

Building a Web service from SOAP to nuts

Doug Tidwell

IBM developerWorks

`dtidwell@us.ibm.com`

0

Agenda

- Creating the service
- Deploying it with Apache Axis
- Writing simple Java clients using Axis
- More advanced SOAP features
- Writing clients in other languages
- Writing Axis handlers
- Enabling service discovery

ibm.com/developerWorks/webservices

0

Key Messages

- **Web services are a new style of computing.**
 - They will change software development as significantly as the Web itself.
 - If you ignore them, your competitors will thank you, possibly over drinks after they acquire what's left of your company.
- **Web services are real...**
 - ...but not finished yet. We'll make it clear what's **really** real and what isn't.

ibm.com/developerWorks/webservices

0

Web services

- “We believe that applications will be based on compositions of services discovered and marshaled dynamically at runtime.”
 - Former U.S. Vice-President Al Gore, in a speech to the Democratic National Convention in 1984

ibm.com/developerWorks/webservices

0

What are Web services?

- Some kind of function you can use across some kind of network
- Revolve around short interactions (connect, access code, disconnect)
- Accessible through service brokers, maybe (UDDI? WS-Inspection?)
- Described with XML

ibm.com/developerWorks/webservices

0

Behind the technology

- Everything we'll discuss here is based on XML at some point:
 - SOAP
 - WSDL
 - UDDI
- In addition, many Web services will be delivering XML-tagged content; that content will most likely be transformed with XSLT.

ibm.com/developerWorks/webservices

0

The big idea

- If you've got some code that does something useful (say, a JavaBean, maybe), you can publish that service to a registry.
- If you need some useful function, you can describe what you're looking for and see what the broker finds.
- **All of this can be automated.**

ibm.com/developerWorks/webservices

SOAP overview

Rub-a-dub-dub-dubya

0

SOAP

- Originally the Simple Object Access Protocol, now it's just **SOAP**.
- Originally developed by Microsoft, UserLand, DevelopMentor, etc.
- SOAP V1.1 was submitted by IBM, Microsoft, UserLand, DevelopMentor, and Lotus.

ibm.com/developerWorks/webservices

0

SOAP

- Currently SOAP 1.2 is being developed by the XML Protocol working group at the W3C.
- `xml.apache.org/axis` is the home of the Axis project, the ASF's most complete SOAP implementation.

ibm.com/developerWorks/webservices

0

SOAP in a nutshell

- An XML-based protocol with:
 1. An envelope structure that contains a message
 2. Encoding rules for data types
 3. Rules for RPC requests and responses
 4. Rules for exchanging messages using an underlying protocol

ibm.com/developerWorks/webservices

0

SOAP design

- Simple
- Vendor-neutral
- Language-neutral
- Object-model-neutral
- Transport-neutral

<xml>

ibm.com/developerWorks/webservices

0

SOAP message structure

- Most SOAP applications use Remote Procedure Call (RPC) style SOAP messages.
- There are **request** and **response** messages:
 - A request invokes a method on a remote object
 - A response returns the result of running the method

ibm.com/developerWorks/webservices

0

Envelopes

- A SOAP **envelope** contains the message itself.
- The message is in an application-specific vocabulary; namespaces are used to distinguish the parts.
- The envelope can contain a **header**, and it must contain a **body**.

0

A SOAP request

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="..."
  SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="my-ns">
      <symbol>IBM</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

0

A SOAP response

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="..."
  SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse
      xmlns:m="my-ns">
      <price>131.25</price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


0

Application view

- If your application is using SOAP to invoke a service, you need to build the request and parse the response.
- You can use XSLT, SAX, DOM, or JDOM to convert the XML data into anything else.
- Much of the XML parsing has disappeared, thanks to various SOAP toolkits.

ibm.com/developerWorks/webservices

The Magic Eight Ball

A small Web service

-or-

American Garbage Culture 101

0

The Magic Eight Ball



- “50,000 years of technological advancement has culminated in a system to bring you mysticism on demand.”
- Feel free to shout your questions...

ibm.com/developerWorks/webservices

0

The service code

- `EightBall.java` consists of an array of 20 phrases and a random number generator.
- Nothing in the code is XML-aware, SOAP-aware, network-aware, etc.
- We can still take this code and deploy it as a SOAP service.

ibm.com/developerWorks/webservices

0

Deploying the service

- The simplest way to deploy the service is to use Axis's Java Web service (**.jws**) file support:
 - Copy your ***.java** file to ***.jws**
 - Copy the ***.jws** file to the **axis** directory
- **That's it!** Axis automatically finds your source file, compiles it, and deploys it.

ibm.com/developerWorks/webservices

0

Deploying the service

- **We didn't change any of our original code** to deplore it as a SOAP service.
 - We also didn't have to think about SOAP when we wrote or designed the code.
- It's likely you can take the useful applications you already have and deploy them as SOAP services as well.

ibm.com/developerWorks/webservices

0

Accessing the service

- To access the SOAP service, you need several things:
 - The URL of the SOAP router
 - The ID of the service
 - The name of the method
 - The parameters for the method (if any)
 - The value for **SOAPAction**
- This is similar to the information we needed to deploy the service.

ibm.com/developerWorks/webservices

0

The client code

- First, we create an Axis Call object:

```
Service service = new Service();  
Call call = (Call)  
    service.createCall();
```

ibm.com/developerWorks/webservices

0

The client code

- We set the URL of the service (the **endpoint**, in Axis terms):

```
String
```

```
    endpt="http://localhost:8080...";
```

```
call.setTargetEndpointAddress  
    (new URL(endpt));
```

0

The client code

- We set the method name (the **operation name**, in Axis terms):
`call.setOperationName`
`("getAnswer") ;`

0

The client code

- We set the ID of the service (the **namespace**, in Axis terms):
`call.setProperty(Call.NAMESPACE,
"urn:Services:EightBall");`

ibm.com/developerWorks/webservices

0

The client code

- Now we're ready to invoke the service, so we'll use the `Call.invoke` method:

```
call.invoke(new Object[] { } );
```
- That's it!

ibm.com/developerWorks/webservices

0

The client code

- The Axis toolkit does the tedious work for us:
 - Building the SOAP request
 - Sending the SOAP request to the correct URL
 - Parsing the SOAP response

ibm.com/developerWorks/webservices

0

Passing parameters

- You can pass any parameters to the `Call` object.
- This adds a parameter named `question` of type `String` (from the XML Schema datatypes spec):

```
call.addParameter("question",  
    XMLType.XSD_STRING,  
    Call.PARAM_MODE_IN);
```

ibm.com/developerWorks/webservices

0

Passing parameters

- To pass the parameters, simply put the objects inside the array when you use the `invoke` method:

```
call.invoke(new Object[]  
    {"Will IBM stock ever be worth  
    $100 again?"});
```

- We'll talk about passing things other than simple datatypes in a minute...

0

Passing parameters

- If you just pass some parameters on the **invoke** method, there must be a method with that signature in the code that provides the service.
- If you don't use **Call.addParameter**, the parameter names will be **<arg0>**, **<arg1>**, etc.

ibm.com/developerWorks/webservices

0

Passing parameters

- If the `invoke` method doesn't have the same number and type of parameters as the `addParameter` calls, Axis throws an exception.

0

Magic Eight Ball resources

- <http://ofb.net/8ball/>
 - A developer site, really, full of 8-ball arcana, the official phrases, how to take an 8-ball apart, etc.
- <http://8ball.federated.com/>
 - My personal favorite, a Linux box, a Magic 8-ball, a Web cam, and a Lego MindStorms robot.

ibm.com/developerWorks/webservices

The Axis project

The Axis of Goodness

0

The Axis project

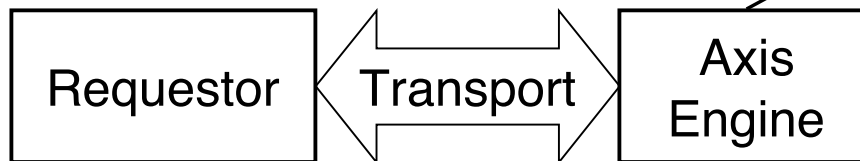
- Based at `xml.apache.org/axis`, the Axis toolkit has several important goals:
 - Better performance (use SAX, not DOM)
 - More flexible architecture
 - Easier to use other transports
 - Make it easy to include other code in message handling (encryption, logging, authentication, etc.)

ibm.com/developerWorks/webservices

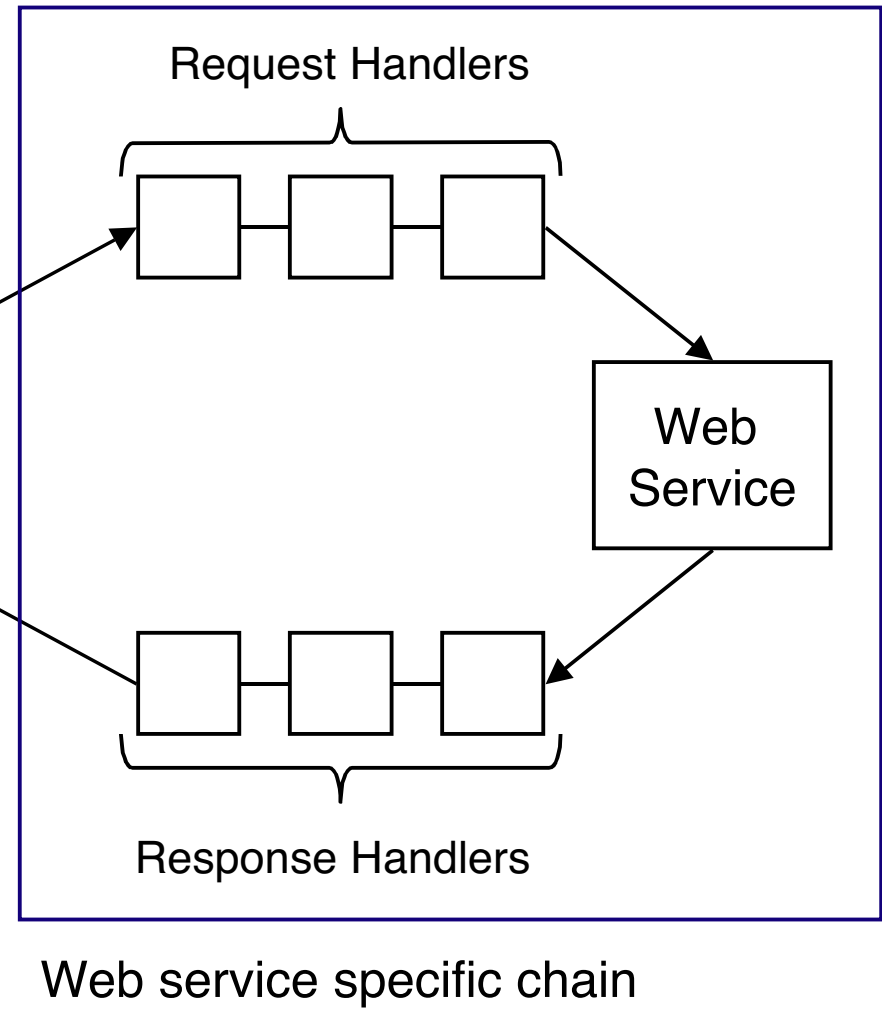
0

Axis architecture

Taken from *Building Web Services with Java*, by Steve Graham, Doug Davis, et al. (more on this later)



Note: Axis also has two optional chains of request handlers: one is global to all requests, and the other is specific to a particular transport. There are two optional chains of response handlers as well.



ibm.com/developerWorks/webservices

0

Axis components

- Axis is made of several components, all of which are interchangeable:
 - The Axis engine itself
 - Handlers (for logging, authentication, etc.)
 - Chains (a series of handlers)
 - Transports (HTTP, FTP, JMS, etc.)

ibm.com/developerWorks/webservices

0

The engine

- The Axis engine is the entry point to the message processing system.
- For flexibility, it can be replaced, although most of your applications will use the **AxisEngine** that's built in to the toolkit.

ibm.com/developerWorks/webservices

0

Handlers

- A handler's job is to look at a SOAP message and (maybe) modify it.
 - A handler might simply log how many times a given Web service has been invoked
 - A handler might look for encrypted data inside a SOAP message and decrypt it.
- Handlers are the basic building blocks in Axis.

ibm.com/developerWorks/webservices

0

Handlers

- If a handler wants to stop a transaction, it throws an **AxisFault**. Here's how you might process a **userID** parameter that's either missing or blank:

```
if (userID = null || userID = "")  
    throw new AxisFault(...);
```

- This causes Axis to send a SOAP fault back to the client.

ibm.com/developerWorks/webservices

0

Chains

- A chain is simply a sequence of handlers.
 - A chain might be an authentication handler, followed by a logging handler, followed by an unencryption handler.

