

Lesson 1 – Introduction

Service Oriented Architectures Security

Module 1 - Basic technologies

Unit 1 – Introduction

Ernesto Damiani

Università di Milano

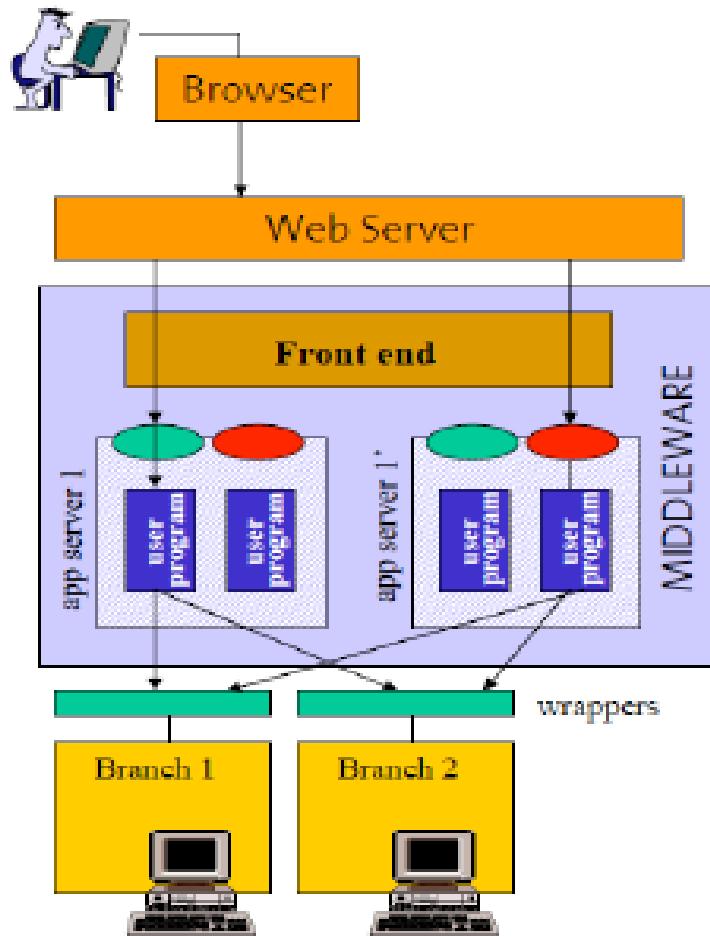
Course outline

- Web page: <http://ra.crema.unimi.it>
 - This is where all the lecture materials and additional pointers can be found
- Course syllabus (summary):
 - Web service basics
 - SOAP
 - WSDL
 - UDDI
 - Service Composition & BPEL
 - Messaging
 - Web 2.0 and REST
 - Mashups
 - SOA and integration architectures
 - Presentations from industry

Multi-tiered structure of Web applications (1)

- Multi-tiered architectures result from adding an additional layer to allow clients to access applications via a Web server
- The addition of the Web layer led to the notion of “application servers”, middleware platforms supporting access via the Web

Multi-tiered structure of Web applications (2)



HTTP parameter passing (1)

- The introduction of forms for allowing users to provide information to a web server required to modify HTML (and HTTP) but it provided a more advanced interface than just retrieving files

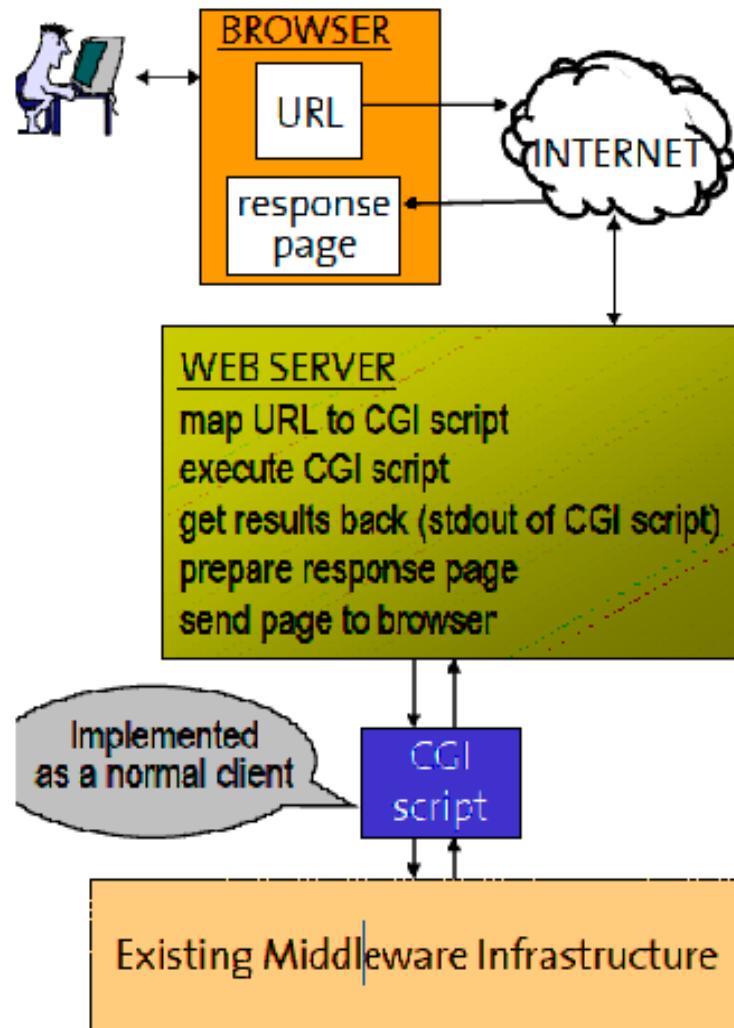
HTTP parameter passing (2)

