xml:lang="en">An electronic supplier company</description>
  <contacts>
    <contact>
      <personName>Technical Director</personName>
      <phone>+34-950-123456</phone>
      <email>manager@acmesol.es</email>
      <address> <addressLine>Ctra. Sacramento s/n 04120</addressLine> </address>
    </contact>
  </contacts>
  <businessServices>
    <businessService businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
                     serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
      <name>Identification service</name>
      <description xml:lang="en">An Internet identification service</description>
      <categoryBag>
        <keyedReference keyName="NAICS: e-commerce" keyValue="..." tModelKey="... "/>
        <keyedReference keyName="NAICS: Database soft." keyValue="..." tModelKey="... "/>
        ...
      </categoryBag>
      <bindingTemplates>
        <bindingTemplate bindingKey="6D8F8DF0-3AAF-11D5-80DC-002035229C64"
                         serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
          <accessPoint>http://acme.solutions.es/servlet/control.Identification</accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="564B5113-1BAE-21A3-906E-122035229C75"/>
          </tModelInstanceDetails>
        </bindingTemplate>
        ...
      </bindingTemplates>
    </businessService>
    ...
  </businessServices>
</businessEntity>
```

Figure 10: The UDDI register in XML

UDDI is basically composed of five types of information: (a) `businessEntity`, information about the company; (b) `businessService`, information about the services that the company offers; (c) `bindingTemplate`, technical information for a service; (d) `tModel`, information about how to interact with the web service; (e) `publisherAssertion`, information about the relation of the company with other ones. A UDDI register is an XML document that includes these five types of information. Fig. 9 shows a meta-model of information created from the UDDI 3.0 data structure specification [7]. Section `businessEntity` encapsulates the types, except the `publisherAssertion` section and the WSDL web services definitions. When the supplier company is subscribed in the UDDI operator, a `businessEntity` document is created for it, including certain business information (such as its name or contact address).

When the supplier company registers the first web service in the UDDI repository, a `businessServices` section is created in the `businessEntity`, and a `businessService` section will also be created for every published web service.

When the web service is stored in the UDDI repository, the service information is structured in the `bindingTemplate` technical sections and in a `tModel` XML external document. A `tModel` document contains a link to the WSDL definition of the web service, which describes how the connections with the service operations are done. A `tModel` document is a service type that can be reused by the supplier company to publish other web services.

Fig. 10 shows an XML UDDI document, which will be useful to analyze better this meta-model and some concepts of the UDDI information model. The figure shows an XML example that corresponds to a meta-model instance in Fig. 9. To analyse the UDDI information model, we will jointly make reference to these two figures (9 and 10).

The diagram shows the `businessEntity` parts for an XML document. Each UML class models a label of the XML document. The class attributes section has been used to model the label attributes. The "`-`" symbol means an optional attribute, and the "`+`" symbol means a mandatory attribute. For example, the `businessEntity` class has a required attribute called `businessKey`, which represents the unique identifier of the supplier company assigned when it made the subscription in the UDDI operator. This class models the `businessEntity` label for a UDDI document (line 1, Fig. 10).

The `businessEntity` document contains: the name of the supplier company (line 2), a company's description (line 3) and contact data (lines 4 to 11). This information corresponds with that on the "white pages".

Web services are registered in the `businessServices` section (from lines 12 to 34). Inside this, a `businessService` section is used to define each web service published by the supplier company (lines 13 to 32). Two attributes are used for each service: one represents the unique identifier of the supplier company, and the other one represents the unique identifier of the published web service. For the latter, the identifier assignment for the published services is similar to the identifier assignment for the supplier company (the `businessKey` identifier).

The services in the `businessService` section have got: a name (line 15), a service description (line 16) and a classification information (from lines 17 to 21). This information corresponds with that on the "yellow pages".

Moreover, a `businessService` service section can contain technical information, collected in the `bindingTemplates` section (from lines 22 to 31). This information corresponds with the "green pages" information. The `bindingTemplates` section includes connection information (line 25) and a link to the XML document `tModel` service type stored in the UDDI operator's repository (26-28). The `tModelInstanceInfo` label contains an attribute with the identifier of the `tModel` service type.

There is a link to the WSDL definition inside a `tModel` document: a "`.wsdl`" file. It contains the description about how the service operations should be invoked, and how these return the data.

UDDI supports functional interfaces to interact with several client objects: (a) the object that publishes web services (*publisher*), and (b) the object that looks for services (*inquiry*). The requests and replies of the UDDI operations are achieved by using XML documents. Table 2 shows the message types for the `Publisher` and `Inquiry` interfaces operations.

The publisher interface's operations use some XML document messages to store, modify, erase or eliminate white, yellow or green information pages. The inquiry interface's operations also use XML documents to interrogate on those three pages types. A `<find_xxx>` document extracts a collection of documents that fulfil the restrictions of the inquiry document. The returned collection is encapsulated in a global label, shown in the fourth column in Table 2. The `<get_xxx>` documents extract some detailed information.

Table 3 shows some query examples to look for services information, together with the results that these generate. Searches can be refined by applying new query documents on those returned documents and limiting increasingly the space of the search. In the example, the search starts with all the services published by the supplier company "Acme solutions S.A.". Then, on the returned result, it looks for a service called "Identification service" published by that supplier. UDDI returns just one reference to the service. Finally, UDDI obtains a more detailed description of the web service in the last query.

Publisher		Inquiry	
Request	Reply	Request	Reply
<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>		

Table 2: Publication and locating messages for the UDDI repository

Due to the extension of the UDDI 3.0 specification report (only the data structures section occupies more than 300 pages) just the most significant document labels have been described in this paper. A fairly detailed description for the UDDI information model can be found in [7, 9, 10].

5 WSDL and UDDI Shortcomings

The shortcomings have been detected based on the experience obtained after analyzing the UDDI 3.0 specification [7] and some UDDI industrial implementations, such as the IBM UDDI4J API (`http://www-124.ibm.com/developerworks/oss/uddi4j`), and the *UBR* of Microsoft (`http://uddi.ibm.com`), IBM (`http://uddi.ibm.com`) and HP (`http://uddi.hp.com`). There are some other non-UDDI web services repositories which have been analyzed too, such as the SalCentral (`http://www.salcentral.com`) and the XMethods (`http://www.xmethods.com`) repositories.

WSDL is a very useful language for the specification of simple components working across the Internet; these components are called web services. These services have generally been encapsulated by web pages from which the "developer" figure appears by Internet-based applications. This kind of software developer needs to know certain implementation details about the web service, usually described by means of its interfaces: for example the way in which the operations are called, or the way in which the data or the service's use restrictions are returned, among other interface implementation details.

In the last few years, WSDL has been established as a standard XML-based language for web services specification. As Section 3 describes, a WSDL document is composed of seven XML elements to define service interface (or interfaces) details, i.e., the input and output data type of the operation signatures, or the way in which the operations are called and returned, and connection details. Nevertheless, the WSDL specification does not take into account the following aspects:

(a) *Protocols.* WSDL uses the protocol term to refer to the transfer protocol (HTTP, FTP, SMTP). Nevertheless, the WSDL XML schema does not deal with a notation to refer to the protocol concept (or choreography) in the sense of knowing the order in which the input and output operations are called. Therefore, the language should enable

Query 1: searches all the services for a supplier company