Deploying the service

- You can also deploy a service with a deployment descriptor, an XML file that describes the properties of this class:

```
<deployment>
  <service name="urn:EightBall"
    provider="Java:RPC">
    <parameter name="className"
      value="EightBall"/>
    <parameter name="methodName"
      value="getAnswer askQuestion"/>
  </service>
</deployment>
```

0

Deploying the service

- Axis provides a utility class, `org.apache.axis.client.AdminClient`, that processes the deployment descriptor.

0

Deploying the service

- You also need to copy your `.class` file to the appropriate directory. When running Axis under Apache Tomcat, copying the `.class` file to the Tomcat `classes` directory will do the trick.

ibm.com/developerWorks/webservices

Deploying the service

- You associate handlers, chains, etc. with a particular Web service in the deployment descriptor. Here's an example:

```
<service name="...">  
  <requestFlow>  
    <handler  
      type="java:org.apache.axis.handlers.  
        SimpleAuthenticationHandler"/>
```

Handling errors

What to do when something goes wrong

0

Debugging SOAP apps

- The Axis toolkit includes a SOAP sniffer.
 - `java`
`org.apache.axis.utils.tcpmon`
`8070 localhost 8080`
 - This tells the sniffer to monitor port 8070, and send all requests to `http://localhost:8080`.
- You can see the SOAP requests and responses as they go.

ibm.com/developerWorks/webservices

0

SOAP faults

- The SOAP spec defines a set of fault codes that indicate where something went wrong.
- In addition to faults, SOAP applications written in Java can throw exceptions.

ibm.com/developerWorks/webservices

0

SOAP faults

- We'll change our SOAP client application so that it throws a fault. The document we get back from the service makes it clear that things didn't go well.

ibm.com/developerWorks/webservices

0



ibm.com/developerWorks/webservices

Advanced SOAP

0

SOAP headers

- A SOAP header is a flexible way of adding features to a SOAP message.
- With Axis, you can set up a handler to process the SOAP header; the handler can take care of authentication, logging, etc., while the SOAP body doesn't change.

ibm.com/developerWorks/webservices

0

A SOAP header

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="..."
  SOAP-ENV:encodingStyle="...">
  <SOAP-ENV:Header>
    <m:authentication
      xmlns:m="my-ns">
      <userid>dtidwell</userid>
      <password>xeroxdemo</password>
    </m:authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>...</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ibm.com/developerWorks/webservices

0

SOAP headers

- There are two attributes that can appear on the elements inside a SOAP header:
 - `mustUnderstand`
 - `actor`
- They are only used in elements that are children of `<SOAP-ENV:Header>`. If they occur on other elements inside the header, they are ignored.

0

mustUnderstand

- The **mustUnderstand** attribute can have a value of 0 or 1. If the value is 1, then whatever processes the header must support that element:

`<authentication`

`SOAP-ENV:mustUnderstand="1"> ...`

- The SOAP service must return a **MustUnderstand** fault if the element isn't supported.

0

actor

- The **actor** attribute specifies a SOAP provider that will process a given element.
- A SOAP message may pass through a number of intermediaries as it goes from the client to the service. The SOAP **actor** attribute lets you specify intermediary that should process the header.

ibm.com/developerWorks/webservices

0

Marshalling

- To this point, we've passed things like numbers and strings in the SOAP envelopes.
- What if we need to send a data structure or an object? That's what marshalling is for.

ibm.com/developerWorks/webservices

0

Serializers

- The Axis toolkit lets you define your own classes for serializing and deserializing objects.
- Take a look at the **BeanSerializer** class for an example. This converts any Java bean into an XML document, or vice versa.
- Axis also has serializers for the data types defined in the XML Schema and SOAP specs, including **base64Binary** for binary data.

ibm.com/developerWorks/webservices

0

Asynchronous SOAP

- Everything we've done to this point has been a remote procedure call.
- SOAP messages can be passed asynchronously...

ibm.com/developerWorks/webservices

0

Document style SOAP

- SOAP also allows you to build **document style** applications.
 - Document style stores an XML document inside the SOAP envelope.
 - The document is much more freeform than the SOAP envelopes we've seen so far.

ibm.com/developerWorks/webservices

0

Document style SOAP

- Useful for:
 - Asynchronous services
 - Huge XML documents
 - Maintaining state/session information
 - Validation – The document can use XML Schema
- See `ibm.com/developerworks/library/ws-docstyle.html`

ibm.com/developerWorks/webservices

0

SOAP over other transports

- Everything we've done to this point was SOAP over HTTP.
- SOAP messages can be passed over any transport. Commonly supported transports are HTTP, FTP, SMTP, Jabber, and IM.

ibm.com/developerWorks/webservices

0

SOAP and encryption

- SOAP messages themselves can be encrypted and/or digitally signed.
- If someone else intercepts the message, they can't make any sense of it.
- If someone else tries to alter the message, the digests won't match.

ibm.com/developerWorks/webservices

0

SOAP and attachments

- Axis supports the proposed SOAP Messages with Attachments spec.
 - If you're using SOAP over HTTP, and you want to transmit binary data, just keep the binary stuff in binary and send a multipart HTTP message.

ibm.com/developerWorks/webservices

Clients in other languages

0

Clients in other languages

- Interoperability is the major value of Web services. To emphasize this, we'll look at SOAP clients and services written in a variety of languages:
 - Java
 - C#
 - Perl
 - Python
 - VisualBasic
 - C++

ibm.com/developerWorks/webservices

0

Java

- Here's some of a Java client, written with the Axis APIs:

```
String endpt =  
    "http://localhost:8070/axis/EightBall.jws";  
Service service = new Service();  
Call call = (Call) service.createCall();  
call.setTargetEndpointAddress  
    (new java.net.URL(endpt));  
call.setOperationName("getAnswer");  
call.setProperty(Call.NAMESPACE,  
    "urn:Services:EightBall");  
String ret = (String) call.invoke  
    (new Object[] { });  
System.out.println("The Eight Ball says: "+ ret);
```

ibm.com/developerWorks/webservices

0

C#

- Written with the .NET Framework:

```
[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://www.EightBall.org/services/getAnswer",  
RequestNamespace="urn:Services:EightBall",  
ResponseNamespace="urn:Services:EightBall",  
Use=System.Web.Services.Description.SoapBindingUse.Encoded, ParameterStyle=  
System.Web.Services.Protocols.SoapParameterStyle.Wrapped) ]  
  
public System.String getAnswer() {  
    object[] args = {"Will I win the lottery?"};  
    object[] results = this.Invoke("getAnswer",  
    args);  
    return ((string) (results[0])); } }
```

ibm.com/developerWorks/webservices

0

Perl

- This uses Paul Kulchenko's SOAP::Lite toolkit (www.soaplite.com):

```
use SOAP::Lite;
print ("PERL SOAP client:\n    ");
print SOAP::Lite
    -> proxy('http://localhost...')
    -> uri('urn:Services:EightBall')
    -> getAnswer()
    -> result;
```

ibm.com/developerWorks/webservices

0

Python

- This uses Cayce Ullman and Brian Matthews' SOAP.py toolkit (see sourceforge.net/projects/pywebsvcs/):

```
#!/usr/bin/env python
import SOAP
print "Python SOAP client:\n  "
server = SOAP.SOAPProxy(
    "http://localhost:8070...",
    namespace="urn:Services:EightBall")
print server.getAnswer()
```

ibm.com/developerWorks/webservices

0

VB Script

- Built with Simon Fell's PocketSOAP (pocketsoap.com):

```
dim env
set env = CreateObject ("pocketSOAP.envelope.2")
dim http
set http = CreateObject
    ("pocketSOAP.HTTPTransport.2")
http.SOAPAction = ""
http.Send "http://localhost...", env.serialize
env.parse http
wscript.echo "Answer from the Great Beyond:" &
    env.Parameters.Item(0).Value
```

0

C++

- Built with the gSOAP toolkit
(www.cs.fsu.edu/~engelen/soap.html):

```
#include "soapH.h" // obtain the generated proxy
int main() {
    struct soap soap; // gSOAP runtime environment
    char* question = "Will I win the lottery?";
    char* answer;
    soap init(&soap); // initialize runtime environment
    if (soap call ns1 getAnswer(&soap,
        "http://localhost:8070/axis/EightBall.jws",
        "", question, answer) == SOAP_OK)
        cout << "The Eight Ball says: " << quote;
    else // an error occurred
        soap print_fault(&soap, stderr); // display msg
}
```


0

The SOAPAction field

- If you're sending SOAP messages across an HTTP connection, you're required to use the **SOAPAction** field in the HTTP header.
- The SOAP spec doesn't define how this field should be used.
 - It could be used by a firewall to filter out SOAP requests, for example.
- This has caused some problems...

ibm.com/developerWorks/webservices

0

Visit **ws-i.org**!

- IBM, Microsoft, and several other companies recently announced the Web Services Interoperability organization, **ws-i.org**.
- Most of the industry has joined the group, with negotiations underway to get the rest of the major players on board.

ibm.com/developerWorks/webservices

Service discovery

0

Service discovery

- Discovery is probably the most hyped and least understood technology in the Web services universe.
- **The heart of discovery is finding a WSDL file**; once we find the complete description of a service, we can build a SOAP envelope and invoke it.

ibm.com/developerWorks/webservices

0

WSDL

- A little over a year ago, IBM and Microsoft submitted the Web Services Description Language to the W3C.
- WSDL is an XML vocabulary that describes a Web service's interface.
- Tools from IBM, Microsoft, BEA, etc., can generate code from a given WSDL file.

ibm.com/developerWorks/webservices

Axis Support for WSDL

- Axis supports WSDL in three ways:
 1. You can get the WSDL file associated with a service by opening the url **`http://my-service's-url?WSDL`**.
 2. You can use the **`Java2wsdl`** tool to generate a WSDL file for your Web service.
 3. You can use **`Wsd12java`** to generate Java proxies and skeletons from existing WSDL files.

Some approaches to service discovery

0

One approach

1. Someone I've never met sends me a file called `readme.exe`.
2. I immediately install it on my system and run it.
3. I *discover* what `readme.exe` does.

bad idea.

ibm.com/developerWorks/webservices

0

Another approach

1. I write a client application that discovers a service in a UDDI registry.
2. I invoke that service, send it my data about my company.
3. I *discover* what the service does.

bad idea.

ibm.com/developerWorks/webservices

Another approach

1. You discover my service in a UDDI registry.
2. You invoke my service sending me your name and credit card number.
3. You *discover* what I do with that information.

ibm.com/developerWorks/webservices

A *sensible* approach

1. Your client application looks for *a particular service* in a WSDL file.
2. Your client gets the WSDL file referenced in the WSDL file.
3. You invoke the service.

great idea!

ibm.com/developerWorks/webservices

0

Another *sensible* approach

1. Your client application searches for *a particular kind of service* (a *threshold*), in a UDDI registry.
2. Your client uses categorization to select one of the providers that has the service you need.
3. You invoke the service.

ibm.com/developerWorks/webservices

0

Web service discovery

0

Web service discovery

- What do we want to discover?
 - ☒ The URL of the machine hosting the service
 - ☒ The ID of the service
 - ☐ The name of the method
 - ☐ The parameters for the method (if any)
 - ☒ The contents of the **SOAPAction** field
- We'll use WSDL, WSIL, and UDDI to discover these things.