- Exercise: explain the meaning of all lines of the following HTTP exchange
 - POST /cgi-bin/post-query HTTP/1.0
 - Accept: www/source
 - Accept: text/html
 - Accept: video/mpeg
 - Accept: image/jpeg
 - ...
 - Accept: application/postscript
 - User-Agent: Lynx/2.2 libwww/2.14
 - From: grobe@www.cc.ukans.edu
 - Content-type: application/x-www-form-urlencoded
 - Content-length: 150
 - * a blank line *
 - &name = E
 - &email= ernesto.damiani@unimi.it

CGI heritage (1)

- The earliest implementations of Web applications were built directly upon the existing systems.
- The CGI script (or program) acted as client in the traditional sense (for instance using RPC)
 - the user clicked in a given URL and the server invoked the corresponding script
 - the script executed, produced the results and passed them back to the server (usually as the address of a web page)
 - the server retrieved the page and send it to the browser

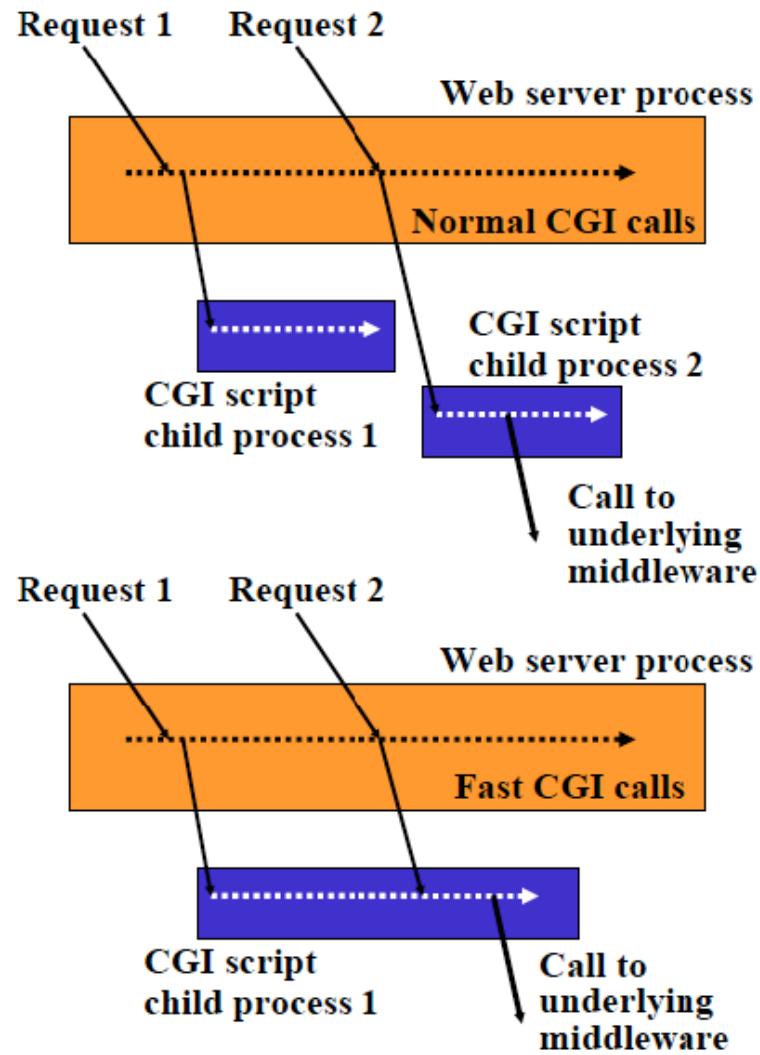
CGI heritage (2)



Why CGIs don't scale (1)

- CGI have several problems that are not easy to solve:
 - CGI scripts are separate processes, requiring additional context switches when a call is made (and thereby adding to the overall delay)
 - Fast-CGI allows calls to be made to a single running process but it still requires two context switches
 - CGI is really a quick hack not designed for performance, security, scalability, etc.

