HELSINKI UNIVERSITY OF TECHNOLOGY
Media Technology

AS-75.3600 Distributed Information Systems: Use Cases and Motivation

Tuukka Ruotsalo
Semantic Computing Research Group (SeCo)
Helsinki University of Technology (TKK),
Laboratory of Media Technology
and
University of Helsinki, Department of Computer Science
Slides partially based on (Alonso & Pautasso, 2004)

http://www.seco.hut.fi/

UNIVERSITY OF HELSINKI

SeCo
semantic computing

Motivation: basic concepts and economical aspects

Basic Concepts

- Architecture supports the evolution of information system life cycle: requirements, design, implementation and maintenance.
  - To understand components and services the life cycle of an IS (Information System) has to be analysed.
- Any architectural decision is based either on faster development or easier maintenance of the system
  - Reuse of the components (easier requirements engineering, design and implementation)
  - Modularity to support easier replacement of the components (easier maintenance)
- SOA (Service Oriented Architecture) is best suited to achieve dynamic control of the development process.
  - No re-designing and re-implementing the whole system whenever a change in the system environment occurs.
- Web Services are essential, but just one, technology to be used to implement SOA

Motivation

HELSINKI UNIVERSITY OF TECHNOLOGY
Media Technology

- The architecture of the information systems we use is becoming increasingly complex.
- The access methods, the capabilities, the goals, and the available technology is continuously changing. What can we learn that will remain valuable in the years to come?
- One example: 70 - 90 % of the software costs are maintenance costs.
- Using the right abstractions helps!! Databases used as services (e.g. JDBC+OR mapping) remove about 40 % of the code of commercial applications!
- Software reuse is truly efficient and make economic sense at a large granularity. How can we build systems that can be tailored to the user needs and yet are applicable in a wide range of areas and environments?

UNIVERSITY OF HELSINKI

SeCo
semantic computing

SOA (Service Oriented Architecture)

- SOAs build applications out of software services.
- Distributed computing and modular programming
- No mutual dependencies between the services
- Relatively large components
  - Reuse on “service level”
  - Changes in system requirements typically occur on service level
- Metadata is sufficient to describe not only the characteristics of services, but also the data that drives them
- XML is the current “state of the art” language for describing the metadata

Use Cases for SOA

- B to B systems
  - Changes in the company's strategy in networked business
    - » Replace a subcontractor with another one
  - Changes in internal business models
    - » In-source / outsource
    - » Re-organizing internal production processes
- B to C systems
  - Ubiquitous computing
- C to C systems
  - Virtual community portals