0

WSDL and WSIL together

- We'll look at a SOAP application that gets a WSIL document from a Web server, then invokes a Web service based on the information in the WSDL document referenced in the WSIL file.

ibm.com/developerWorks/webservices

0

WSDL and UDDI together

- We'll also look at a SOAP application that gets the URL of a WSDL document from a UDDI registry, then invokes a Web Service based on the information in the WSDL document.

ibm.com/developerWorks/webservices

0

What's next

- With the WSDL file, we have all the information we need to invoke our Web service.
- Now we'll look at demo code for Web service discovery using both WSIL and UDDI.
- **In both cases, our ultimate goal is to find the WSDL file.**

ibm.com/developerWorks/webservices

WS-Inspection

The Web Services Inspection
Language

0

WSIL

- With WSIL, **I know the service provider I want to work with.**
- I typically go to the service provider's Web site and get the file **`inspection.wsil`**.

ibm.com/developerWorks/webservices

0

Getting services ready for WSIL

1. Create the WSIL file (typically `inspection.wsil`)
2. Put the WSIL file on your Web server.
3. Client applications ask for your WSIL file, find the service they want, get the WSDL file that describes it, and invoke the service.

ibm.com/developerWorks/webservices

0

Sample WSDL file

```
<?xml version="1.0"?>
<inspection xmlns="..."
            xmlns:wsilwsdl="...">
  <service>
    <abstract xml:lang="en-US">
      Eight Ball service
    </abstract>
    <name xml:lang="en-US">
      urn:EightBall
    </name>
    <description referencedNamespace="..."
      location="http://localhost/eightball.wsdl"/>
  </service>
</inspection>
```

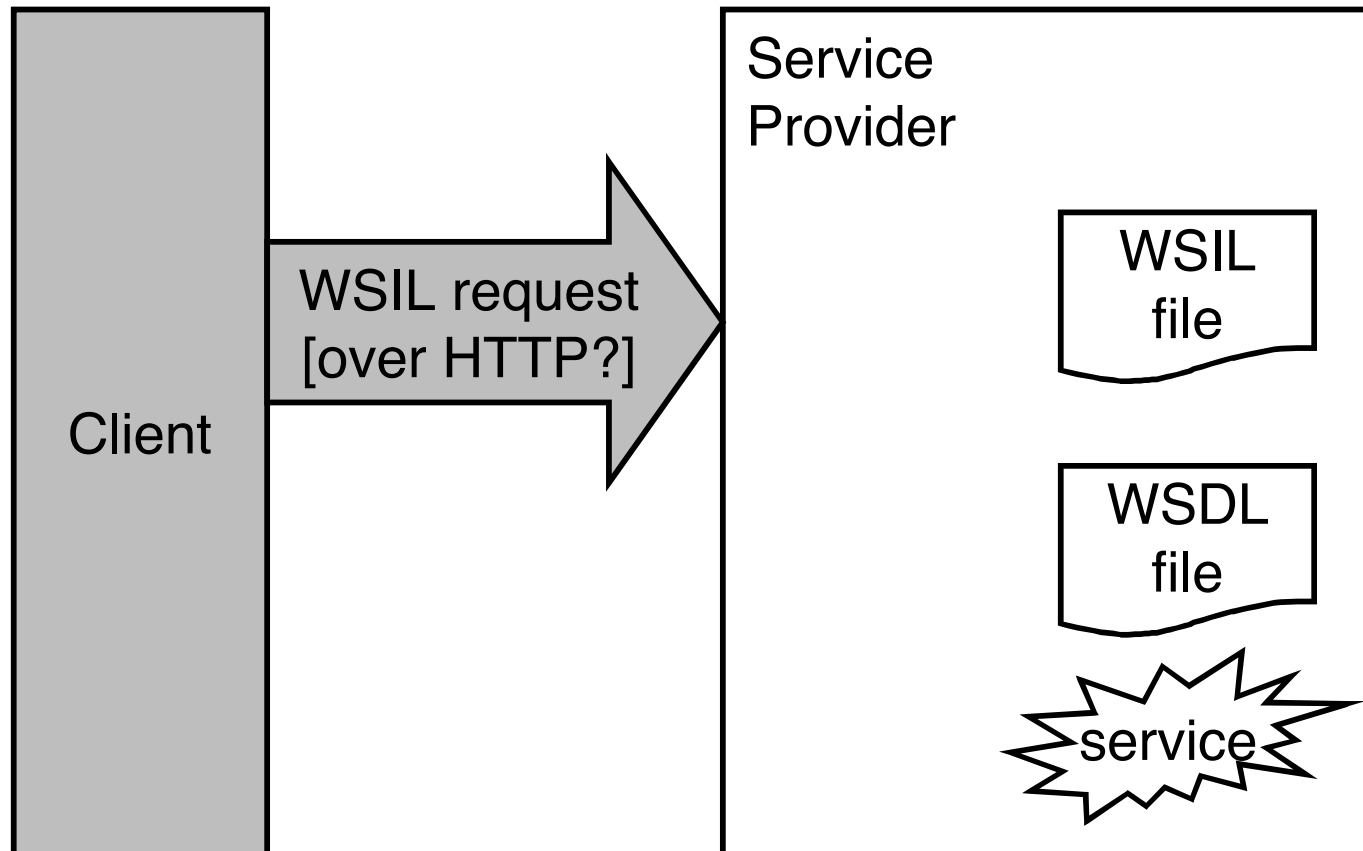
ibm.com/developerWorks/webservices

For your edification and amusement, we present...

The Big Picture for WSIL discovery

0

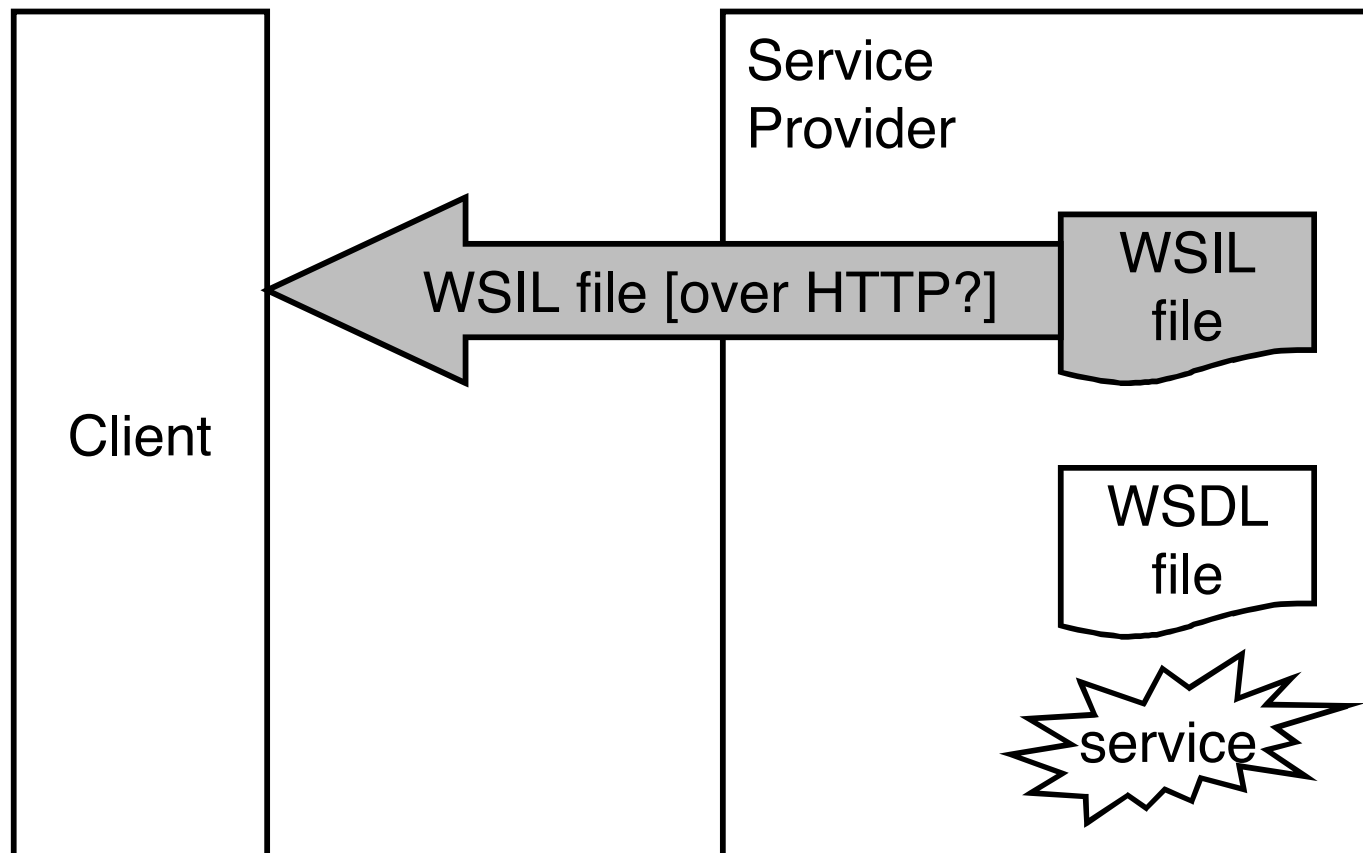
WSIL discovery step 1: The client requests `inspection.wsil` from the server.



ibm.com/developerWorks/webservices

0

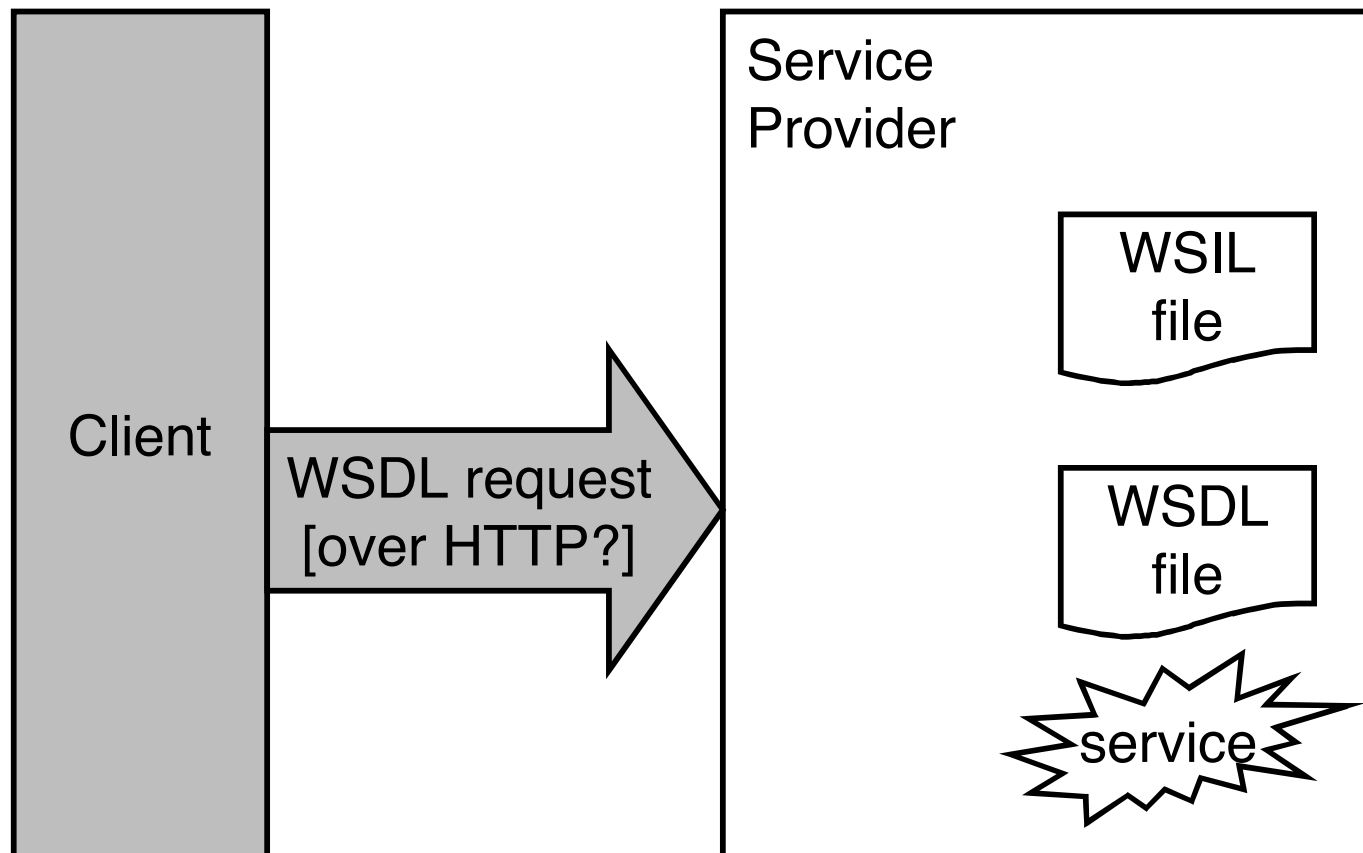
WSIL discovery step 2: The client gets `inspection.wsil` from the server.