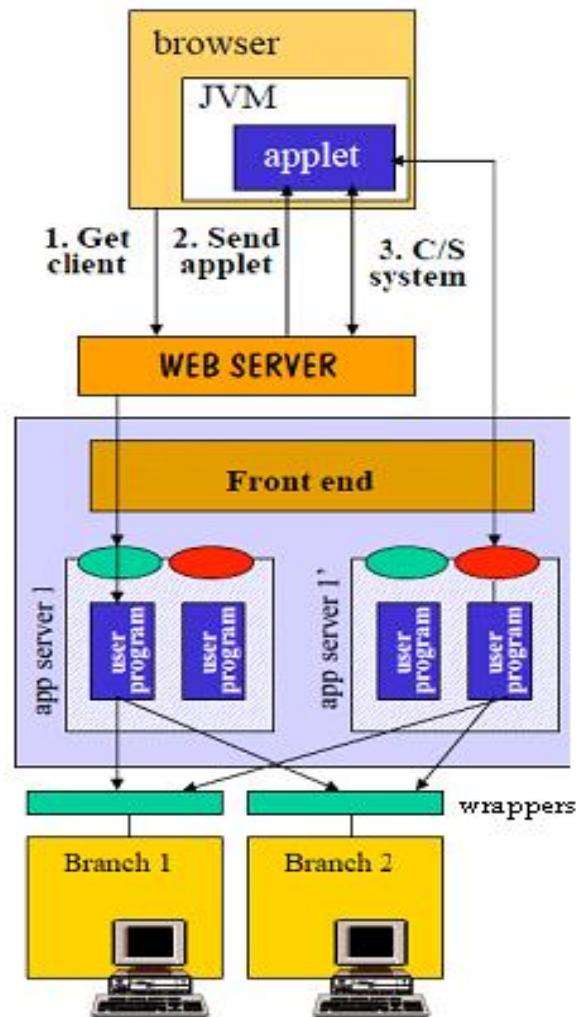
Why CGIs don't scale (2)



From thin to thick clients (1)

- A conventional web browser does not do much beyond displaying data
 - the processing power at the client is not used
- By adding a JVM (Java Virtual Machine) to the browser, now it becomes possible to dynamically download the client functionality (an applet) every time it is needed
 - The client becomes truly independent of the operating system and is always under the control of the server

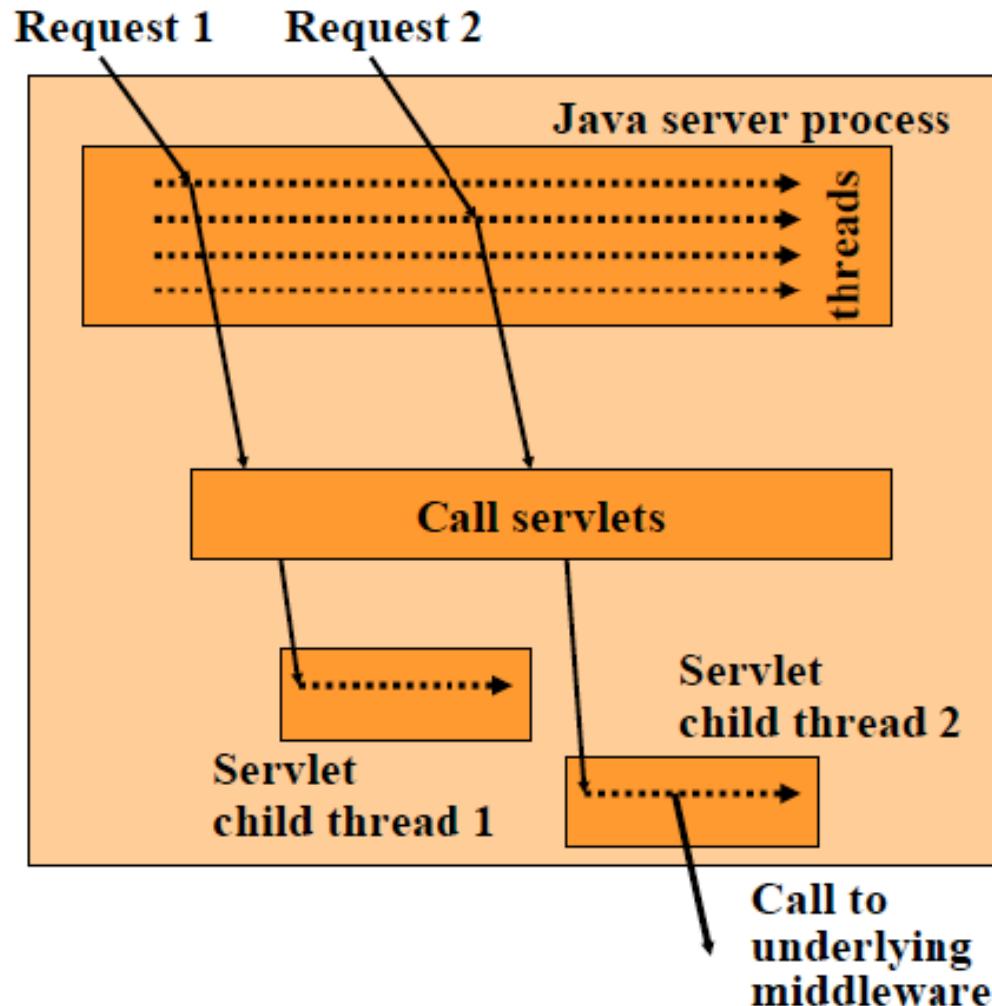
From thin to thick clients (2)



