Towards Web Services

A long way to get here…

What is a Web Service?

- *Software service* : something that accepts (digital) requests and returns (digital) responses
- C-function, Java object, SQL-stored-procedure
- Web services: SW *components*
  - Language independent
  - Platform independent
  - Location independent
- Building blocks for distributed applications, possibly spread worldwide, based on XML
- Web services → web of services

What is a Web Service?

Definition:

- A Web service is a
  software system identified by a URI,
  whose public interfaces and bindings
  are **defined and described using XML**.
  Its definition can be discovered by other software systems.
  These systems may then interact with the Web Service in a manner prescribed by its definition,
  using **XML based messages**
  conveyed by **Internet protocols**.

Software Services in Procedural Languages

- Assembly subroutines, in a single memory space
- FORTRAN, COBOL etc. functions,
  orchestrated by the program control flow

Software Services & TCP/IP

- Support to transport protocols
  is provided at the Operating System level
- Data can be easily exchanged in real time
  between two computers on the net
- IP addressing is the mean to identify
  the partner computer in the communication
- Programs must directly handle
  the communication phases
- ***No location transparency is provided***

RPC (and RMI…)

- Invented to allow procedures/functions written in FORTRAN, C, etc. to call each other regardless of their actual geographical position
- This technology allows *a simple organization of different modules* spread over several computers on the net, to build up a complete distributed system
- A step forward: OO and RMI
- What about dealing with heterogeneous platforms?

XDR: eXternal Data Representation

- Capability to exchange data does not imply ability to understand the MEANING of what is actually exchanged
- Hint for a solution: try to agree ***on the coding*** of what it is transmitted
- XDR is a solution, proposed in RFC 1014 (1987) and in RFC 1832 (1995)

  "XDR is a standard for the description and encoding of data. It is useful for transferring data between different computer architectures, and has been used to communicate data between such diverse machines as the SUN WORKSTATION*, VAX*, IBM-PC*, and Cray*. XDR fits into the ISO presentation layer, and is roughly analogous in purpose to X.409, ISO Abstract Syntax Notation. The major difference between these two is that XDR uses implicit typing, while X.409 uses explicit typing.

Objects and Services

- OO languages focus on employing high levels of abstraction for creating “software objects” that mimic their real-world counterparts
- Object: a software service (through methods)
- The role of encapsulation and information hiding
- Actors
- Still standalone?

CORBA vs. DCOM

- How to let objects communicate across the net?
- New “protocols” (middleware) applying different solutions:
- CORBA and DCOM
- CORBA (Common Object Request Broker Architecture) has been supported by OMG
- DCOM (Distributed Common Object Model) by Microsoft
- Deployment in (homogeneous?) environments
- What about IIOP?

CORBA and DCOM

- Unfortunately,
  DCOM and CORBA aren’t compatible
- How to make CORBA and DCOM collaborate?
- Protocol converters, “gateways”
- Again, heterogeneity is not addressed
  in a straightforward manner

Web's Coming!

- Through the Web, information can easily viewed by means of a single tool, the "web browser"
- Everything is based on a set of simple standards:
- HTTP and HTML allow the access to remote info on web pages
- It has become a PERVASING technology!

HTTP and HTML

- HTTP is a simple, *stateless* protocol to exchange data (hypertext);
  it tells apart the role of client and server;
  it can be built upon the IP stack
- HTML is a simple language
- What was missing was
  a way to share data and software services
  across the Internet

XML Enters the Scene

- XML, eXtended Markup Language, allows many different kinds of data to be represented
- The structure of data is accurately described
- It does not rely on any particular standard coding of data at lower levels
- It does not provide any SEMANTICS!
(as HTML does)
- It can be regarded as a good, NEUTRAL way
to support data exchange and sharing

Sharing of Services

- The sharing of software services
  has then being addressed
  by developing a new set of standards called:

  SOAP

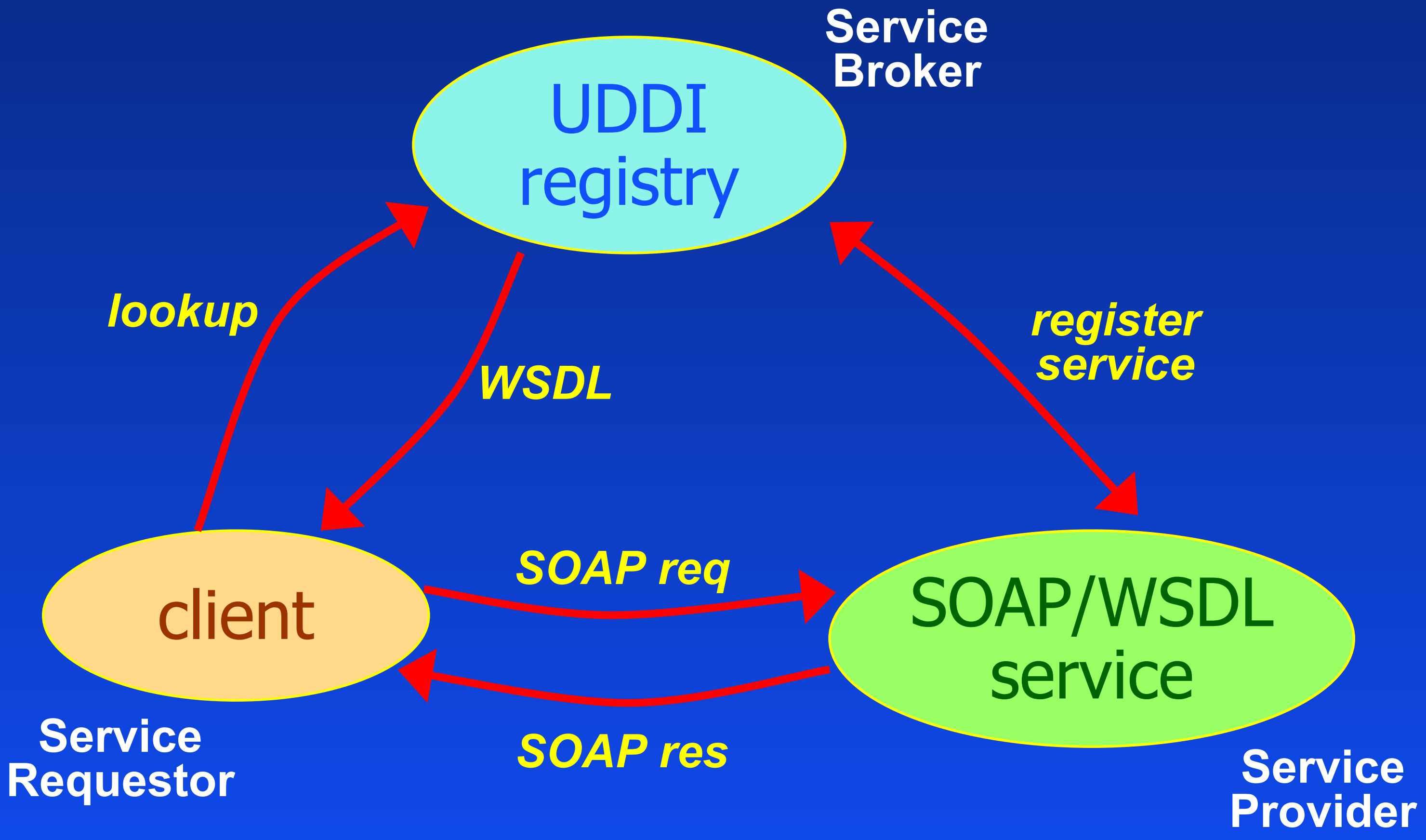
  WSDL

  UDDI
- They are built upon XML and (usually) HTTP
- They provide a way for services to be
  published, located and invoked:
  language, platform and location independently

WS Architecture: Components

- **Service provider**
  - Delivers services across the network
  - Publishes services to a broker
- **Service requestor**
  - Asks the broker for a service
  - Binds to the provider once it is found
- **Service broker**
  - Matchmaker between providers and requestors

WS “platform”: Overview



WS Architecture: Operations

- **Publish/unpublish with a Service Broker**
  - Service providers may possibly advertise their services with a service broker
- **Find**
  - Service requestors ask the broker for a specific service that meets certain criteria
- **Bind**
  - Service requestors bind to the service providers, then transactions follow.

