Web Services - Concepts, Architecture and Applications
Part 1: Introduction to Web services

Gustavo Alonso and Cesare Pautasso
Computer Science Department
ETH Zürich
alonso@inf.ethz.ch
http://www.inf.ethz.ch/~alonso
Motivation for the course

Overview of technology

- Web services have not appeared out of the blue but are the result of the natural evolution of middleware and enterprise application integration platforms as they try to leverage the WWW, the Internet and the globalization of society as a whole, particularly in its economic aspects.

- A key to understanding Web services, how they are and how they might evolve is understanding how we got there and what the relation of Web services with existing technology is. This relation is inescapable as only from this perspective is it possible to understand what is happening in the Web services world.

Web services

- Web services have captured the attention of many companies and vendors. We are currently witnessing a flurry of activity almost without precedent in the computer industry. New proposals appear every month, many of them never to be heard of again.

- The nature of Web services and the motivation to use them is often blurred by hype as well as the many contradictory and overlapping proposals and specifications out there.

- The questions we will try to answer in this course are: what are Web services? What can be done with Web services today?
Objectives of the course

Understanding Web services

- In order to understand Web services, we need to take a step back and look at the way middleware and enterprise application integration technology has been evolving in the last decades. Only so we will then be able to understand Web services.

- The most popular version of Web services (SOAP, UDDI, and WSDL) is a very poor and limiting view on what true Web services should be. Alternative proposals such as ebXML, xCBL, or RosettaNet provide a much deeper insight on what is needed for electronic commerce through Web services.

Putting Web services into perspective

- We will go over the basic Web service technology available today and discuss SOAP, UDDI and WSDL in detail.

- We will also discuss what can be done with these specifications and what is missing nowadays for them to be truly useful tools for electronic commerce.

- The course aims at developing a critical understanding of Web service technology and its possibilities today. The goal is for participants to be able to look at current and future developments with enough background to be able to judge how much of a contribution they are and what their true potential is.
Two competing views of Web services
bottom up: application integration
top down: business exchanges
A popular interpretation of Web services is based on IBM’s Web service architecture based on three elements:

- Service requester: The potential user of a service
- Service provider: The entity that implements the service and offers to carry it out on behalf of the requester
- Service registry: A place where available services are listed and which allows providers to advertise their services and requesters to query for services

The goal is just-in-time integration of applications by discovering and orchestrating network-available services
Why is it bottom up?

- The Web service architecture proposed by IBM is based on two key concepts:
  - architecture of existing synchronous middleware platforms
  - current specifications of SOAP, UDDI and WSDL
- The architecture has a remarkable client/server flavor
- It reflects only what can be done with
  - SOAP (Simple Object Access Protocol)
  - UDDI (Universal Description and Discovery Protocol)
  - WSDL (Web Services Description Language)
ebXML architecture

1. Request Business Details
2. Build Local System Implementation
3. Register Implementation Details
4. Query about COMPANY A Profile
5. Agree on Business Arrangement
6. DO BUSINESS TRANSACTIONS

COMPANY A

COMPANY B

ebXML compliant system

©2000 ebXML TM
ebXML architecture: electronic commerce
Why is it top down?

- The ebXML (electronic business XML) architecture tries to describe all elements and steps of a commercial exchange, including the characteristics of the technology that supports those exchanges. Its goals are an attempt at:
  - Providing a view for integration of business processes among ad-hoc or established independent business partners by electronic means
  - Reducing the need for collaborative business partners to have individual and expensive prior agreement on how to integrate business processes
  - Providing a high-level business-centric view of distributed e-business processes
  - Supporting and representing business processes independent of the technical solution
  - Providing and supporting a library of common, standard intra-business processes
  - Allowing for both business processes and enabling technologies to evolve independently while retaining long-term investments in both
  - Integrating with new and legacy systems throughout the enterprise
  - Leveraging existing technologies and standards
Background for these views

- The IBM’s Web service architecture follows traditional middleware architecture and it is very tied in spirit to conventional middleware.

- It interprets electronic commerce as an essentially synchronous activity where a client requests services (in the middleware sense) from a server. If necessary, a name and directory service infrastructure (the registry) is available for matchmaking between clients and servers in this interpretation.

- This view of Web services focuses on the low level mechanisms necessary to implement Web services. It does not say anything about how these mechanisms map to real business exchanges.

- The ebXML electronic business architecture predates the efforts around Web services. It sees electronic commerce as an asynchronous sequence of message exchanges.

- ebXML has inherited many ideas from the EDI (Electronic Data Interchange) world as one of the main supporters of ebXML is the same organization that is in charge of EDI. As such, its main interest is standardizing business processes and business operations.

