Web Services

Mark Volkmann
Partner
Object Computing, Inc.

What Are Web Services?

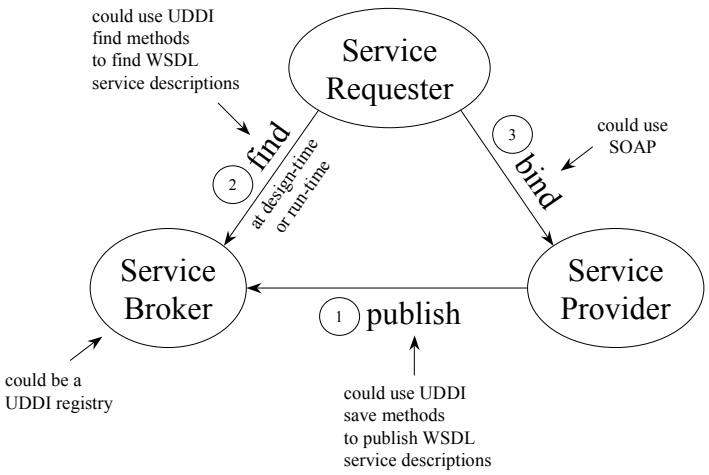
- Software services available over the internet
- Web-accessible components used to create distributed applications
  - can call each other
  - can provide a bridge between other distributed architectures such as CORBA, DCOM, EJB and Tuxedo
- Why now?
  - simplicity
    - HTTP is a simple transport protocol
    - XML is simple message protocol
  - web ubiquity
    - it's everywhere and accessible from many types of devices
  - decrease in bandwidth cost
    - DSL and cable modems now common in homes

Example of a Web Service

- Suppose a company maintains a large inventory of items from many different suppliers
- It must monitor the inventory so that items that are nearly depleted can be reordered on time
- To reduce its own effort it could allow trusted suppliers to monitor the inventory and ship items without waiting to receive an order
- Inventory quantities can be exposed to suppliers through web services
- Suppliers would write applications that use these web services

not exposing the entire database, just a subset

Basic Roles and Operations



Benefits of Using Web Services

- Easier to maintain
  - applications that use web services don't have to track, download and install service code changes
  - service vendor is responsible for maintenance
- More up-to-date
  - applications that use web services can always use the latest versions
- More choices of services
  - can wrap access to any kind of service so not limited to a set based on
    - operating system
    - programming language
    - distributed architecture (CORBA, DCOM, EJB, Tuxedo, …)
- Fewer software configuration issues
  - when using a registry such as UDDI to find services at run-time, code just passes data, doesn't make calls using service-specific APIs
- Easier to debug
  - compared to applications that directly use service-specific APIs because
    - you're responsible for less of the code
    - requests/responses are readable XML
- Loose coupling
  - see "Just-In-Time Integration" on next page
- Legacy apps. interoperate
  - see "Web Services as Glue" on page after next

Just-In-Time Integration

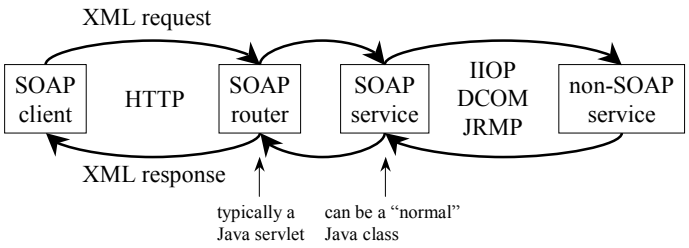
- Need to be able to catalog, discover and invoke services
  - could select from a set of equivalent web services at run-time based on availability, reliability, quality, cost, …
- Not coding to specific interfaces so less coupling than
  - calls to statically bound services
  - other distributed architectures like CORBA and EJB where typically specific services with fixed interfaces are invoked

applications built with these are more sensitive to change, especially API changes