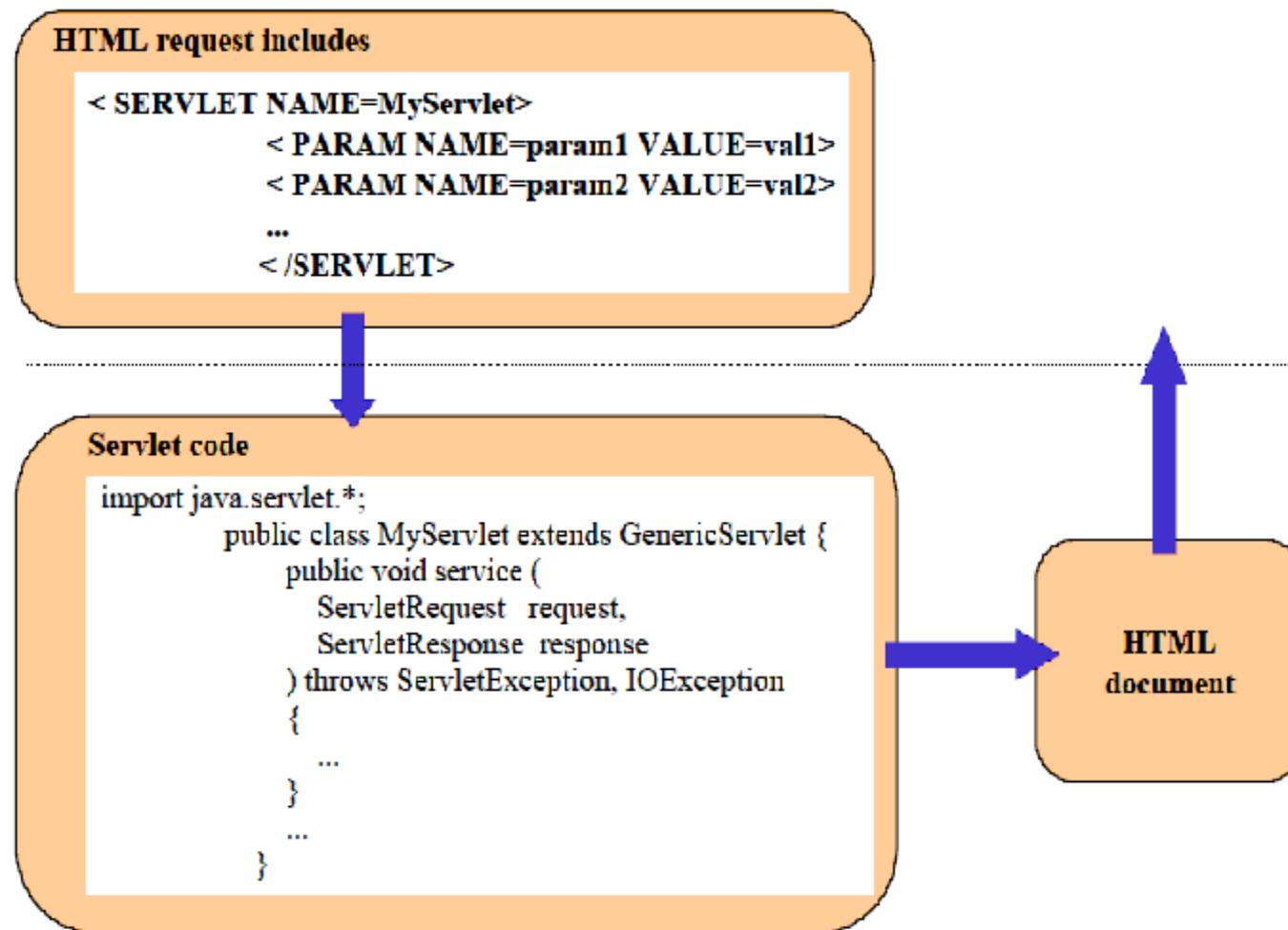
From CGI to servlets (1)