Web Services Usage Scenarios

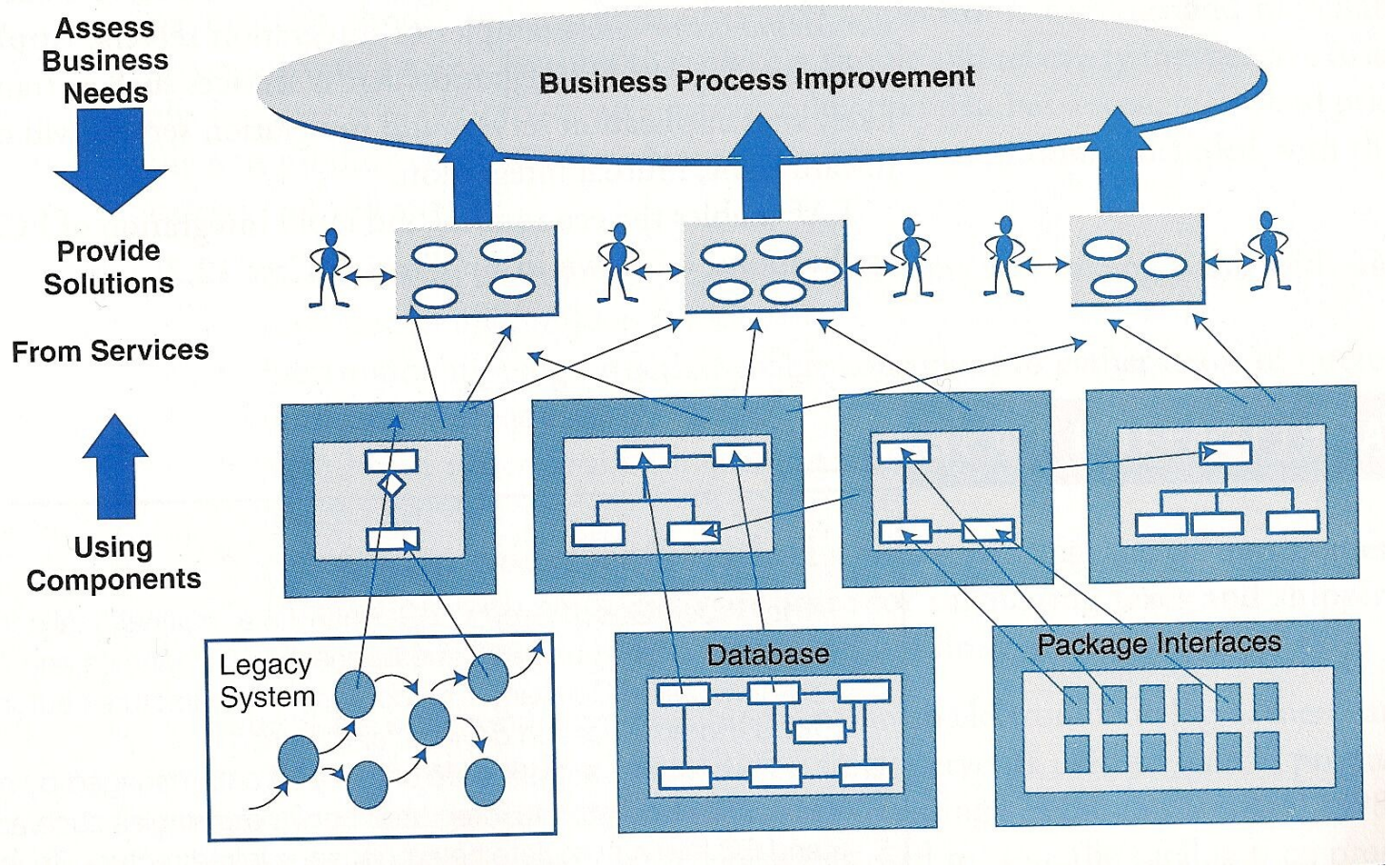
- B2B external: Purchasing electronic components from subcontractors
  - Partners are known
  - Primary needs: rigid data and process integration with as low changing costs as possible
- B2B internal: Reorganization of production in a workplace
  - Partners are known
  - Primary needs: process integration with fast configuration
- B2C: Print a document at airport
  - Partners are not known
  - Primary needs: service discovery, dynamic data and process integration
- C2C: Watch all photographs taken from the party last Saturday
  - Partners are not known
  - Primary needs: semantic data integration

Summary: data-, process- and semantic integration

- Data integration
  - Make two or more formal data models interoperable
- Process integration
  - Make two or more formal process models interoperable
- Goal: as low changing costs as possible
- Traditional approach
  - Programmer changes the code whenever there are changes in the requirements
- Configurable systems approach
  - Change configuration instead of code
- Semantic approach
  - Use rich data and let the computer plan, reason and decide the best solution (computer does the configuration)