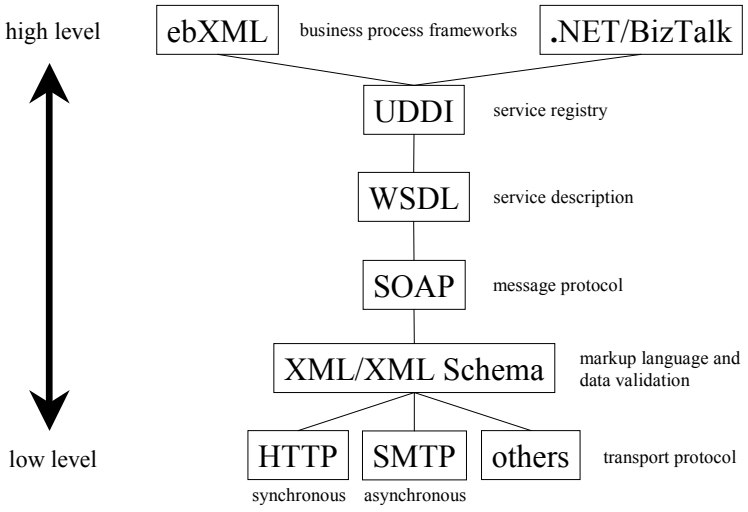
- You just specify the characteristics of the services needed and manage communication with them
  - UDDI tModels describe web service metadata
  - not required to find them this way; can just bind to known services
- Introduces a challenge for testing!
  - different types of UDDI registries provide some safety - see page 25

Web Services as Glue

- Can wrap access to other kinds of distributed services to make them web accessible
- Makes it possible for them to utilize each other
  - for example, EJB methods invoking CORBA services
  - especially useful when companies merge and existing software needs to be integrated



Pieces of the Puzzle



SOAP

- Simple Object Access Protocol
  - an XML-based message protocol that supports Remote Procedure Calls (RPC)
  - requests and responses are XML documents ← uses XML Namespaces and XML Schema
  - can also be used to send one-way messages
  - WSDL greatly reduces need to understand details of the XML
    - more on this later
- Not tied to a particular transport protocol
  - commonly uses HTTP for synchronous communication
    - greatly simplifies firewall issues
  - can use SMTP for asynchronous communication
- Simplicity is a key benefit
  - compared to other distributed architectures such as CORBA, DCOM and EJB

SOAP Standardization

- SOAP 1.2 spec. is a W3C Working Draft
  - see SOAP Version 1.2 Part 1: Messaging Framework and Part 2: Adjuncts at http://www.w3.org/2000/xp/
- Also "SOAP Messages with Attachments" W3C Note
  - http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211
  - specifies how SOAP messages can include attachments such as binary image data
  - allows all data needed by a service to be passed in one request message
- W3C XML Protocol (XMLP) Working Group
  - formed in September 2000
  - used original SOAP Note as a starting point is further defining it
  - three participants of the AXIS project within Apache are members
  - watch progress of this at http://www.w3.org/2000/xp

Who's Backing SOAP?

- SOAP has broad vendor support
- These companies and more have representatives on the W3C XML Protocol Working Group
  - Allaire, AT&T, **BEA Systems**, Bowstreet, Compaq, Commerce One, DevelopMentor, Fujitsu, **Hewlett Packard**, **IBM**, Informix, Intel, **IONA**, **Microsoft**, MITRE, Netscape, Novell, **Oracle**, Rogue Wave, Software AG, SAP AG, **Sun Microsystems**, Unisys, Xerox

SOAP Performance

- A concern since many steps are required to execute a call
  - client
    - create XML-based request
    - create and send HTTP (or some other transport protocol) request to server
  - server
    - parse HTTP request
    - parse XML request within HTTP request
    - possibly create objects from XML to hold parameter data
    - make service call
    - create XML-based response from return value
    - create and send HTTP (or some other transport protocol) response
  - client
    - parse HTTP response
    - parse XML response within HTTP response
    - possibly create objects from XML to represent result

could be an object containing lots of data

When Is SOAP Appropriate?

- Invoking code outside firewall
  - most other distributed architectures have difficulty with this
  - SOAP does this by communicating on port 80 which is typically open
  - don't need special web servers, routers, firewalls or proxy servers
  - firewalls can filter SOAP traffic
    - based on HTTP "Content-Type" header which is "text/xml"
    - based on "SOAPAction" HTTP header
      - specifies message intent of which may be the name of the service being invoked
- Invoking code written in a variety of languages
  - available to more languages than CORBA
  - only need support for XML and HTTP
  - clients and servers can even be written in scripting languages

When Is SOAP Appropriate? (Cont'd)

- Invoking code not implemented with a particular distributed architecture
  - CORBA, DCOM, EJB, RMI, Tuxedo, …
  - these have difficulty communicating with each other
  - can wrap these calls in SOAP services to allow them to interoperate
- Invoking course-grained services
  - fine-grained SOAP services would be too expensive in terms of network traffic and message construction/parsing overhead
- Invoking code whose performance is not critical
  - SOAP will likely be slower than other distributed architectures due to XML generation/parsing and using HTTP