ibm.com/developerWorks/webservices

0

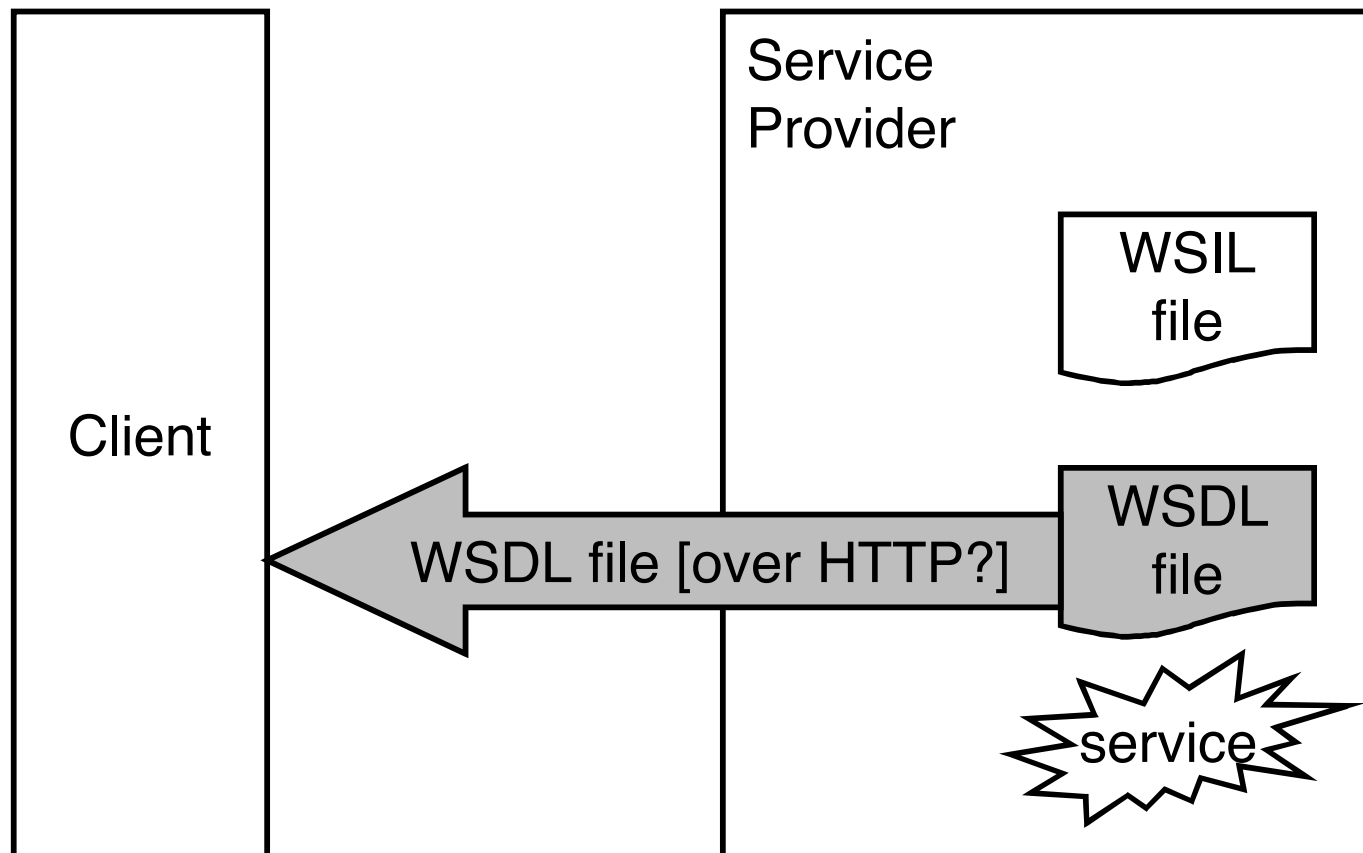
WSIL discovery step 3: The client requests a particular WSDL file.



ibm.com/developerWorks/webservices

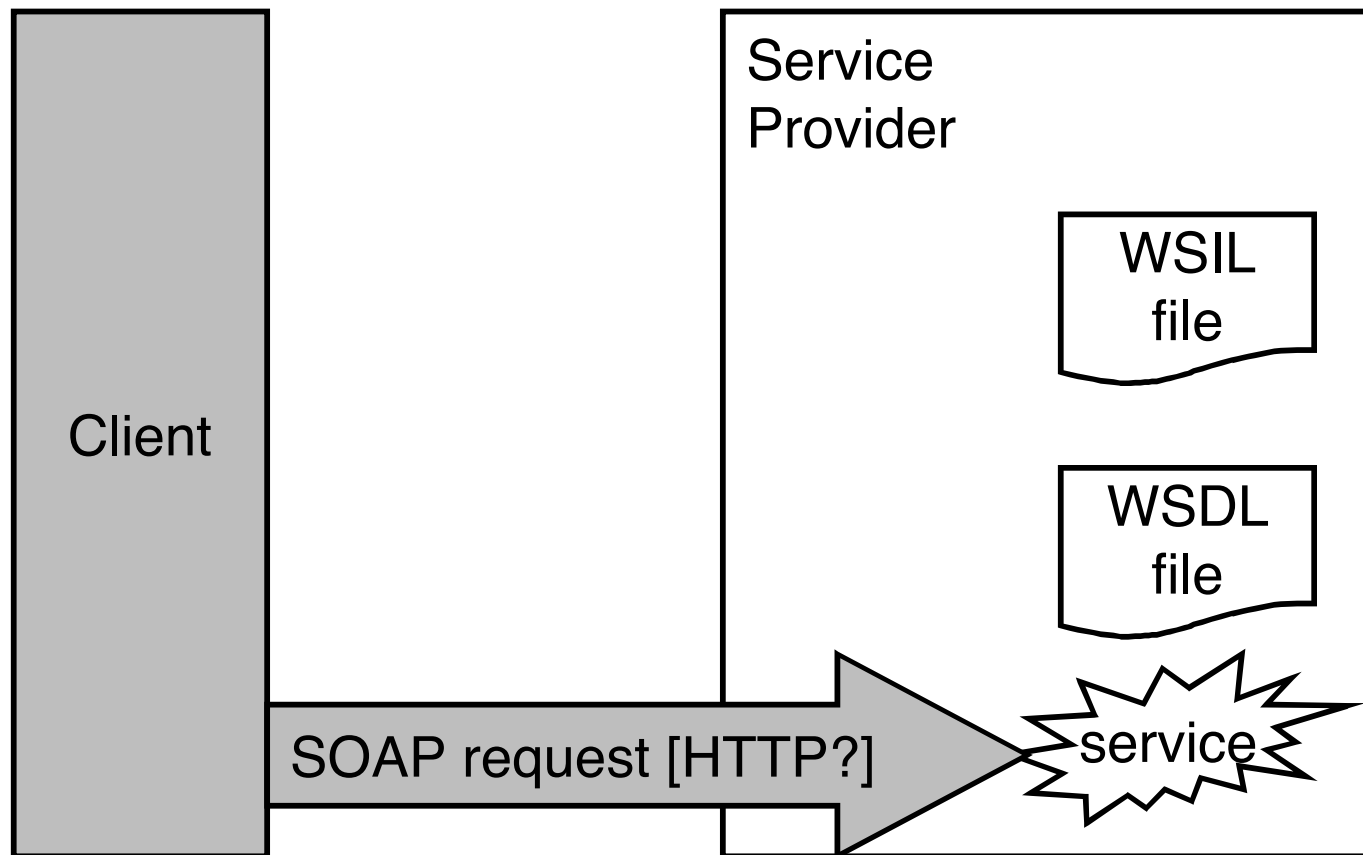
0

WSIL discovery step 4: The provider returns the WSDL service description.



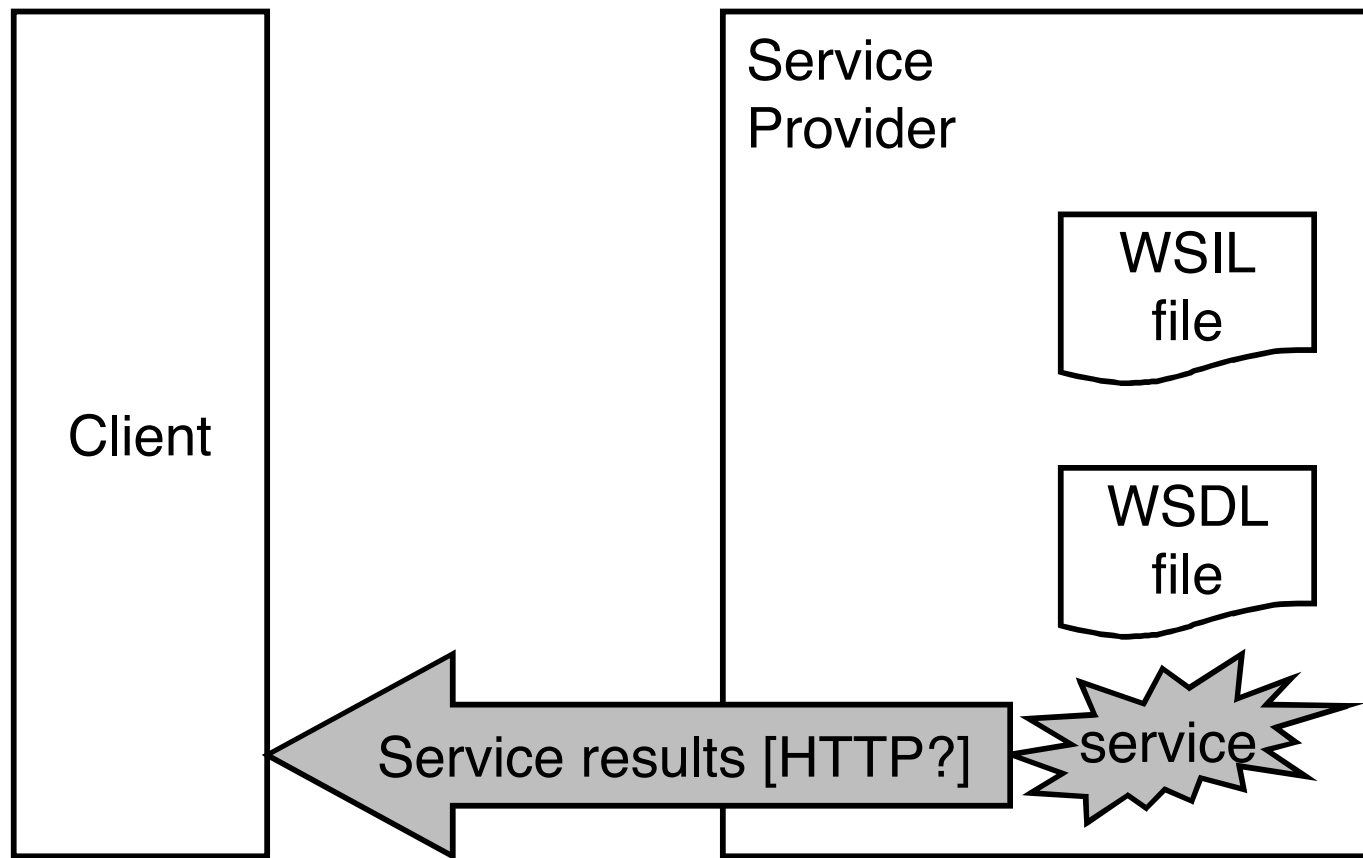
0

WSIL discovery step 5: The client invokes the service with the WSDL info.