Architectures supporting modularity

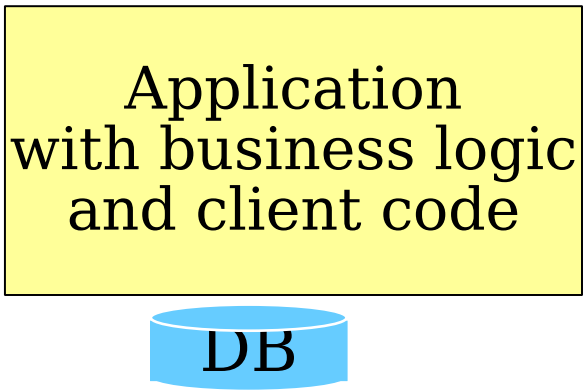
Service Oriented Design

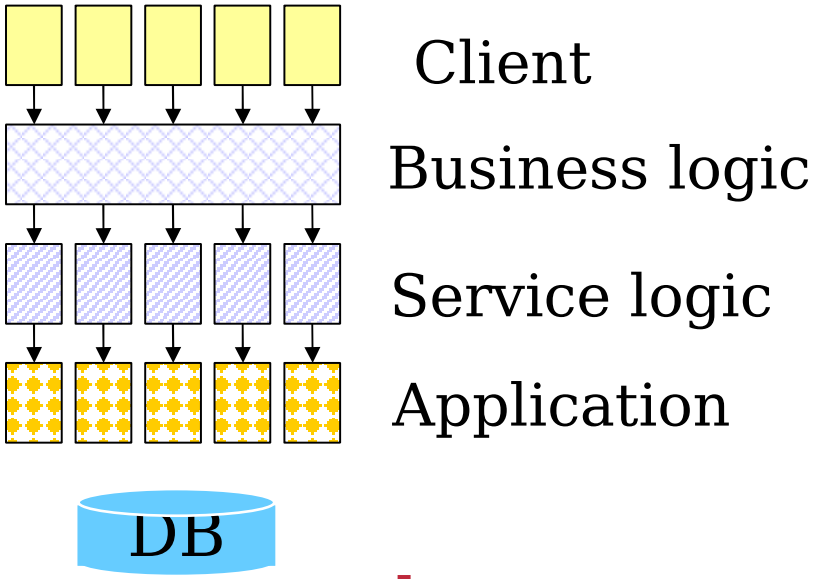


(Allen and Frost, 1998).

Consider a system of a shop that sells products and uses payment- and
in-house delivery services.

What happens if we want to replace the payment service provider,
make the shop available in mobile environment and outsource the delivery?





How did we get there: Architecture and Design Approaches of an IS

HELSINKI UNIVERSITY OF TECHNOLOGY
Media Technology

- Design of an Information System
  - Bottom-up: application integration
  - Top-down: business requirements
  - Meet-in-the-middle: applications are configured to meet the requirements
- Architecture of an Information System
  - 1,2,3,n tier architectures
  - Middleware

UNIVERSITY OF HELSINKI

SeCo
semantic computing

Top-down design

- The functionality of a system is divided among several modules.
- Modules cannot act as a separate component, their functionality depends on the functionality of other modules.
- Hardware is typically homogeneous and the system is designed to be distributed from the beginning.
- What you need is what you implement.

Bottom-up design

HELSINKI UNIVERSITY OF TECHNOLOGY
Media Technology

- In a bottom up design, many of the basic components already exist. These are stand alone systems which need to be integrated into new systems.
- The components do not necessarily cease to work as stand alone components. Often old applications continue running at the same time as new applications.
- This approach has a wide application because the underlying systems already exist and cannot be easily replaced.
- Much of the work and products in this area are related to middleware, the intermediate layer used to provide a common interface, bridge heterogeneity, and cope with distribution.
- What you need is what you compose from existing implementations.