When Is SOAP Appropriate? (Cont'd)

- Don't have software required by other distributed architectures
  - perhaps due to cost; currently most SOAP implementations are free
  - if considering CORBA, perhaps to gain features or performance, checkout The Ace ORB (TAO), free CORBA implementation at http://theaceorb.com/
- Don't have expertise to use other distributed architectures
  - less training required compared to other distributed architectures
  - many important features such as transaction support were purposely omitted to keep the design simple; features like this will be layered on by other specs. (see next page)
  - translating data to and from XML is done for you by toolkits
- Passing hierarchical data
  - better than using HTTP which posts name/value pairs since XML data is structured

SOAP's Missing Pieces

- Other capabilities will be layered on SOAP by other specs.
  - security
    - authentication and authorization
    - see XKMS, S2ML and AuthXML
  - transactions
    - see XAML and XLANG
  - payment
    - billing for use of services
    - possible models include subscription and per use charges
  - reliability
    - such as guaranteed message delivery in the event of client or server failures
  - quality of service
    - prioritized requests
    - responses in a guaranteed amount of time

Comparing SOAP To Other Distributed Architectures

- Dependence on platform, language or protocol
  - DCOM is Windows platform-specific
  - Java RMI and EJB are Java language-specific
  - CORBA uses a specific protocol (IIOP)
  - SOAP doesn't depend on a particular platform, language or protocol
- Object-orientedness
  - CORBA, RMI and other OO distributed architectures support stateful, object-oriented methods on remote objects
  - SOAP is more suited to executing remote, standalone functions (services)
  - SOAP doesn't operate on or pass remote object references
    - although you could pass object ids that would be used to find server-side objects that live across requests
  - service implementations can be OO, but calls to them aren't particularly

Comparing SOAP To Other Distributed Architectures (Cont'd)

- Encoding/Protocol
  - SOAP messages are encoded as XML text
  - other distributed architectures use incompatible binary encodings
    - good for performance, but bad for debugging and interoperability
- Adaptability to the internet
  - IIOP and DCOM don't adapt well, at least as currently implemented
  - SOAP can send requests and receive responses using HTTP

Replacement For Similar Technologies?

- No!
  - remote services can still be defined and accessed using other distributed architectures such as CORBA, DCOM and EJB
  - in situations where it is useful to access those services over HTTP, they can be wrapped as SOAP services
    - for example, could make selected EJB session bean methods web accessible
    - see "Web Services as Glue" on page 7
  - when more efficiency is needed, faster, non-SOAP protocols can be used
- Consider separating business logic from code that is specific to a distributed architecture
  - similar to the way business logic is typically separated from user interface code

SOAP Server Implementation

- Only message content is standardized, not an API
  - pro - each language-specific SOAP toolkit can be tailored to take advantage of the strengths and style of a particular programming language
  - con - experience gained in using SOAP from one programming language doesn't transfer over to using it from a different one
  - each implementation decides how it maps SOAP requests to service calls
- Typical OO implementation
  - accept an HTTP request containing a SOAP request
  - instantiate a server object of some class indicated in SOAP request
    - when activation mode is "request" (as opposed to "session")
  - unmarshall XML arguments to objects of classes indicated in request
  - pass argument objects to a method of the server object indicated in request
  - marshall the return value into a SOAP response
  - send SOAP response to client in an HTTP response

WSDL

- Web Services Description Language
  - version 1.1 spec. has been submitted to the W3C as a Note
    - by Ariba, IBM and Microsoft
    - available at http://www.w3.org/TR/wsdl
- Describes web service requests and responses in XML
  - similar to CORBA IDL but also includes the location of services (via a URL)
  - two kinds of descriptions, service interfaces and implementations
    - allows multiple implementations of the same interface
  - these can be cataloged and searched in a registry such as UDDI
- Supports four kinds of operations
  - one-way (client to server)
  - request-response (client to server and back)
  - solicit-response (server to client and back)
  - notification (server to client)

WSDL Standardization

- W3C standardization of this concept is likely to begin soon
  - probably will use WSDL as a starting point in the same way that the W3C XMLP group is using SOAP as a starting point
  - more on XMLP later

The importance of WSDL will be emphasized later!