0

WSIL discovery step 6: The service delivers the requested data to the client.



0

WSIF

- IBM's Web Services Invocation Framework, recently donated to Apache, simplifies the task of invoking a Web service.
- Both of the discovery demos we'll look at use WSIF; you could build the appropriate `Call` objects yourself if you wanted.

ibm.com/developerWorks/webservices

0

WSIL discovery

- Using WSIL is relatively simple. If you know your service provider, you can find the WSIL file quickly and invoke the correct service.
- UDDI is much more flexible, but more difficult to set up and use...

ibm.com/developerWorks/webservices

Getting ready for UDDI

Setting up your registry

0

Getting services ready for UDDI

1. Split your WSDL file
2. Register the interface portion as a **TModel**
3. Create a **BusinessEntity** to describe your business
4. Add a **BusinessService** record to describe your service. It will refer to the **BusinessEntity** and the **TModel** you just created.

ibm.com/developerWorks/webservices

0

Getting services ready for UDDI

5. Put the WSDL files on your Web server
6. A client application looks for services that implement a given **TModel**, selects a service, gets the details about the service from a WSDL file, and finally invokes the service.

ibm.com/developerWorks/webservices

0

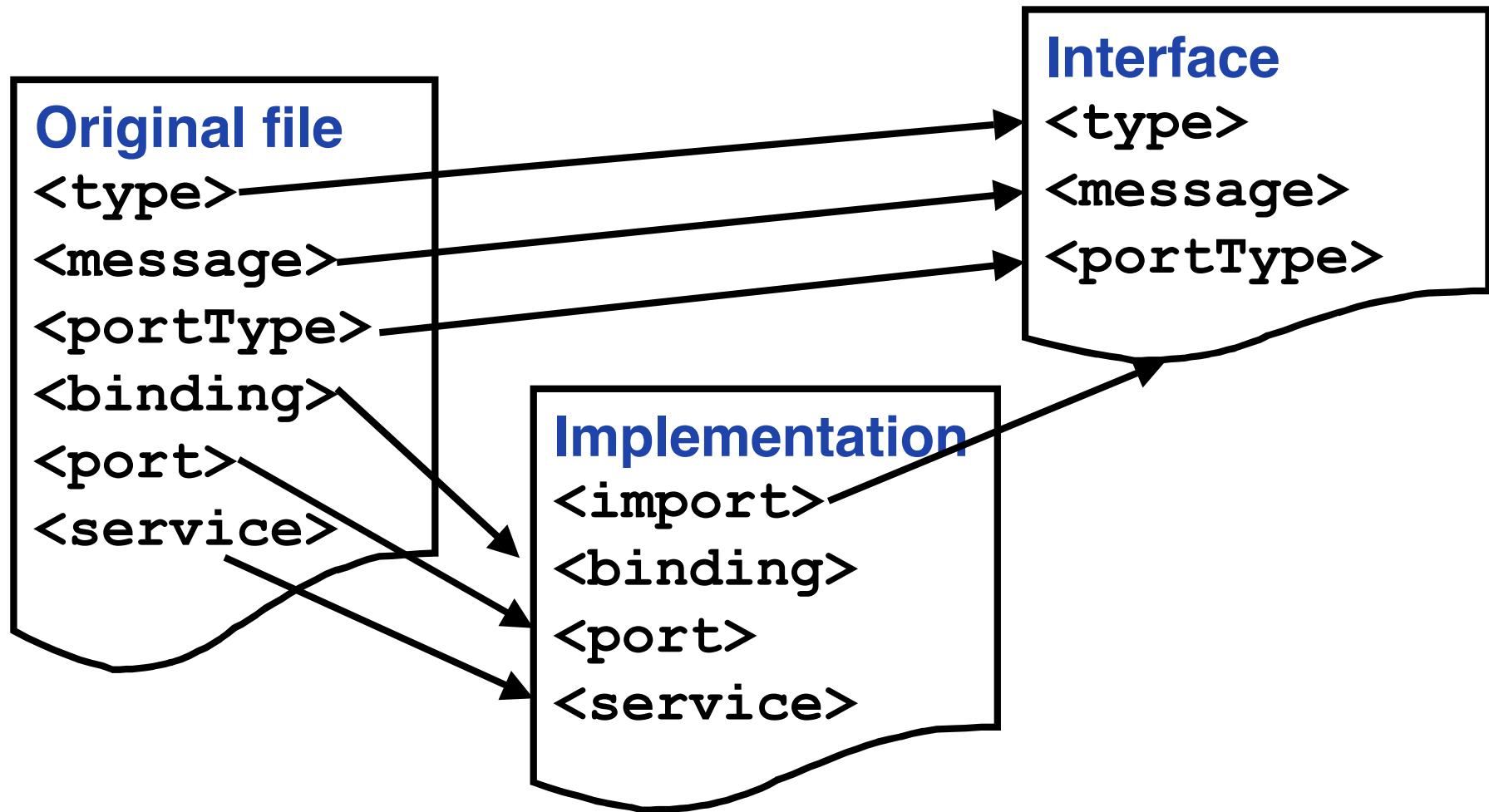
Splitting a WSDL file

- Your WSDL file should be in at least two pieces:
 - The first piece defines the **interface** to any Web service like yours (method names, argument types, etc.)
 - The second piece defines your **implementation** (the address of your server, etc.)

ibm.com/developerWorks/webservices

0

Splitting a WSDL file



ibm.com/developerWorks/webservices

0

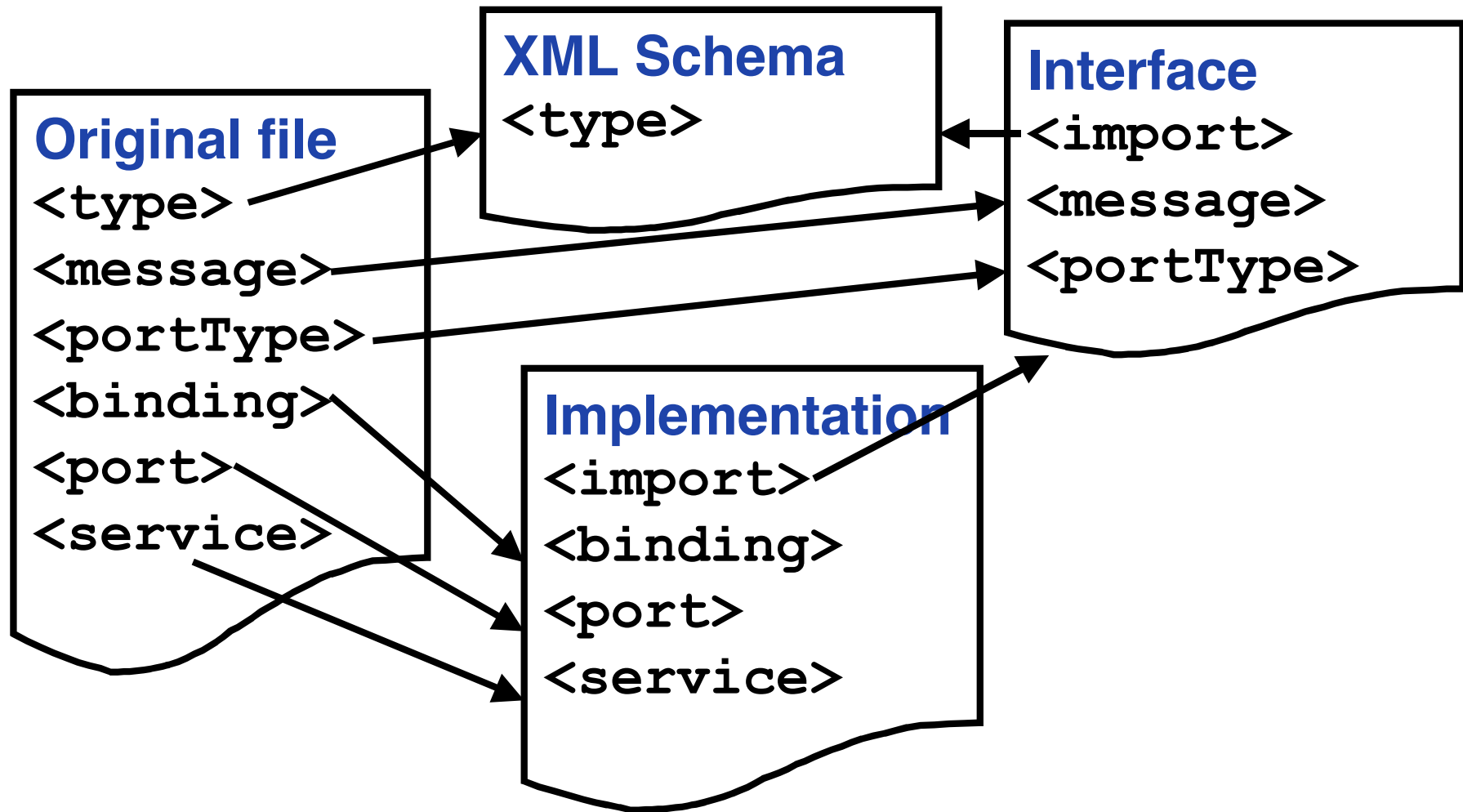
Splitting a WSDL file

- Depending on your application, it may make sense to define your data types (the `<type>` information) in an XML Schema.
- This is particularly useful if you use XML Schemas in several places already.
 - Maybe your Web service is a front-end to a database, and the database structure is built from the XML Schema.

ibm.com/developerWorks/webservices

0

Splitting a WSDL file



ibm.com/developerWorks/webservices

0

Splitting a WSDL file – notes

- In our examples, we put the `<binding>` element in the implementation file; in some cases, it may be better to put it in the interface definition.
- IBM's Web Services Toolkit features `wsdltool`, a utility that splits a WSDL file into two or three parts as we've discussed here.

ibm.com/developerWorks/webservices

0

Working with the registry

- We'll look at some applications that use UDDI4J for the next few steps:
 - Create a **TModel**
 - Create a **BusinessEntity**
 - Create a **BusinessService**
- Later we'll look at sample code that searches the registry to find our service.