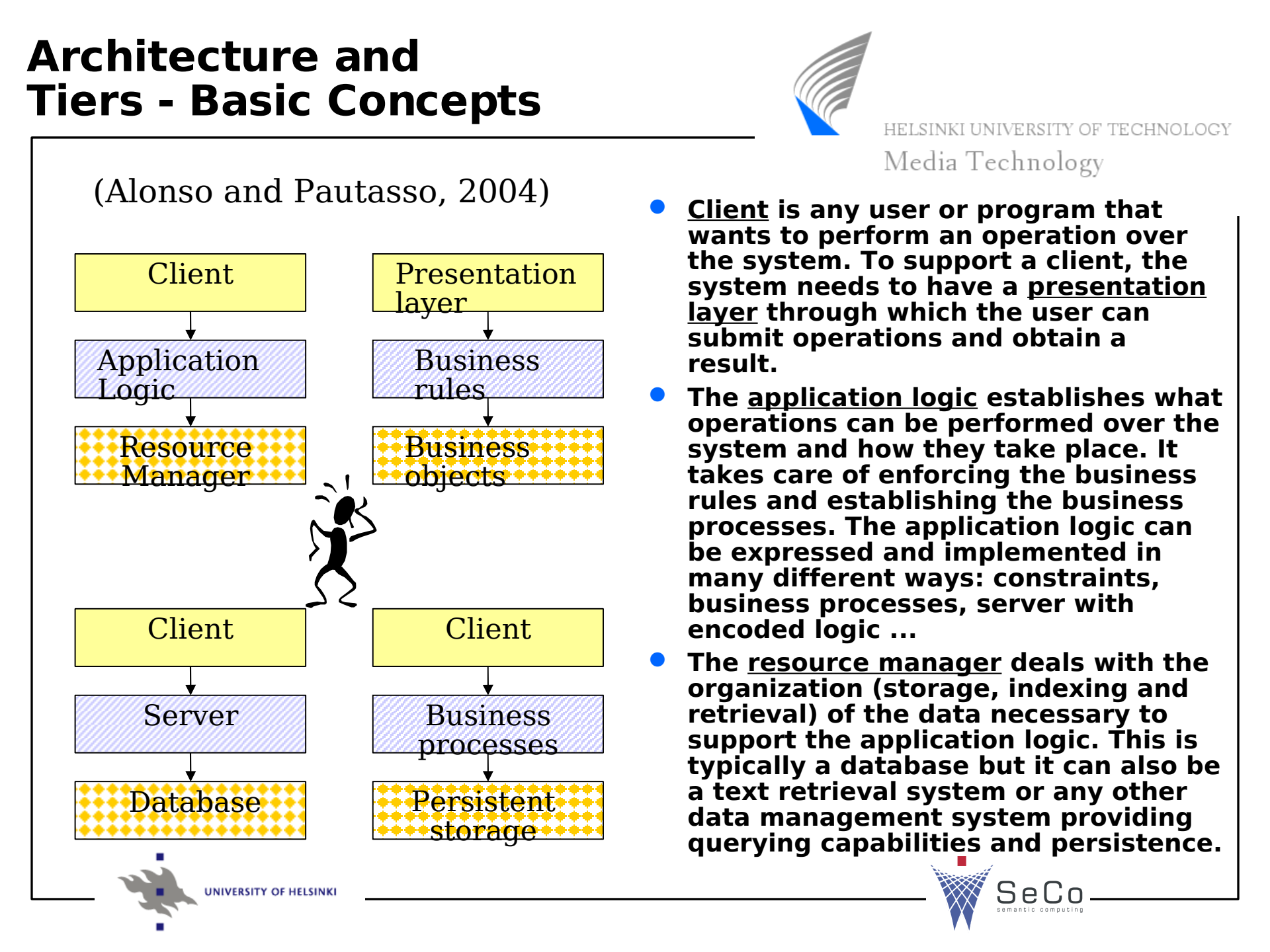
UNIVERSITY OF HELSINKI

SeCo
semantic computing

Meet-in-the-middle

- Usually systems can't be designed from the scratch. Several existing subsystems are needed and new subsystems have to be implemented.
- Example: How many different (inter-organizational) systems are needed to provide a complete web based shop?
  - Catalog interface, payment interface, transportation interface, ordering interface etc...
- Conclusion: We often need to use both: top-down and bottom-up design methodologies
- What you need is the new implementations and existing services working together