UDDI

- Universal Description, Discovery and Integration
- Provides a registry for web services
  - similar to CORBA Naming and Trader services
- Can register and find several types of information
  - white pages (service provider data)
    - info. about service providers such as business name, description, contacts and references to yellow pages data
  - yellow pages (high-level service data)
    - service name, description, category list and reference to green pages data
    - services can be listed by several taxonomies (NAICS, UN/SPSC, geographical)
      - North American Industry Classification System (NAICS)
      - Universal Standard Products and Services Classification (UN/SPSC)
  - green pages (low-level service data)
    - service location, protocol to use (such as SOAP) and parameter info.
    - can be supplied by referencing a WSDL file

UDDI Consortium and Registries

- Consortium
  - UDDI was created by a consortium of 36 companies including IBM, Microsoft and Ariba
  - as of May 2001, there were 260 member companies including
    - BEA Systems, HP, IBM, Intel, IONA, Microsoft, Oracle, Rational, SAP, Sun Microsystems
    - Boeing, Dell, Fujitsu, Merrill Lynch, Nortel Networks, ObjectSpace, SilverStream, TIBCO, Verisign
- Registries
  - IBM and Microsoft host free (currently) UDDI repositories
    - HP will host one by the end of 2001
  - services advertised to any of them are replicated to the others within 24 hours
    - typically much faster

Proposed UDDI Registry Types

(answers question of how dynamic service binding can be made more reliable)

- UDDI operator
  - currently only IBM and Microsoft; soon HP
  - could reduce risk of invoking a "bad" service by only using to find businesses and services at design-time, not dynamically binding
- e-marketplace
  - populated with a collections of related, legitimate businesses and services
  - can restrict publish, find and bind usage to members
- Portal
  - hosts business descriptions and services of a single company
  - can restrict access to specific customers and monitor usage (what is being used and by whom)
- Partner catalog
  - company owned registry containing only entries from trusted partners
  - only used from within the company
- Internal enterprise
  - like partner catalog but only contains entries from departments within company
- Test bed
  - to test UDDI entries, web services and applications that use them

ebXML

- Electronic business XML framework
  - primarily targeted toward B2B communication
- Defines standard
  - business processes
  - message structures
    - based of SOAP Messages with Attachments
      - more on this later
  - company profile descriptions
  - trading partner agreements
  - registry for publishing and finding business processes
    - defines its own registry, but implementations could use UDDI

ebXML Goals

- "Enable a global electronic marketplace where enterprises of **any size** and in **any** geographical **location** can meet and conduct business with each other through the exchange of **XML based messages**"
- Improve on Electronic Data Interchange (EDI)
  - be more cost effective
  - eventually provide more functionality
- Allow businesses to automate the following with no human involvement
  - find partners that support specific business processes
  - enter into trading partner agreements with them
  - invoke their web services

Who's Supporting ebXML?

- Standards organization support
  - UN/CEFACT
    - **U**nited **N**ations **Ce**ntre for Trade **Fac**ili**t**iaon and Electronic Business
  - OASIS
    - Organization for the Advancement of Structured Information Standards
    - a big player in XML-related standards
      - DocBook (a DTD for computer documentation), conformance test suites for XML, XPath and XSLT, Relax NG (alternative to XML Schema), Security Assertion Markup Language (SAML) and more
  - OMG
    - Object Management Group
      - moving toward publishing specifications that apply to all distributed architectures, not just CORBA
- Significant vendor support
  - IBM, Oracle, Sun (not Microsoft since they prefer their .NET framework)
  - Ariba, Commerce One, DataChannel, Fujitsu, Mitre, NEC, PeopleSoft

the UN also sponsors EDIFACT (EDI for Administration, Commerce and Transport)

Microsoft .NET / BizTalk

- Has basically the same set of requirements as ebXML
- .NET is a framework of server products
- BizTalk is one of them
- Uses other standards
  - SOAP as message protocol
  - WSDL to describe services
  - UDDI for service registry
  - XLANG to model business processes
- Seems complex
  - compared to just using SOAP, WSDL and UDDI
  - opinion gained from scanning books on .NET

Current State of Web Services

- Current web service tools
  - suitable for creating simple web services and toy examples, not enterprise-level services
  - tools need more time to evolve
  - experiment now to gain experience and provide feedback to tool developers