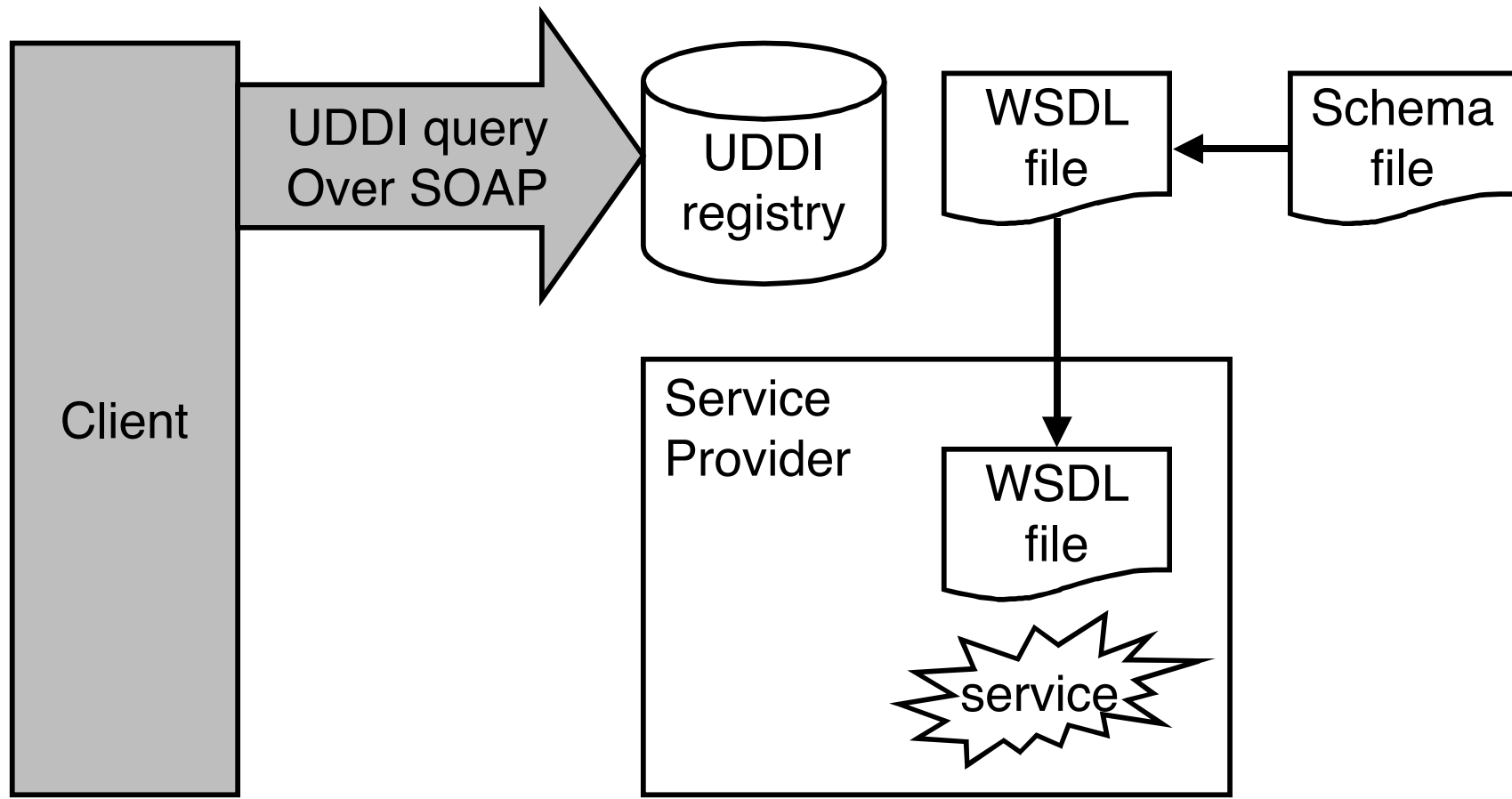
ibm.com/developerWorks/webservices

For your edification and amusement, we present...

The Big Picture for UDDI discovery

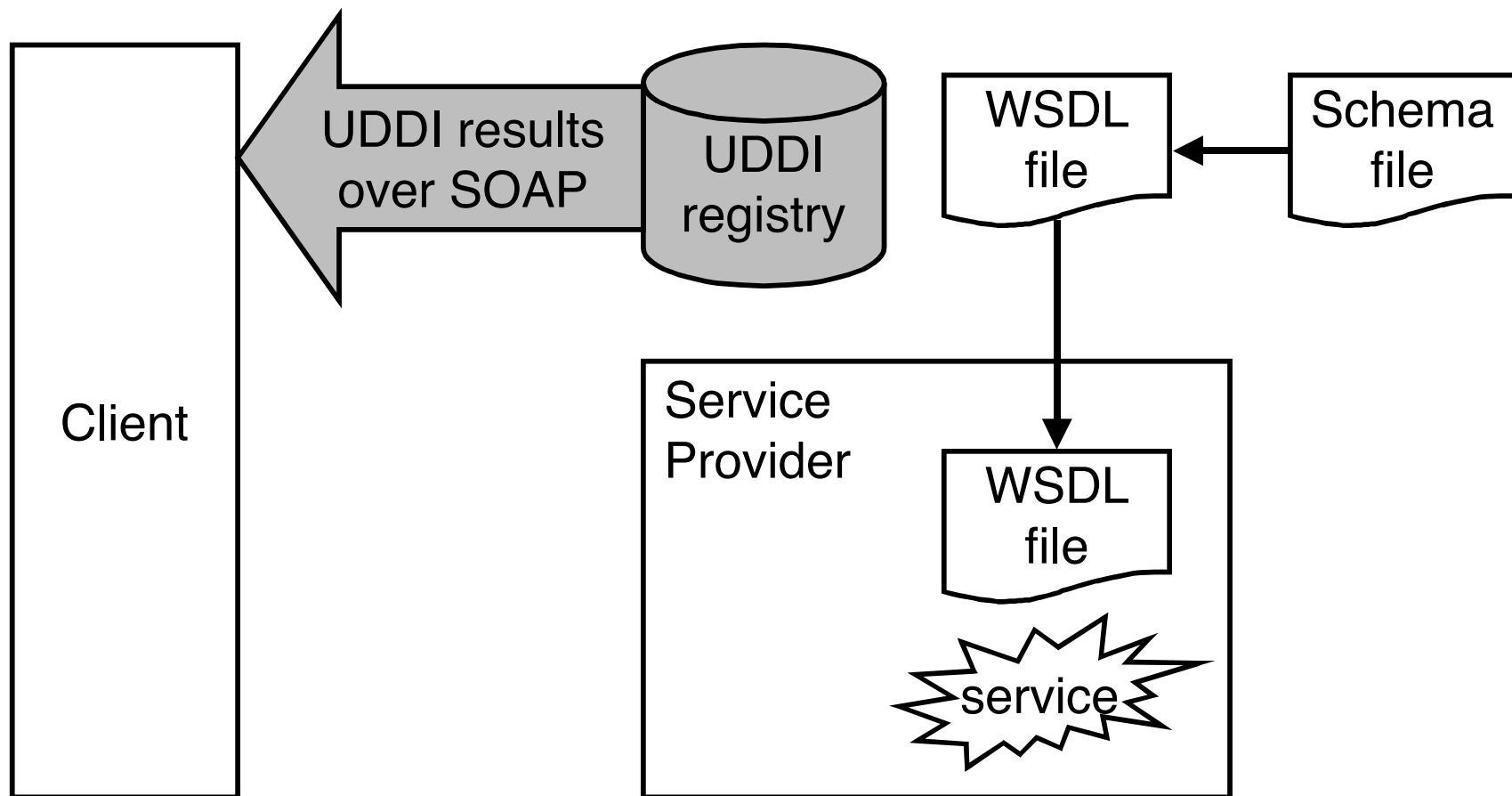
0

UDDI discovery step 1: The client queries the UDDI registry for a service.



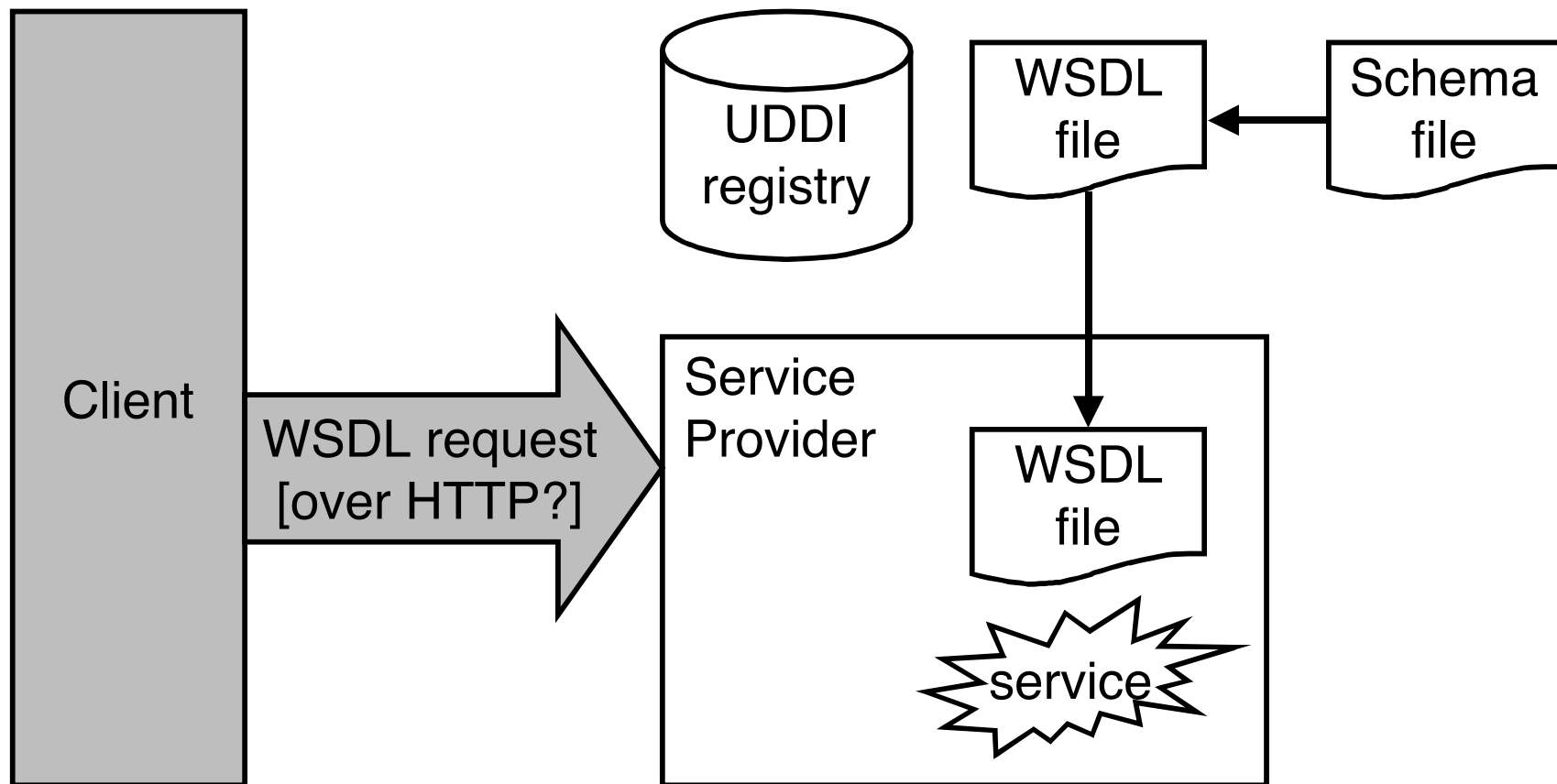
0

UDDI discovery step 2: The UDDI registry sends the query results back to the client.