- Servlets have the same role as CGI scripts: they provide a way to invoke a program in response to an http request.
- BUT, servlets run as threads of the Java server process (not necessarily the web server) not as separate OS processes

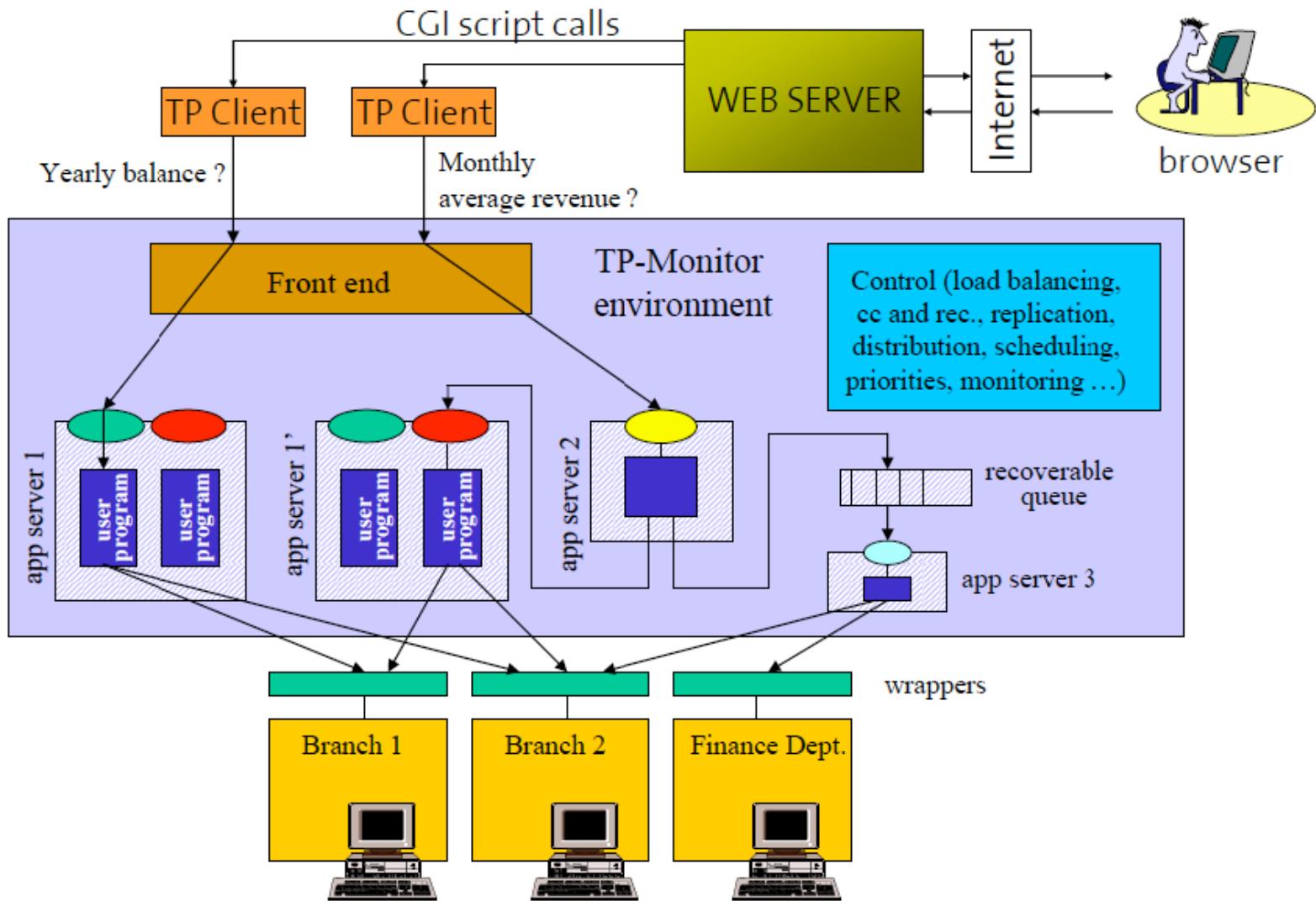
From CGI to servlets (2)