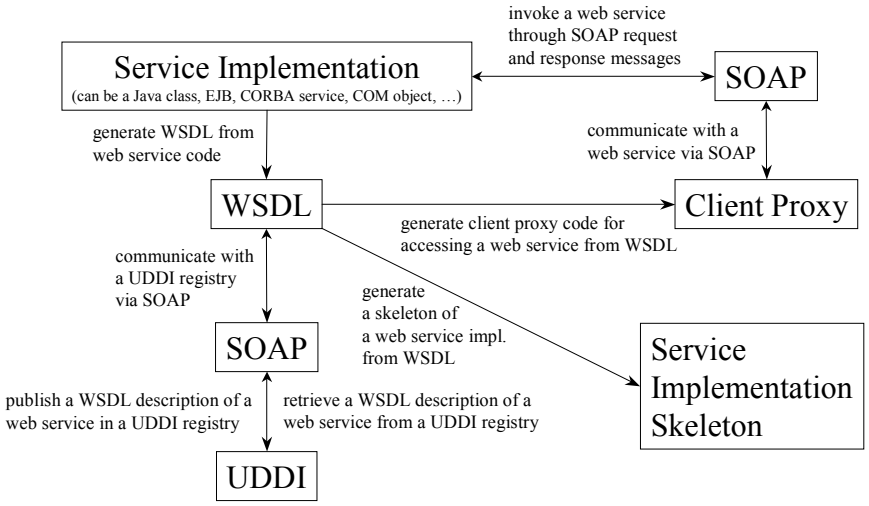
need support for security, transactions and more

- As of August 2001 there were at least 89 SOAP implementations (see http://www.soapware.org)
  - covering a wide variety of operating systems and programming languages
  - Apache AXIS, GLUE from The Mind Electric, IBM Websphere and Microsoft .NET are likely to be the first enterprise-level tools
- Interoperability between these still needs work
  - they don't support the same feature subsets
  - the Microsoft and Apache teams are working to improve interoperability

WSDL is the Key!

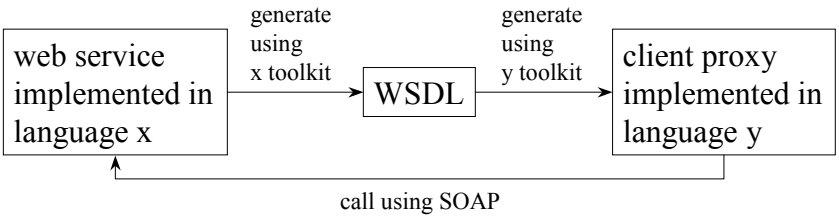
- WSDL can
  - describe a web service, including it's location
  - be used to invoke a web service
  - be generated from an existing web service implementation
  - be used to generate a client stub for a web service
    - provides compile-time type checking of parameters being passed to a web service
  - be used to generate a web service implementation skeleton
    - in the case of Java, includes class definition with empty methods
- SOAP is under the covers
  - no need to create or parse SOAP XML requests and responses
    - toolkits do it for you
  - no need to understand structure of SOAP XML requests and responses
    - although it can be useful for debugging
  - likewise there's no need to understand structure of WSDL

General Web Service Toolkit Functionality



Future Made Possible By WSDL

- Will be able to easily make calls between any programming languages that can
  - generate WSDL from web services implemented in the language
  - generate client proxies in the language from WSDL



Web Services Toolkit (WSTK)

- From IBM alphaWorks
  - http://www.alphaworks.ibm.com/tech/webservicestoolkit
- Composed of the following components
- WSDL Generator Tool
  - creates WSDL and Apache SOAP deployment descriptors from web service implementation classes
    - don't have to implement or extend any particular interface or class
  - uses reflection to allow selection of methods to expose in a Swing GUI
- Service Proxy Generator
  - generates client stub Java code from WSDL
- Service Implementation Template Generator
  - generates the skeleton of a web service implementation from WSDL
- Web Services Browser
  - publishes, unpublishes and finds web services described by WSDL in a UDDI repository using a Swing GUI
- WSDL Document Classes
  - creates and modifies WSDL from Java code
- UDDI4J client API
  - saves, deletes and finds data in a UDDI registry from Java code
- Private UDDI registry
  - can also use public UDDI registries
- Apache SOAP

Apache eXtensible Interaction System (AXIS)

- From Apache
  - http://xml.apache.org/axis
  - current version is alpha 2
- WSDL support
  - ServiceClient class can invoke any web service described with WSDL
    - need to supply URL of SOAP router, URL of WSDL, operation name and operation parameters
    - no compile-time parameter type checking
    - supports proxy servers
  - Wsdl2java tool generates
    - client stubs for type-safe invocations
    - service implementation skeletons for implementing services described by WSDL
  - automatically generates WSDL for deployed web services
    - clients can access by appending "`?WSDL`" to the URL of the web service which is typically `http://<host>/axis/services/<service-name>`

AXIS (Cont'd)