SOAP

- Officially: Simple Object Access Protocol
- A.k.a. "Service-Oriented Architecture Protocol"
- It is a general-purpose technology for sending messages between endpoints, and may be used for
  - Plain document transfer
  - RPC
- Key feature: SOAP messages are represented using XML (meta-info is present too)
- SOAP messages can be sent over any transport means (usually, HTTP)

SOAP characteristics

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

"Publishing"/Deploying a Web Service

- Publishing a sw component as a web service means to make it available for external clients
- The access to the web service is implemented via SOAP messages
- The easiest way to publish a web service is to use a SOAP container (a.k.a. SOAP router); the service will then be available to any SOAP-enabled client…

SOAP Container

- A SOAP container:
- Accepts incoming requests;
- Dispatches them to published (deployed) components;
- Automatically translates between SOAP and the component’s native language interface
- SOAP containers are available for most programming languages: Java, C++, Perl, C#, Pyton, VB, …

- Basic issue: what’s the info a SOAP client must know to exploit a specific web service?

Info for SOAP Clients

- Any SOAP-enabled client must know:
- The network address of the web service (container address + WS ID)
- The messages that it can understand
- Once this info is known to the client, it can send a SOAP request and get back a SOAP response

- To get the required info, SOAP clients read a file that describes the web service: the language used for this purpose is WSDL
- Usually, SOAP containers are able to automatically generate WSDL files for the hosted services (recall Javadoc…)

WSDL: Resumes for Software Services

- WSDL stands for Web Services Description Language
- A WSLD file is a sort of resume for a web service:
- It describes
  - what the WS can do,
  - where it is placed,
  - how to invoke it
- We can compare its role to that of IDL in CORBA systems…
- WSDL makes use of XML (again!)
- Why WSDL is important?

“Labor Market” for Web Services

- Web services are seen as the basic components (spread over the net) of whatever distributed application
- An application could choose to employ one out different equivalent components,
  as long as they are carefully described
- The correct exploitation of WS strictly depends on the proper description for them!
- This story is a kind of metaphor
  of the human labor market…
- Basic issue:
  how a client may know of existing WSs?

Discovery: Approach #1

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

- Mmmmhhhh….

Discovery: Approach #2

- I write a client application that discovers a service in a specialized registry.
- I invoke that service, sending it vital data about my company.
- I ***discover*** what the service does.

- Mmmmhhhh….

Discovery: Approach #3

- You discover my service in a specialized (UDDI) registry.
- You invoke my service, sending me your name and credit card number.
- You ***discover*** what I do with that information.

- Me: AH!
- You: Mmmmhhhh….

Discovery: Approach #4

- Your client application looks for ***a particular service*** in a particular file.
- Your client gets the WSDL file referenced there.
- You invoke the service.
- AH!

Discovery: Approach #5

- Your client application searches for ***a particular kind of service*** (a "tModel") in a UDDI registry.
- Your client uses categories to select one of the providers that has the service you need.
- You invoke the service.

- AH!

UDDI: Let’s Go on with the Hype!

- UDDI is a standard that allows information about businesses and services to be electronically published and queried
- UDDI stands for Universal Description, Discovery and Integration
- An UDDI registry is used to store published info
- An UDDI registry is usually accessible both through a web browser and through SOAP
- An UDDI registry is accessed by companies (businesses) to let know about the availability of their WSs, and by clients to search for adequate WSs

UDDI Informs about WSDL

- The info on a WS at an UDDI repository includes:
  - its enpoint address
  - A reference to its WSDL
- A distributed application can search the UDDI registry, and find out the required services at runtime. Then, by the gathered description in the WSDLs, it can create SOAP clients to invoke them.
- UDDI registries can be either public or "private", depending on who is authorized to access them.
- Public UDDI registries are hosted e.g. by IBM, Microsoft and HP; their content is periodically synchronized

…and What about Performance?

- Interoperability vs. Performance:
- Binary vs. XML encoding
- Absence vs. Presence of meta-information
- Lightweight vs. Heavyweight marshalling/unmarshalling
- At the end, most depends on the performance of the underlying communication protocols
- In a close environment, optimizations can be made: the wider, the slower!
- On a standalone PC: 500 SOAP msg/sec
- On a fast LAN: 300 SOAP msg/sec

Challenges

- Reliability – any guarantee on QoS?
- Security – What granularity? Authentication?
- Discovery – What future for UDDI?
- Transaction management – Long-lasting transactions… is 2-phase commit still fitting?
- Scalability – Need for ad-hoc Application Servers?
- Manageability – What does it means to manage such a kind of distributed application?
- Testing – The internals of most of the components are not directly accessible…