Launching a servlet



Web-based access to enterprise systems



A critical view of the business Web (1)

- HTTP was originally designed as a document exchange protocol (request a document, get the document, render/display it)
 - It is almost like e-mail (in fact, it uses RFC 822 compliant mail headers and MIME types)
 - The document format (HTML) was designed to cope with GUI problems

A critical view of the business Web (2)

- Interaction through document exchange can be very inefficient when the two sides of the interaction are programs (documents must be created, sent, parsed on arrival, etc.)
- The initial WWW model was biased towards the server
 - the client (the browser) does not do much beyond displaying the document
 - for complex applications, that meant much more traffic between client and server
 - high loads at the server as the number of users increases

From Web-mediated application access to B2B (1)

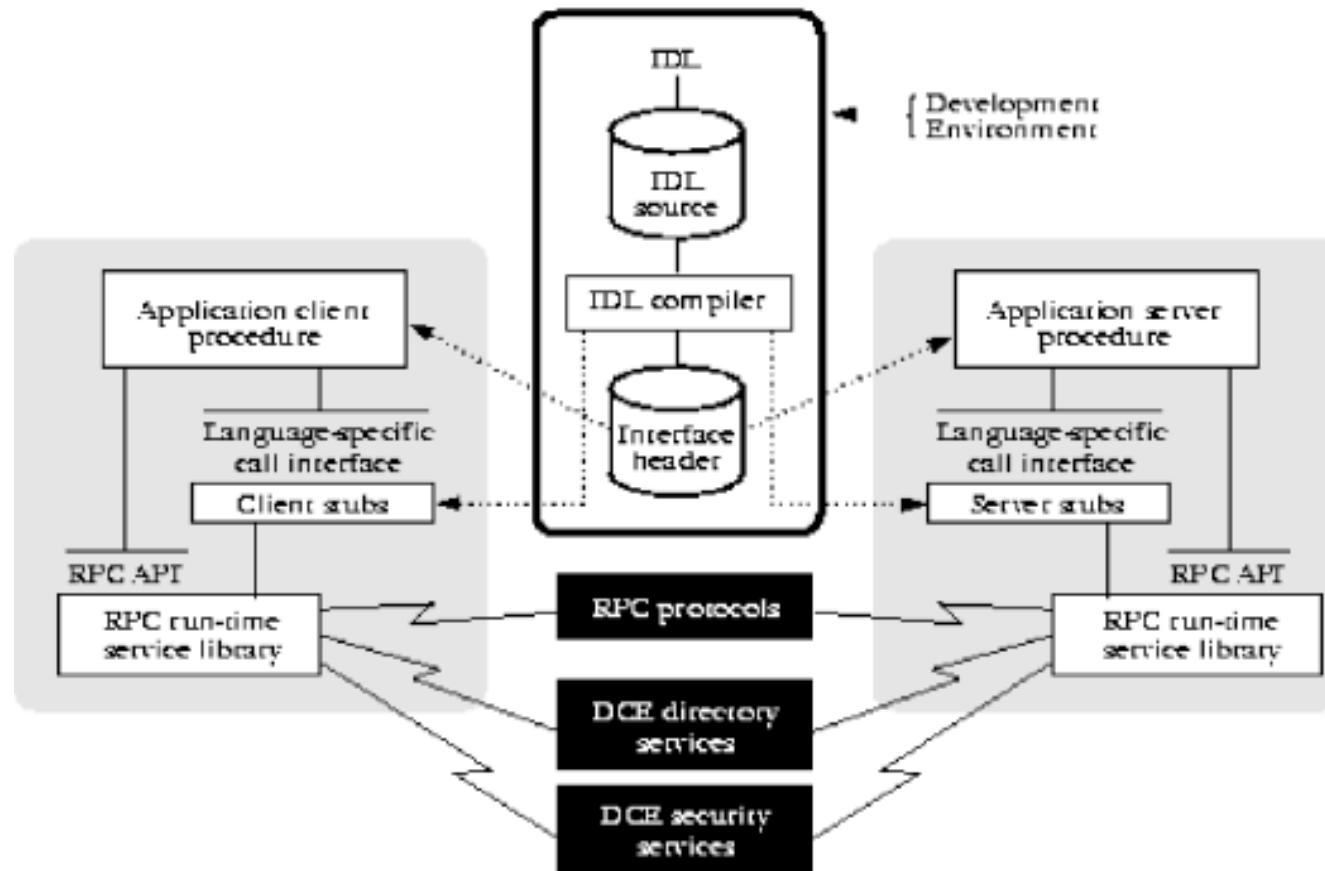
- The basic idea behind B2B follows the client/server model
- A service provided by one company can be directly invoked by a client running in another company