- Web service deployment
  - instant deployment
    - simply copy a .java file to axis web app. directory and change the extension to .jws (for Java Web Service)
  - custom deployment using a deployment descriptor
    - allows more control for deployment without source code, custom type mappings and more
- Type mapping
  - refers to serializing Java objects to an from XML in SOAP messages
  - handles automatically for specified Java classes that follow the Java Beans pattern
  - can customize for specified Java classes
- SOAP message monitoring
  - TCP Monitor tool monitors SOAP request and response messages

WebLogic Server

- From BEA
  - http://www.bea.com/products/weblogic/server
  - current version is 6.1
- A commercial Java server product that supports
  - servlets, JSP, JDBC, EJB, JMS, web services and more
  - version 6.1 is first to offer web service support
- Features
  - supports SOAP 1.1 with Attachments and WSDL 1.1
  - web service requests are routed by a special servlet
- Missing
  - client proxy generation from WSDL
  - service skeleton generation from WSDL
  - support for UDDI

WebLogic Supports Two Types of Web Services

- WebLogic web services must be implemented as stateless session beans or JMS destinations
  - a heavy requirement in terms of development time
  - other web service toolkits don't require this
  - what is needed is a utility to automatically generate EJB code from a "normal" Java class
- Which type to use?
  - use session beans for synchronous, RPC-style web services
  - use JMS for asynchronous, message-style web services

WebLogic RPC-style Web Service Steps

- Steps (all of which can be automated using Ant)
  - implement web service as a stateless session EJB
    - remote interface, home interface and bean class
  - compile these .java files
  - create a JAR file containing
    - .class files, ejb-jar.xml and weblogic-ejb-jar.xml ← these XML files specify characteristics of the EJBs
  - run weblogic.ejbc on the JAR file to create
    - an EJB JAR file containing the contents of the first JAR plus EJB container-generated classes
    - a JAR containing only the classes needed by client applications
  - run wsgen on the EJB JAR file to create an EAR file
    - enterprise archive
  - deploy the EAR file to WebLogic
    - copy to WL_HOME/config/*domain*/applications

WebLogic Web Service Web Pages

- A web page is automatically generated for each web service context
  - url is `http://weblogic-host:weblogic-port/bean-name`
- Contains
  - a link that returns WSDL for all operations defined for the context
    - generated automatically!
  - a link to a client JAR file
    - contains all the code necessary for Java-clients to invoke the operations
    - requires Clients to hard-code knowledge of the EJB remote interfaces
    - if you're going to use this, what's the point of using SOAP?
    - for Java client to Java service implementations, EJB calls can be used

GLUE

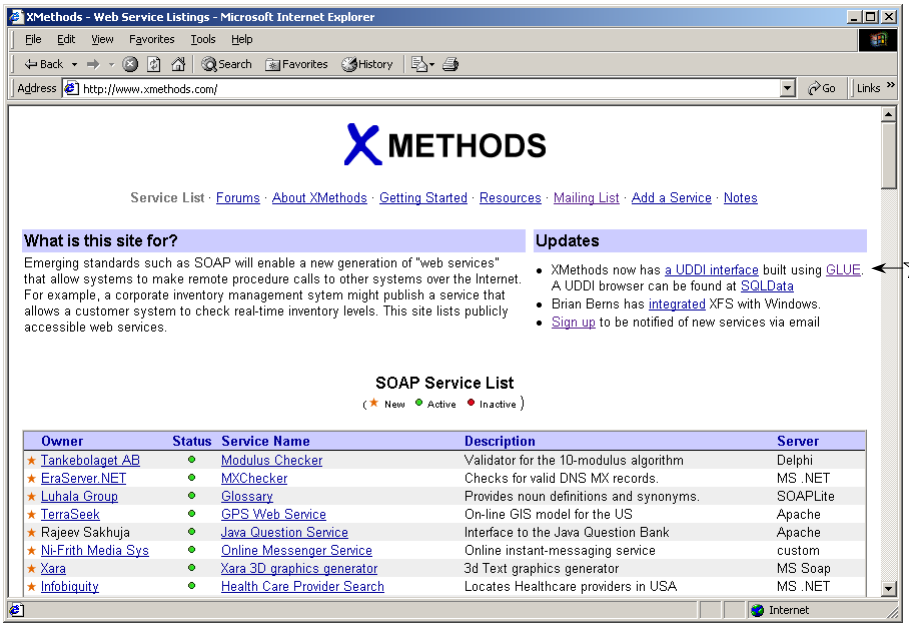
- From The Mind Electric
  - http://www.themindelectric.com
  - current version is 1.2
  - GLUE Professional is a commercial product
  - GLUE Standard is free and has a subset of the capabilities
  - one of the easiest toolkits to use
  - created by Graham Glass
    - CEO, chief architect and founder of The Mind Electric
    - has written many excellent articles about web services
      - see http://www-106.ibm.com/developerworks/webservices/ under "Columns"
    - his Prentice Hall book "Web Services: Building Blocks for Distributed Systems" will be out in late 2001
- Not an acronym
  - named after a common use of SOAP which is to create applications by gluing together pieces of distributed functionality