- ebXML contains a much richer vocabulary and deeper understanding of business relations that uses to establish what the low level technology should do. As a result, it is far more encompassing than IBM’s Web service architecture.
How did we get there: the evolution of the architecture of information systems
Basic concepts and notation

- **Client** is any user or program that wants to perform an operation over the system. To support a client, the system needs to have a **presentation layer** through which the user can submit operations and obtain a result.

- The **application logic** establishes what operations can be performed over the system and how they take place. It takes care of enforcing the business rules and establishing the business processes. The application logic can be expressed and implemented in many different ways: constraints, business processes, server with encoded logic ...

- The **resource manager** deals with the organization (storage, indexing and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence.
Distribution at the different layers

- Support for multiple clients
- Separated application logic
- Presentation logic
- Application Logic
- Resource Manager
- Data distribution or replication
- Any combination thereof
A game of boxes and arrows

- Each box represents a part of the system. Each arrow is a connection between two parts of the system.
- The more boxes, the more modular the system: more opportunities for distribution and parallelism. This allows encapsulation, component based design, reuse.
- The more boxes, the more arrows: more sessions (connections) need to be maintained, more coordination is necessary. The system becomes more complex to monitor and manage.
- The more boxes, the greater the number of context switches and intermediate steps to go through before one gets to the data. Performance suffers considerably.
- System designers try to balance the capacity of the computers involved and the advantages and disadvantages of the different architectures.

There is no problem in system design that cannot be solved by adding a level of indirection.
There is no performance problem that cannot be solved by removing a level of indirection.
Architectures (1): fully centralized

- The presentation layer, application logic and resource manager are built as a monolithic entity.
- Users/programs access the system through display terminals but what is displayed and how it appears is controlled by the server. (These are the “dumb” terminals).
- This was the typical architecture of mainframe applications, offering several advantages:
  - no forced context switches in the control flow (everything happens within the system),
  - all is centralized; managing and controlling resources is easier,
  - the design can be highly optimized by blurring the separation between layers.
As computers became more powerful, the presentation layer moves to the client. This has several advantages:

- Clients are independent of each other: one could have several presentation layers depending on what each client wants to do.
- One can take advantage of the computing power at the client machine to have more sophisticated presentation layers. This also saves computer resources at the server machine.
- It introduces the concept of API (Application Program Interface). An interface to invoke the system from the outside. It also allows to think about federating these systems by linking several of them.
- The resource manager only sees one client: the application logic. This greatly helps with performance since there are no connections/sessions to maintain.
Technical aspects of the 2-tier architecture

- There are clear technical advantages: work within the server takes place within one scope (almost as in 1-tier), the design is tighter and can be optimized, less race conditions to worry about (still easy to manage and control), etc.

- However, not all are advantages:
  - The system has to deal with all possible connections. The maximum number of clients is given by the number of connections supported by the server.
  - Clients are “tied” to the system since there is no standard presentation layer. If one wants to connect to two systems, then the client needs two presentation layers.
  - There is no failure or load encapsulation. If the system fails, nobody can work. Similarly, the load created by a client will directly affect the work of others since they are all competing for the same resources.
  - The design of the application logic and the resource manager is tightly coupled, making it very difficult to change or separate while making it more efficient.
  - The design is complex and difficult to port to other platforms.
The client is always right

- The underlying systems are probably different. The complexity of dealing with two heterogeneous systems needs to be addressed by the client.
- The client becomes responsible for knowing where things are, how to get to them, and how to ensure consistency!
- This is tremendously inefficient from all points of view (very fat clients are not a solution).
- There is very little that can be done to solve this problems if staying within the 2-tier model. It can be solved by adding a level of indirection: **MIDDLEWARE**

- Clients end up wanting to access two or more systems. With a 2-tier architecture, this creates several problems:
  - the underlying systems don’t know about each other; there is no common business logic. If it is necessary, it needs to be implemented at the client.
Middleware is just a level of indirection between clients and other layers of the system. It introduces an additional layer of business logic encompassing all underlying systems. By doing this, a middleware system:

- simplifies the design of the clients by reducing the number of interfaces,
- provides transparent access to the underlying systems,
- acts as the platform for inter-system functionality and high level application logic, and
- takes care of locating resources, accessing them, and gathering results.
In a 3-tier system, the three layers are fully separated.

For some people, a middleware based system is a 3-tier architecture. This is a bit oversimplified but conceptually correct since the underlying systems can be treated as black boxes. In fact, 3-tier makes only sense in the context of middleware systems (otherwise the client has the same problems as in a 2-tier system!).

We will see examples of this architecture when concrete middleware systems are discussed.

A 3-tier systems has the same advantages as a middleware system and also its disadvantages.