From Web-mediated application access to B2B (2)

- **Problems**

- joint development of client and server is not possible
- the server and client are likely to be hidden behind firewalls
- the interaction takes place among existing systems, it is not possible to assume the IT platforms will be uniform
- the Internet is cheap but open to everybody (unlike leased lines that are expensive but private)
 - Existing systems/protocols are not really designed for such type of interactions

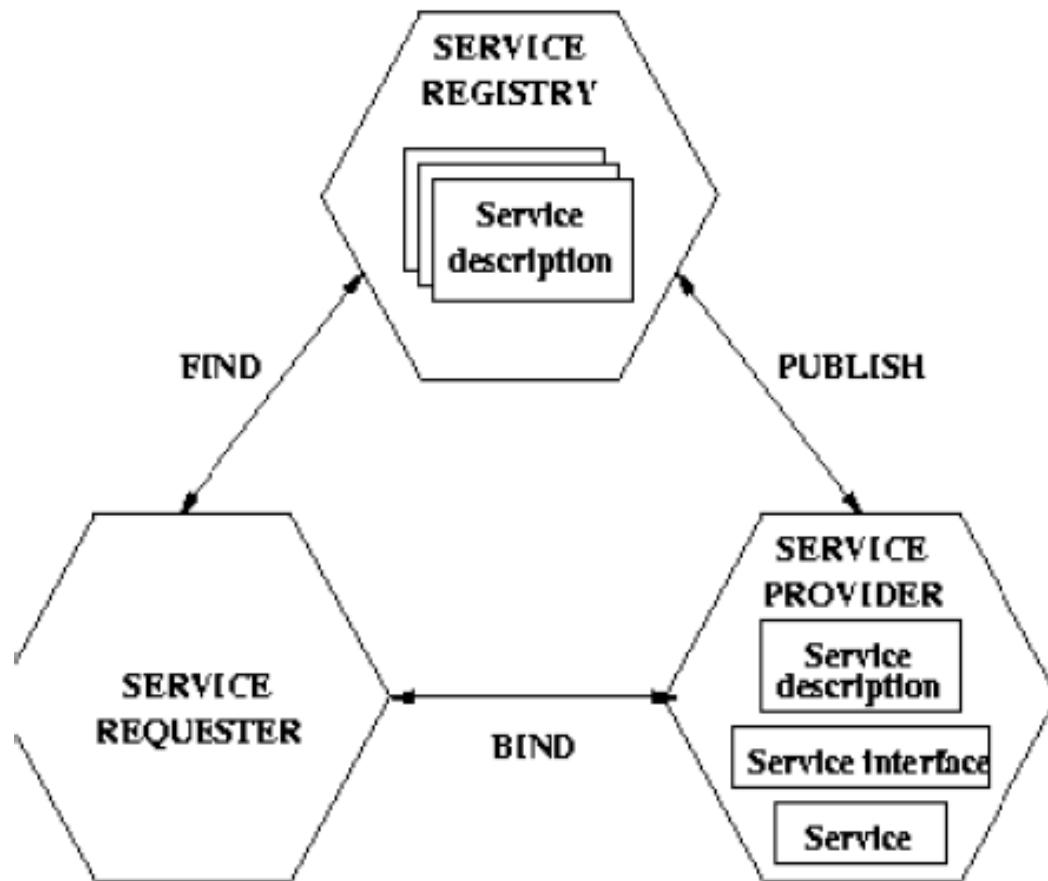
Predecessors: RPC remote calls



Web service architecture (1)

- IBM's Web service architecture composed of three elements:
 1. Service **requester**: the potential user of a service (the client)
 2. Service **provider**: the entity that implements the service and offers to carry it out on behalf of the requester (the server)
 3. Service **registry**: a place where available services are listed and that allows providers to advertise their services and requesters to lookup and query for services