GLUE Toolkit Contents

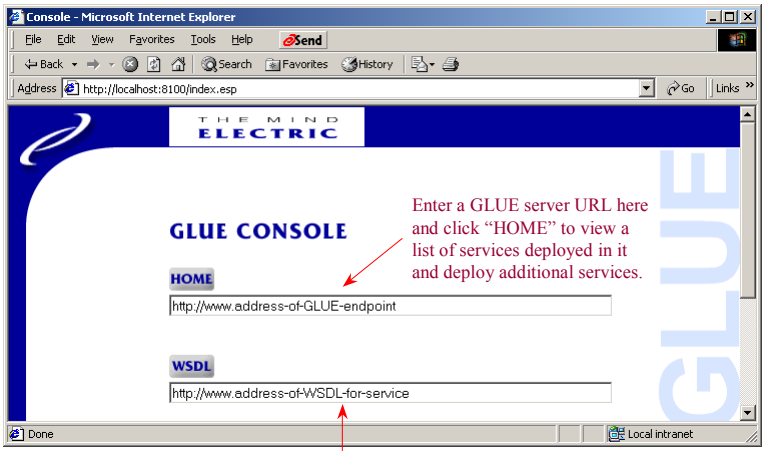
- A Java toolkit that provides
  - embedded web server with a servlet engine
  - SOAP processor
  - Electric XML parser
    - a free DOM-alternative that may be faster and use fewer resources
  - graphical console ← see examples of this later
  - dynamic WSDL generator
  - dynamic Java/XML mapping (similar to what JAXB provides)
  - UDDI client and server
  - WAP support
  - XML persistent storage system

Publishing and Invoking Web Services in GLUE

- Three ways to publish services
  - from console web interface
  - from command line using GLUEServer
  - from Java using Registry.publish
- Four ways to invoke services
  - from console web interface
    - see "GLUE Console Method Screen" page ahead
    - invoking services this way is mainly for testing
  - from command line using WSDL
  - from Java using WSDL
  - from Java using generated client stubs
    - see code example on "GLUE Console Java Screen" page ahead
    - produces most readable client code



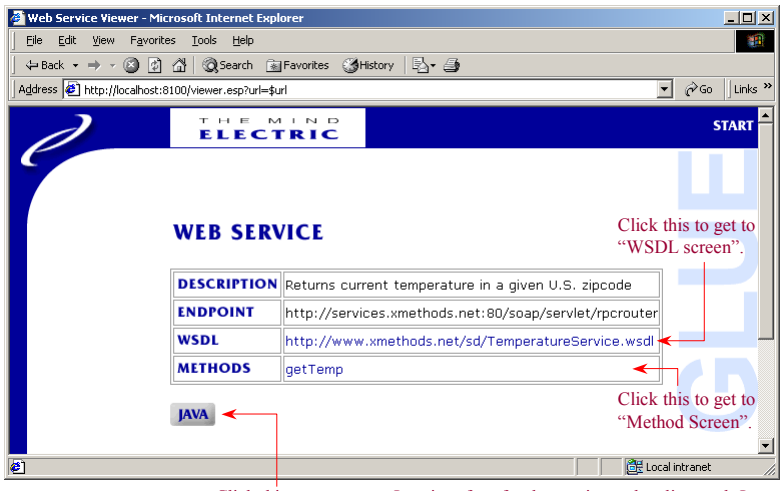
GLUE Console Opening Screen



Enter a GLUE server URL here and click "HOME" to view a list of services deployed in it and deploy additional services.

Enter a WSDL URL here and click "WSDL" to view information about the services described in it on the "Service Screen".