In practice, things are not as simple as they seem … there are several hidden layers that are not necessarily trivial: the wrappers.
3-tier middleware architecture

Yearly balance?
Monthly average revenue?

Front end

TP-Monitor environment

Control (load balancing, cc and rec., distribution, scheduling, priorities, monitoring)

recoverable queue

app server 3

wrappers

Branch 1

Branch 2

Finance Dept.
Message oriented middleware (EAI)

**MESSAGE BROKERING**

**SIMPLE MESSAGING**

- Input queue
- Output queue
- External application

- Message filter: apply rules, transform syntax

- Input queue
- Application

- Input queue
- External application

- Input queue
- Application
Extending information systems to communicate through the Internet
The service world

- The WWW suddenly opened up software systems that had remained hidden within the IT organization of a company.
- The nature of the interaction did not change. Behind the WWW there is the same client/server model as in basic RPC. However, the WWW made everything much easier, cheaper and efficient:
  - integration at the level of user interface became possible
  - services could be accessed from anywhere in the world
  - the clients could now be not just an internal or selected user but anybody with a browser
Remote clients

- Remote clients
- User defined application logic
- Database
- Resource manager
- Database management system
- Client
- Stored procedure
- Embedded SQL
- API
- WEB SERVER
- FIREWALL
- Browser
- XML
Business to Business (B2B)
The next step ...

- The next step in that progression leads immediately to the notion of Web services as considered in IBM’s Web service architecture.
  - The notion of service in the conventional middleware is now translated into the notion of Web service based on the access channel to that service (the service in fact can be a pre-existing middleware service, e.g., stored procedures in databases made available as Web services)
  - The only thing that changes from the middleware and enterprise application integration world is that a few details need to be changed so that they match the needs of exchanges through the Internet rather than a LAN:
    - XML as the data representation format
    - SOAP as a protocol wrapper to allow conventional communication protocols of middleware platforms to cross the Internet and firewalls (essentially turns invocations into document exchanges)
    - WSDL as the XML version of IDLs (plus a few other things)
    - UDDI as the WWW version of basic name and directory services
The message world
Documents rather than methods

- When exchanges occur as documents, the emphasis is no longer on the mechanisms for sending and receiving documents (there are many to choose from) but on the contents of the documents and how to interpret them.

- The arrival of documents is not a function or a method call. It is not even a service request as such but one element of a business process. Hence, in addition to specifying the format for the documents and the semantics for the contents of the documents, it is necessary to agree on what documents are needed to get things done, i.e., the business process that defines the interaction between the participants.

- The basic document exchange standard is the Electronic Data Interchange (EDI), a set of common data format standards developed in the late 1970s.
  - EDI is a collection of basic data elements (e.g., a price) that can be combined to form composite data elements, which are then combined into functionally related units called data segments (e.g., the complete description of an item), and finally grouped into transaction sets (minimal meaningful unit for document exchanges between companies, e.g., an invoice or a purchase order)
  - Additional standards (EDIFACT, X12) describe how to use transaction sets as part of concrete business processes.
Processes rather than service calls

“The xCBL 3.5 ChangeOrder document is a buyer-initiated document that can be used to change an existing Order already received and responded to by a seller. The document can be used to make changes to header level information, change line items, cancel line items, add line items, etc. Note that if an OrderResponse has not been received for a given Order, a ChangeOrder is not necessary (an Order with a purpose of “Replace” should be used). Similarly, if an entire order is to be cancelled (regardless of whether a response has been received or not) an Order with a purpose of “Cancellation” should be used.”

xCBL 3.5 Order Management Recommended Use, Version 1.0 November 19, 2001
The next step for message based electronic commerce is to take advantage of the Internet to improve electronic commerce: e.g., ebXML and the many attempts at producing an XML version of EDI. Note, these efforts also include linking to the standardization efforts around XML and Web services. However, the final goal is still the standardization of business exchanges rather than the standardization of the low level mechanisms used to implement those exchanges.

- With this interpretation, there is no client server flavor to the exchanges. Business interactions occur between trading partners.
- Middleware architectures are not obvious in these proposals. One assumes they are there to implement the business processes but they do not determine the specification of the exchanges.
- There is much concern with the semantics of business exchanges than with the syntax (unlike in the Web services architecture which is purely syntactic).
- Many concepts that are unknown in the Web service architecture play an important role in these architectures: role of the participant, business agreement, automatic enforcement of business agreements, conversations, business protocols, integration of business processes, etc.
Further reading

- IBM Web service architecture
- ebXML
  - http://www.ebxml.org/
- xCBL
  - http://www.xcbl.org/
- EDI
  - http://www.unece.org/trade/untdid/welcome.htm
  - http://www.xml-edifact.org/