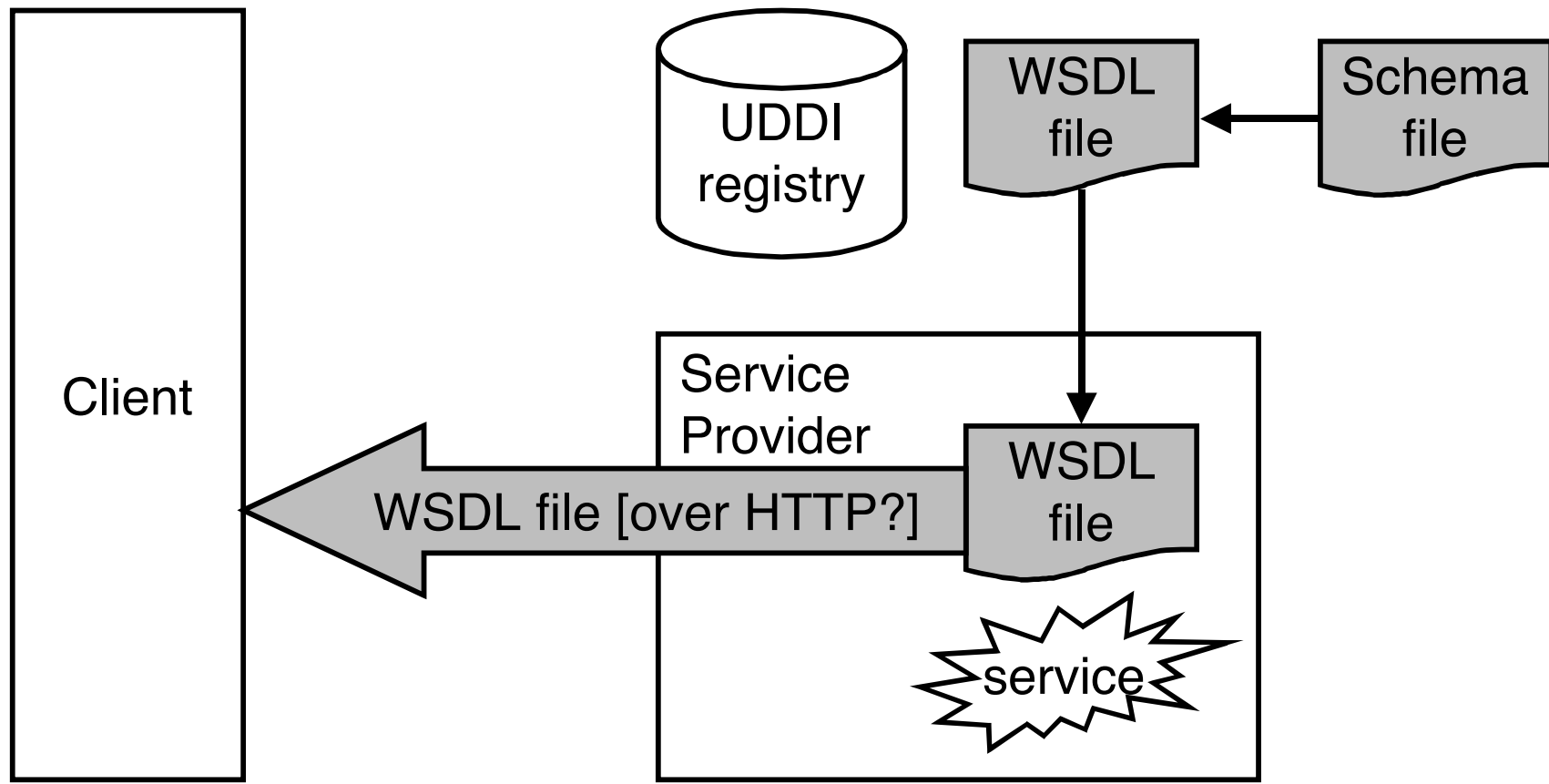
0

UDDI discovery step 3: The client uses the UDDI results to contact the provider.



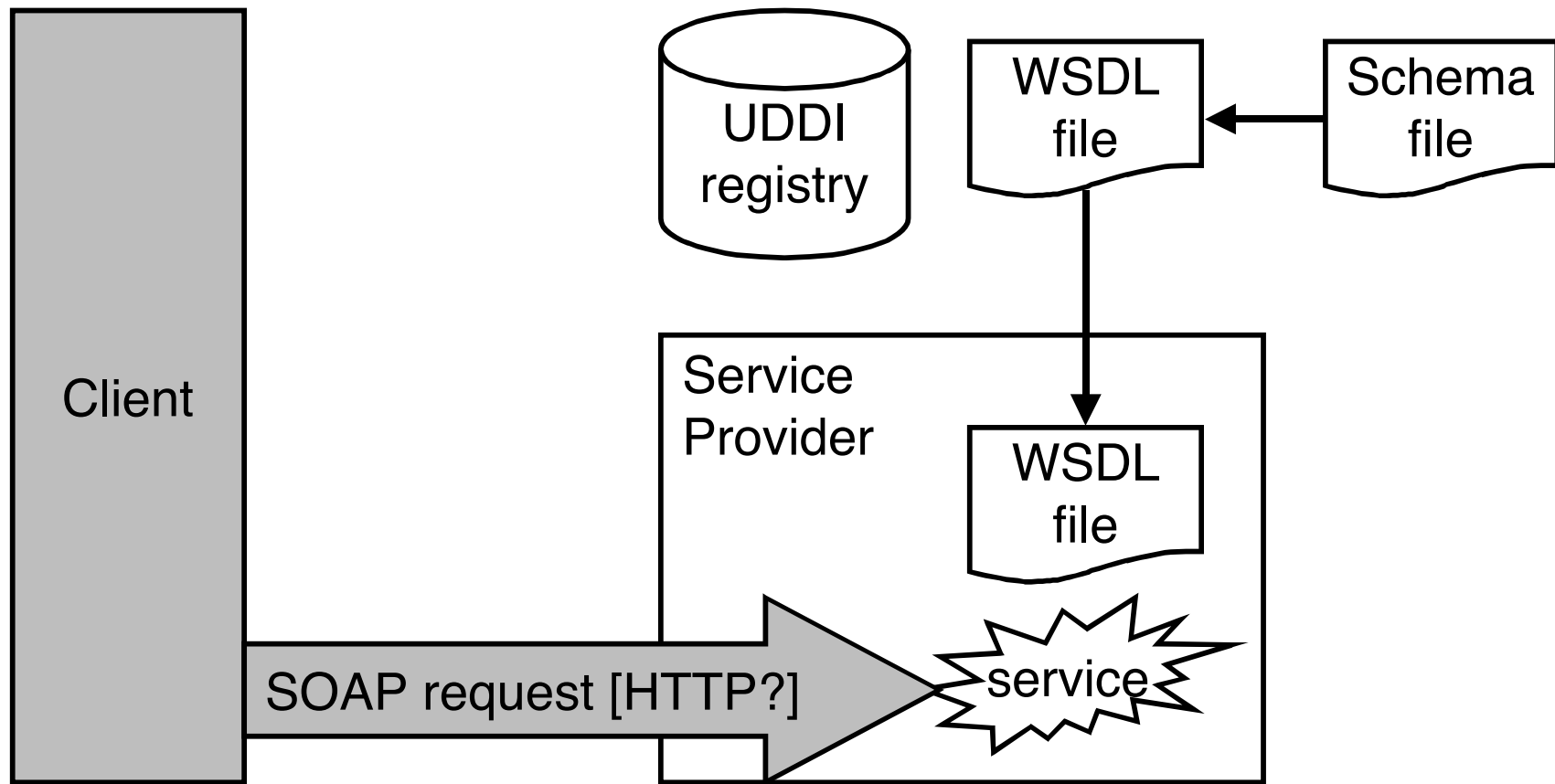
0

UDDI discovery step 4: The provider returns the WSDL service description.



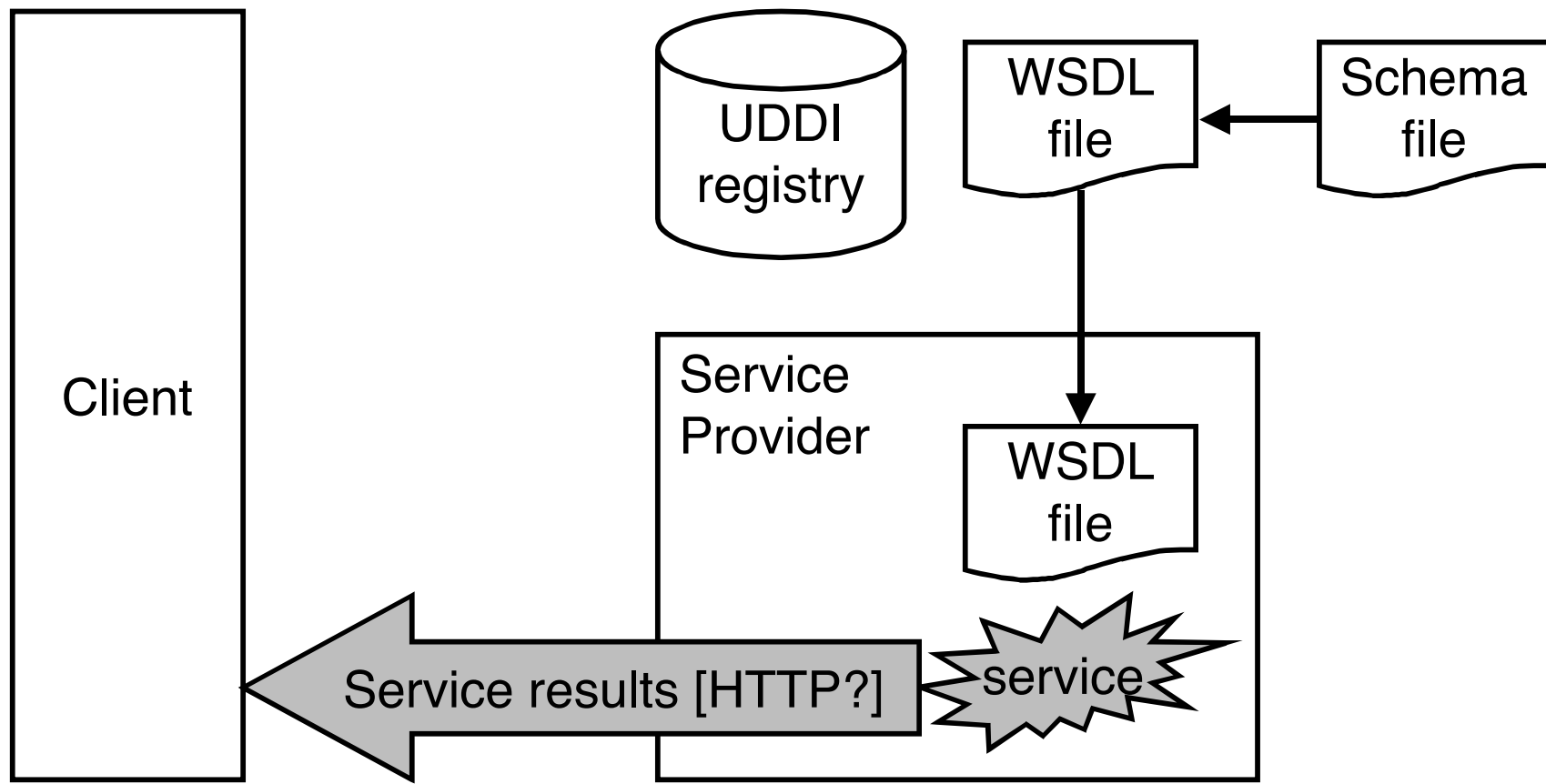
0

UDDI discovery step 5: The client invokes the service with the WSDL info.



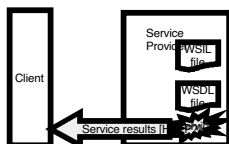
0

UDDI discovery step 6: The service delivers the requested data to the client.