GLUE Console Service Screen

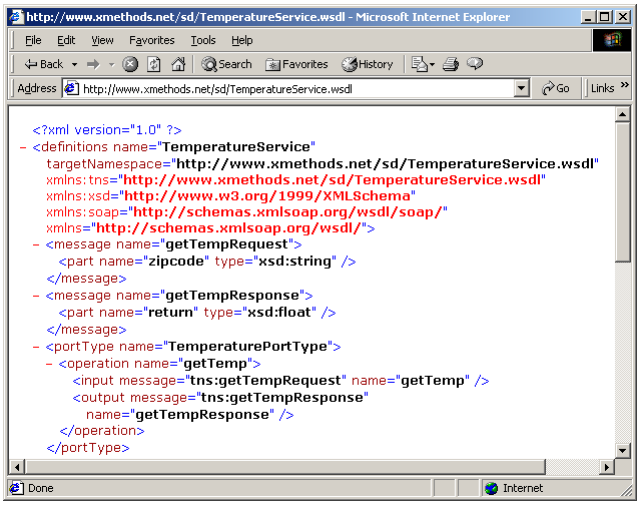


Click this to get to "WSDL screen".

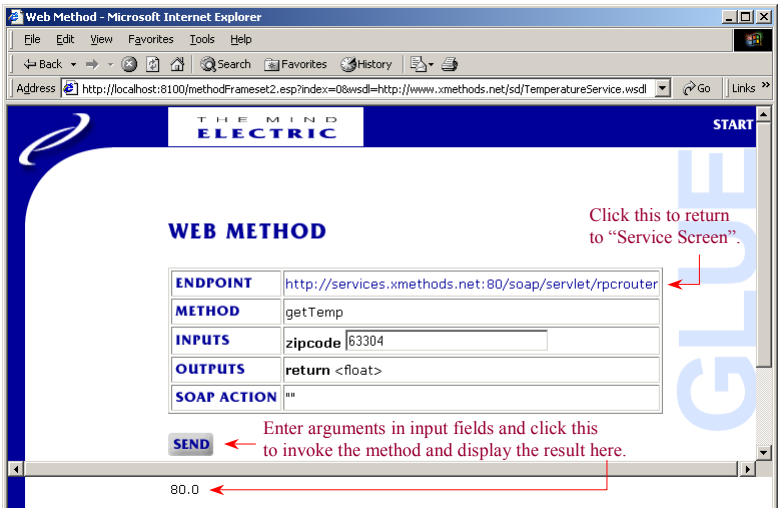
Click this to get to "Method Screen".

Click this to generate a Java interface for the service and a client stub Java class on the "Java Screen". To use them, copy the code into a text editor and save.

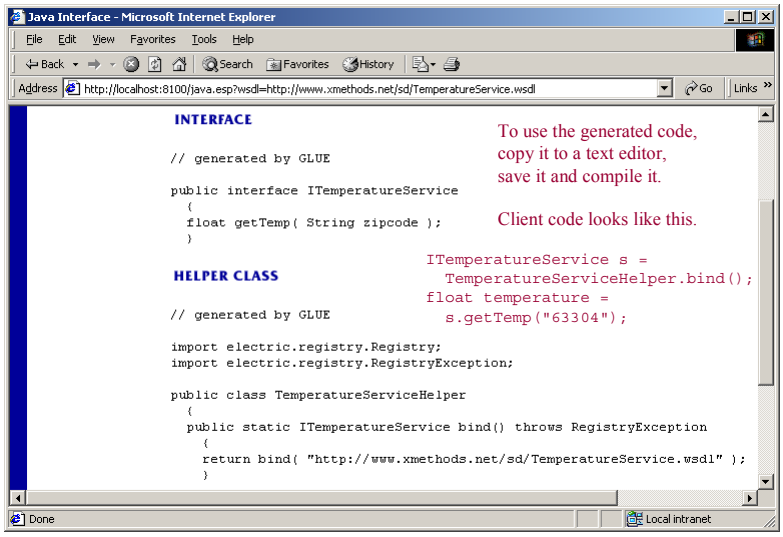
GLUE Console WSDL Screen



GLUE Console Method Screen



GLUE Console Java Screen



SOAP and Web Services

- Provides a working knowledge of standards that comprise "web services"
- Duration: 3 days
- Prerequisites
  - Java Programming course or equivalent Java experience
  - eXtensible Markup Language course or equivalent XML experience
- Topics (tentative)
  - Overview of Web Services
  - Simple Object Access Protocol (SOAP)
  - Web Services Description Language (WSDL)
  - Universal Description, Discovery and Integration (UDDI)
  - Electronic Business XML (ebXML)
  - Web Service toolkits (AXIS, WSTK, GLUE)
  - JUDDI - a Java API for accessing UDDI registries

For more information on this and other OCI courses, contact Jessica Hardin at 314-579-0066.