```
<find_business>
  <name>Acme solutions S.A.</name>
</find_business>
```

Return 1: the service list

```
<businessList>
  <businessInfos>
    <businessInfo businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
      <name>Acme solutions S.A.</name>
      <description xml:lang="en">An electronic supplier bussiness</description>
      <serviceInfos>
        <serviceInfo businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
                     serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
          <name>Identification service</name>
        </serviceInfo>
        ...
      </serviceInfos>
    </businessInfo>
  <businessInfos>
</businessList>
```

Query 2: now it searches a service in the service list

```
<find_service businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
  <name>Identification service</name>
</find_service>
```

Return 2: a reference to the found service

```
<serviceList>
  <serviceInfos>
    <serviceInfo businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
                 serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
      <name>Identification service</name>
    </serviceInfo>
  </serviceInfos>
</serviceList>
```

Query 3: it searches more detailed information of the service

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

Return 3: the information of the service

```
<serviceDetail>
  <businessService businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
    Lines 13 to 32 in Fig. 10
  </businessService>
</serviceDetail>
```

Table 3: Several query examples in the UDDI notation

to include directly (in XML) or indirectly (an external link) a notation to describe protocols; for example, a protocol described in an SDL or π-calculus notation.

(b) *Behavior.* It is necessary that the language also enables a semantic behavior description for the operations (pre/post conditions), and not only a syntactical definition for the signatures. Therefore, the language should enable again to include directly (in XML) or indirectly (an external link) semantics notations; for example, a definition in Larch.

(c) *Non-functional information.* Finally, it is also necessary that the language can use a notation to define non-functional properties, apart from the functional information (syntax, semantics and protocols of the interfaces) and the technical information of the service. Again, it should enable to include directly or indirectly this kind of information; for example, the properties described in the ODP way: name, type, value.

The WSDL shortcomings also result in the UDDI model, and basically affect to the searching operations in the repository. Therefore, as a first UDDI limitation, the query operations are only held to technical aspects of a web services supplier company, to technical aspects of their web services, and also to aspects of location, connection and communication of web service operations. Nevertheless, these operations do not allow searches for: (a) the interaction protocols of the operations, (b) the behavior of the operations; and (c) the non-functional information.

Secondly, although the UDDI specification enables the affiliation (`publisherAssertion`), it does not really take into account the federation of UDDI repositories in the same way as the ODP trading service standard [21]. The affiliation of UDDI operators enables to maintain the consistency of the information for a single virtual UBR (*UDDI Business Registry*) repository. Everytime a service is published in the UDDI operator's repository, it is duplicated to those affiliated repositories. Nevertheless, a subsidiary operator can be associated to more than one UBR, but this can lead to certain problems.

For example, let us suppose the existence of two UDDI business registry (UBR): *UBR1* and *UBR2*. Let us also suppose that operators A and B are affiliated to *UBR1*, and operators B and C to *UBR2*. Operator B is affiliated to both UBRs. According to this, a service publication in *UBR1* does not imply that the service is also published in *UBR2*, by means of a duplicate (and vice versa); even if operator B is as a subsidiary in both UBRs.

Apart from this, the propagation of queries between UBR operators is not possible either (the query is achieved on the operator, not on the UBR). Only a duplicate of the published service is propagated toward its affiliated operators. Carrying on with the example, let us now suppose that services publication is more frequent on operator *UBR1* than operator *UBR2*. Let us also suppose that a web services client X can inquire indistinctly on operator A or B, even if the client does not know that these two operators are affiliated, and (in principle) they should have the same registered services. Nevertheless, queries achieved by another client Y on operator C are only limited to the repository of that operator, and they are not propagated to operator B, which is supposed to have more information.

In this sense, UDDI is a useful and complete directory service, but not a trading service, such as it could be wished on the UDDI model. A trading service can be considered as an advanced directory service that permits searches based on attributes, i.e., attributes of quality [23].

6 Conclusions and Future Work