0

The little picture



0

Discovery demos

0

WS-Inspection discovery demo

- We'll get the `inspection.wsdl` file, find the service we're looking for, then invoke it.
- The code is very straightforward; knowing the service provider beforehand simplifies things enormously.

ibm.com/developerWorks/webservices

0

UDDI discovery demo

- Our challenge here is to use UDDI to discover and invoke a particular Web service.
- There's a lot more work to be done here, simply because we don't know our service provider beforehand.

ibm.com/developerWorks/webservices

0

Discovery demos

- At this point, I hope you understand:
 - How discovery works
 - When discovery is a good idea
 - The differences between WS-Inspection and UDDI, and when you should use each one

ibm.com/developerWorks/webservices

0



ibm.com/developerWorks/webservices

Wrapup

The home stretch

0

Wrapup

- XML is the foundation of the Web of the future:
 - TBL's semantic web
 - Web services
 - **.NET**
- XSLT gives you unparalleled flexibility for transforming XML documents.

ibm.com/developerWorks/webservices

0

Wrapup

- All of the tools and interfaces we've covered today are:
 - Based on open standards
 - Available on any Java-enabled platform
 - Open source
 - Available at no cost to you, the home viewer

0

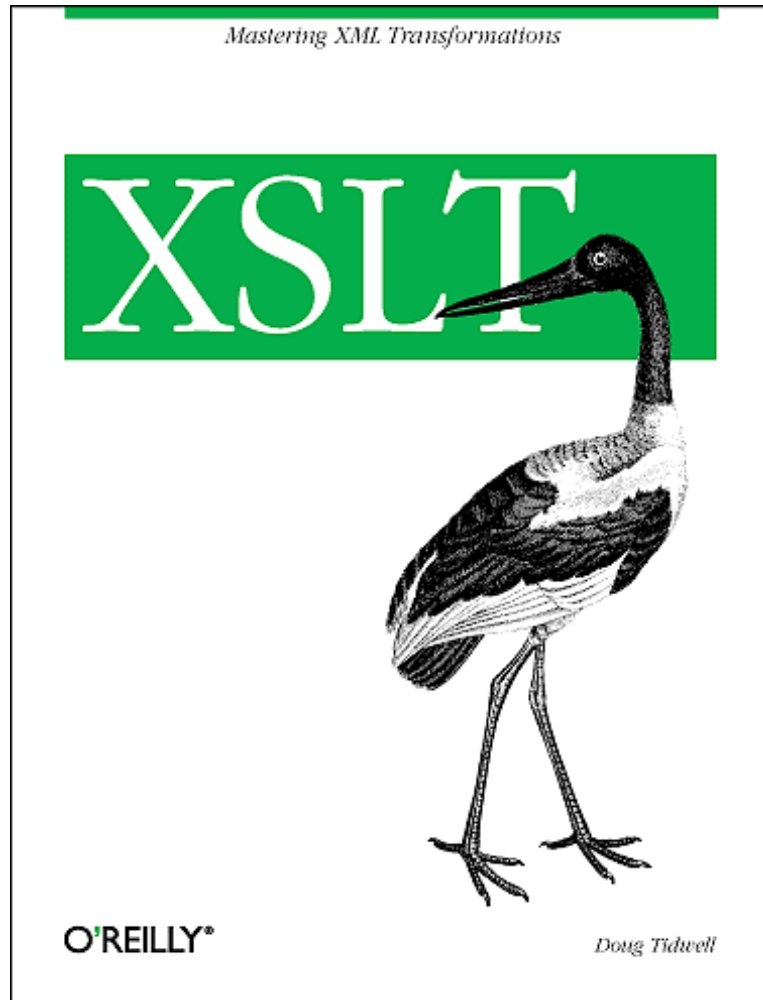
Behold the works of the ASF!

- Our demos here have used:
 - Apache 2.0.39
 - Tomcat 4.0.3
 - Axis 1.0
 - WSIF4J 1.0
 - UDDI4J 2.0
 - WSIL4J
 - WSDL4J

ibm.com/developerWorks/webservices

0

Shameless self-promotion



ISBN 0596000537

Order your copy at
amazon.com today!

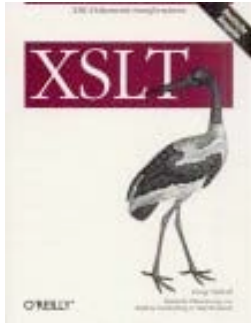
Makes a great holiday gift!

Show your loved ones
how much you care
by showing them the power
of XSLT!

ibm.com/developerWorks/webservices

0

Schamlos – Teil zwei

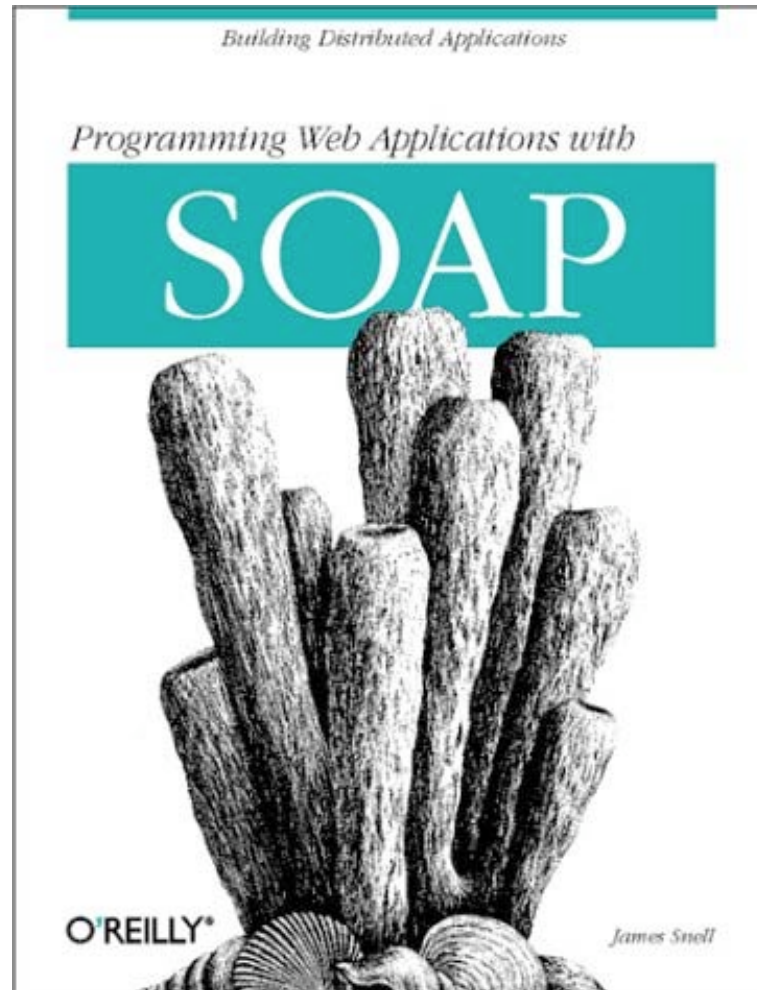


Wer XML bereits kennt und ein Problem schnell lösen möchte oder noch nach einem Problem sucht, für den ist XSLT genau richtig. Doug Tidwell richtet sich an XML-erfahrene Entwickler, die schnell in die komplexen Sphären von XSL eintauchen möchten und Antworten auf ihre eigenen Problemstellungen suchen. – Norbert Hartl
See www.oreilly.de/catalog/xsltger.

ibm.com/developerWorks/webservices

0

Shameless – Part 3



ISBN 0596000952

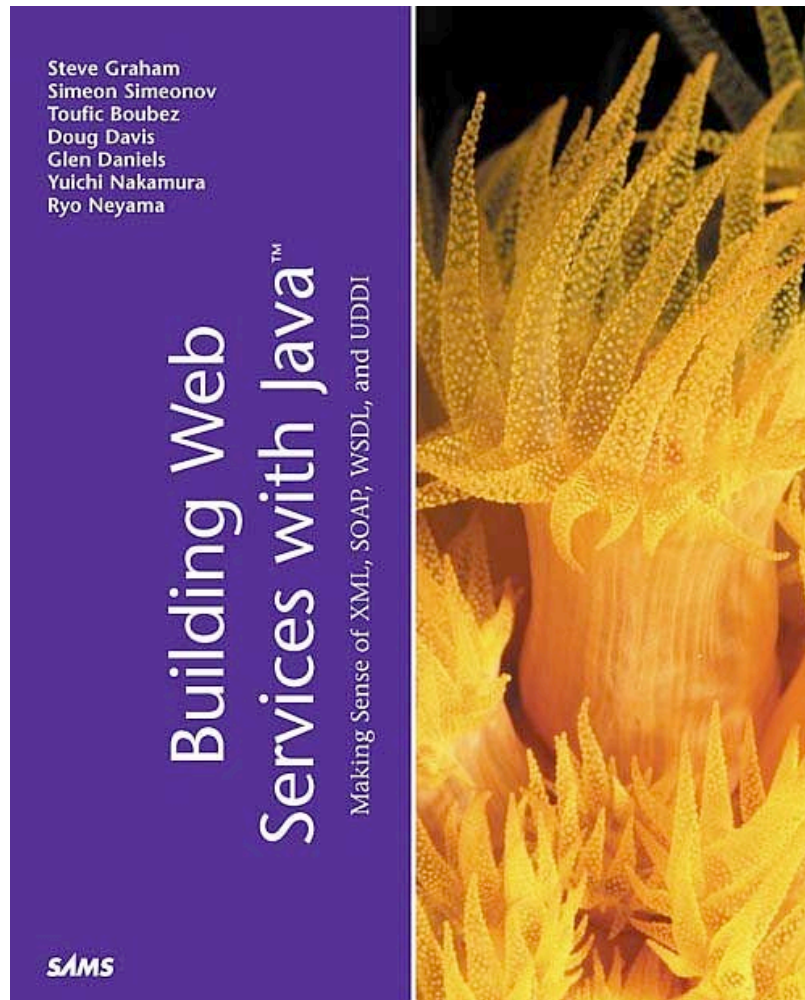
Co-Written with James
Snell (soap-wrc.org)
and Paul Kulchenko.

Available at amazon.com
as well.

ibm.com/developerWorks/webservices

0

Semi-shameless – Part 4



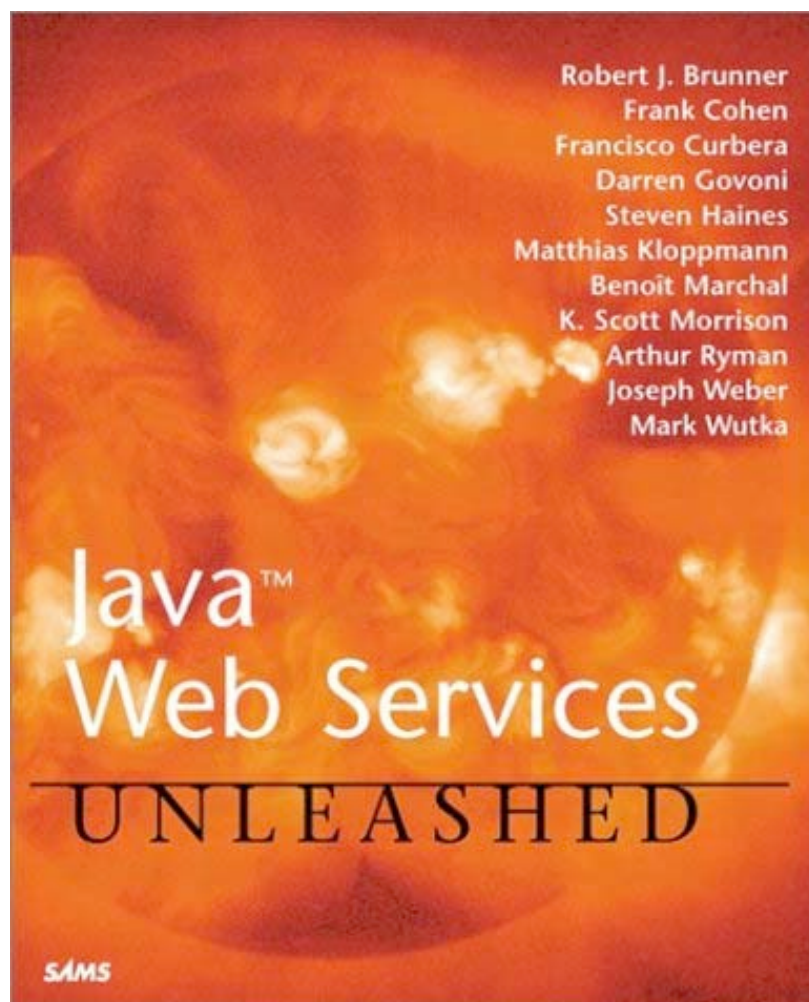
ISBN 0672321815

Written by several talented folks from IBM, Macromedia, and Saffron Technologies, it's a great book on XML, SOAP, WSDL, and UDDI.

Highly recommended!
(After you've bought the other two books...)

ibm.com/developerWorks/webservices

Semi-shameless – Part 5



ISBN 067232363X

Written by IBM's Arthur Ryman and Francisco Curbera, along with other technical luminaries.

Coming soon to a bookseller near you!

ibm.com/developerWorks/webservices

0

These slides and samples...

- ...are available at
`www.ibm.com/developerworks/
speakers/dtidwell.`

[ibm.com/developerWorks/webservices](http://www.ibm.com/developerWorks/webservices)

Thanks for coming!

Doug Tidwell

IBM developerWorks

`dtidwell@us.ibm.com`