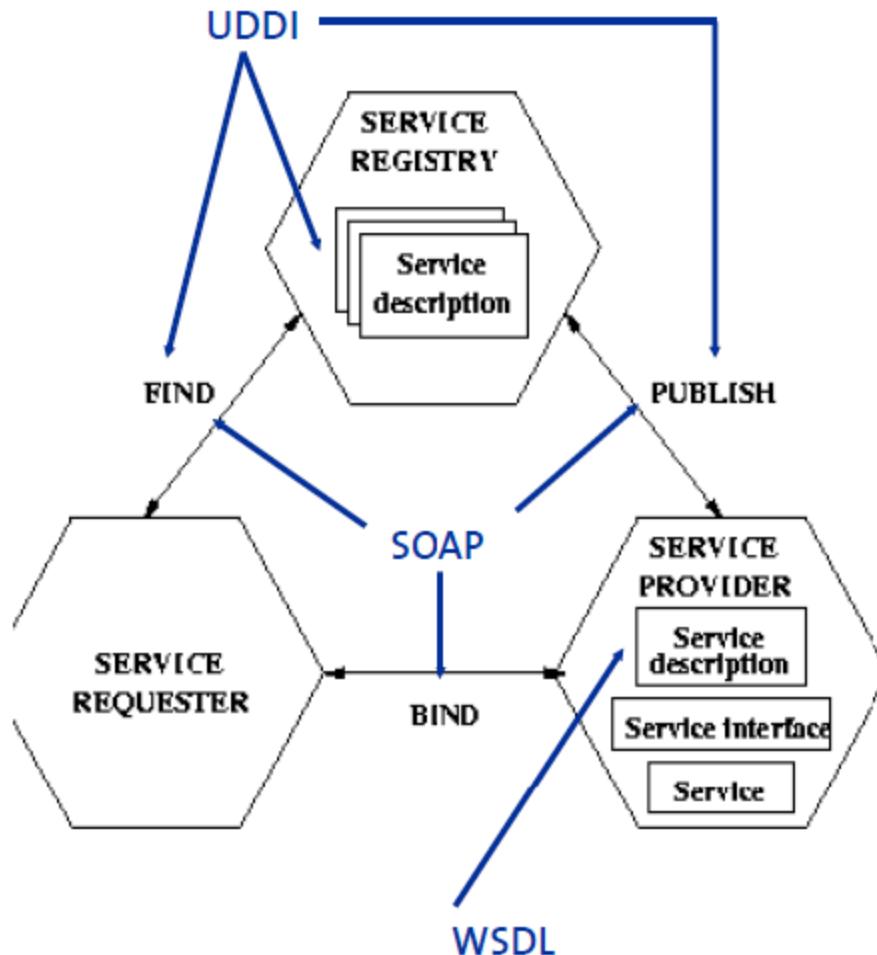
Web service architecture (2)



Main standards (1)

- 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
- It 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)

Main standards (2)



WS benefits

- One important difference with conventional middleware is related to the standardization efforts at the W3C that guarantee:
 - Platform independence (Hardware, Operating System)
 - Use of existing networking infrastructure (HTTP)
 - Programming language neutrality (.NET talks with Java)
 - Portability across middleware tools of different Vendors
- Web services are “loosely coupled” components that foster software reuse
- WS technologies are composable and can be adopted incrementally

WS standards

Transport	HTTP, IIOP, SMTP, JMS		
Messaging	XML, SOAP		WS-Addressing
Description	XML Schema, WSDL		WS-Policy, SSDL
Discovery	UDDI		WS-MetadataExchange
Choreography	WSCL	WSCI	WS-Coordination
Business Processes	WS-BPEL	BPML	WSCDL
Stateful Resources	WS-Resource Framework		
Transactions	WS-CAF	WS-Transactions WS-Business Activities	
Reliable Messaging	WS-Reliability		WS-ReliableMessaging
Security	WS-Security SAML, XACML		WS-Trust, WS-Privacy WS-SecureConversation
Event Notification	WS-Notification		WS-Eventing
Management	WSDM		WS-Management
Data Access	OGSA-DAI		SDO

- Web services are about Interoperability
 - Standardization
 - Integration across heterogeneous, distributed systems
 - Service Oriented Architectures are about:
 - Large scale software design
 - Software Engineering
 - Architecture of distributed systems
- SOA is possible but more difficult without Web services
 - SOA introduces some radical changes to software:
 - Language independence (what matters is the interface)
 - Event based interaction (no longer synchronous models)
 - Message based exchanges (no RPC)
 - Composition and orchestration

Dynamic Binding

- WS Invocation Framework
 - Use WSDL to describe a service
 - Use WSIF to let the system decide what to do when the service is invoked:
 - If the call is to a local EJB then do nothing
 - If the call is to a remote EJB then use RMI
 - If the call is to a queue then use JMS
 - If the call is to a remote Web service then use SOAP and XML
- There is a single interface description, the system decides on the binding
 - This type of functionality is at the core of the notion of Service Oriented Architecture