Similar to the earlier software components definitions, described as a unique interface, currently the web services descriptions encapsulate the functionality of an individual object that actives itself and responds via web. Nevertheless, the component models recently support the definition of multiple provided and required interfaces for a software component. On the other hand, the component community seems to know what kind of information is needed in order to describe a software component (syntactic, semantics, protocols and non-functional information) to develop systems by means of the composition of multiple components. The web services should also take into account this kind of information which is useful for the engineers to build their systems from web services collections. Moreover, they are supposed to be able to interoperate and/or to be developed by third parts.

However, in contrast to the software components where the way of collecting this component information does not seem to be extended, in the web services arena there exists an extended XML-based language to describe web services, such as the language WSDL. Nevertheless, as we previously studied in this work, it does not seem very appropriate to use either the current web services description language (WSDL) or the directory service specification to describe, publish and locate web services (UDDI) in traditional component-based development processes, that is, by means of the composition of several software parts. In a description perspective, a possible solution could be to extend the WSDL schemas including those XML tags in order to approach software components (i.e., syntactic, semantic, protocol and non-functional descriptions), similar to the COTScomponent templates [19].

Although this paper does not offer a solution in this sense, it analyses a study of the WSDL and UDDI techniques for web services-based development, and it identifies a problem in the conception of these techniques for the building of complex systems from multiple web services.

As a future work, the current WSDL and UDDI notation should be extended so that they can really be used in CBD processes. It would also be interesting to observe the federation between those WSDL repositories that respect the UDDI specification and those that do not, similar to the Salcentral repositories implementing privates WSDL repositories. Finally, since a web services market is likely to appear soon, it would also be interesting to approach web services and COTS components, considering the web services in the commercial components trading services, such as the COTStrader service [19].

Acknowledgments

This work has been partially supported by Spanish CICYT Projects TIC2002-04309-C02-02 and TIC2002-03968. I would like to thank Antonio Vallecillo, from the University of Málaga (Spain).

References

[1] F. Acherman, M. Lumpe, J. Schneider, and O. Nierstrasz. Piccola—A Small Composition Language, 1999. University of Berne. `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. In *ESCOM (European Software Control and Metrics conference)*, pages 337–346, April 2001.

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

[5] L. Bass, P.C. 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. Clement, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. Riegen. UDDI Version 3.0, July 2002. `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. `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. `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, 2003.

[12] P. Cauldwell, R. Chawla, V. Chopra, G. Damschen, C. Dix, T. Hong, F. Norton, 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.A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers, 1999. ISBN: 07-923-86663.

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

[15] 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.

[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, 1995.

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

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

[19] L. Iribarne, J. M. Troya, and A. Vallecillo. A Trading Service for COTS Components. *The Computer Journal*, 47(3), may 2004.

[20] ISO/IEC-9126. Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use, 1991. International Standard ISO/IEC 9126.

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

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

[23] L. Kutvonen. Achieving Interoperability through ODP Trading function. In *Second International Symposium on Autonomous Decentralized systems (ISADS'95)*, pages 63–69, Arizona, April 1995. IEEE Computer Society.

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

[25] 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.

[26] G. T. Leavens, L. Baker, and C. Ruby. *Behavioral Specifications of Businesses and Systems*, chapter JML: A Notation for Detail Desing. Kluwer Academic, 1999.

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

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

[29] A. Mikhajlova. *Ensuring Correctness of Object and Component Systems*. PhD thesis, October 1999.

[30] R.A. Riemenschneider and V. Stavridou. The Role of Architecture Description Languages in Component-Based Development: The SRI Perspective. In *21st International Conference on Software Engineering*, May 1999. `http://www.sei.cmu.edu/cbs/icse99/papers/42/42.htm`.

[31] C. Thompson. Workshop on Compositional Software Architectures: Workshop Report. January 1998. Monterey, Clifornia. `http://www.objs.com/workshops/ws9801/report.html`.

[32] 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.

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

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

[35] 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.

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