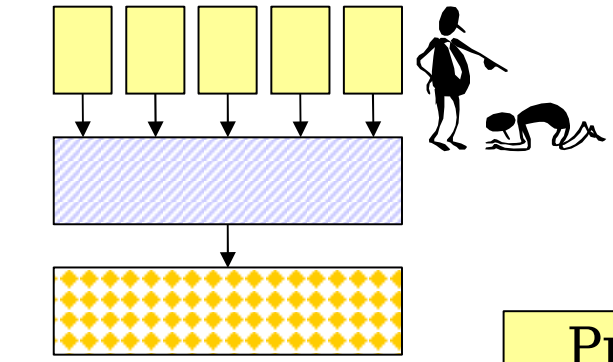
Architecture and Tiers - Basic Concepts



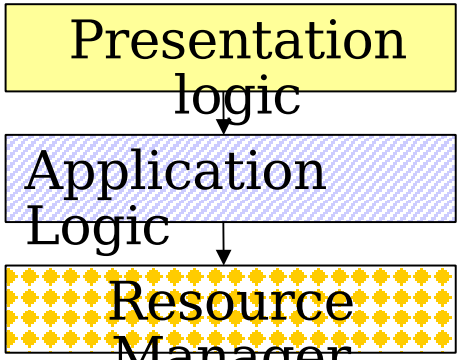
(Alonso and Pautasso, 2004)

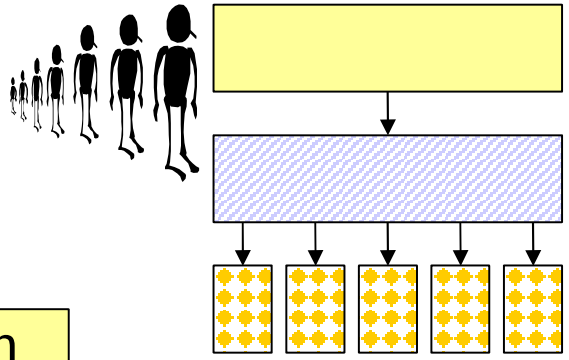
- **Client is any user or program that wants to perform an operation over the system. To support a client, the system needs to have a presentation layer through which the user can submit operations and obtain a result.**
- **The application logic establishes what operations can be performed over the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

Tiers and Layers



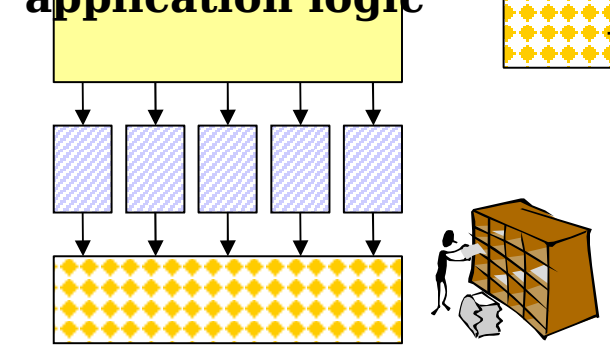
Support for multiple clients



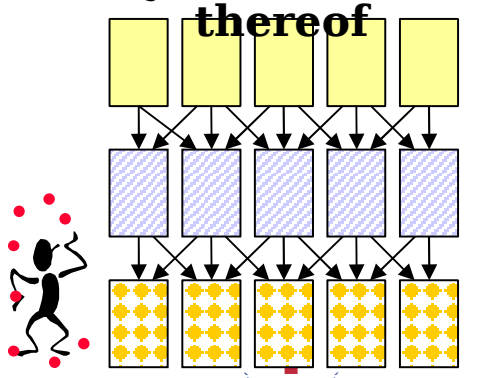


Data distribution or replication

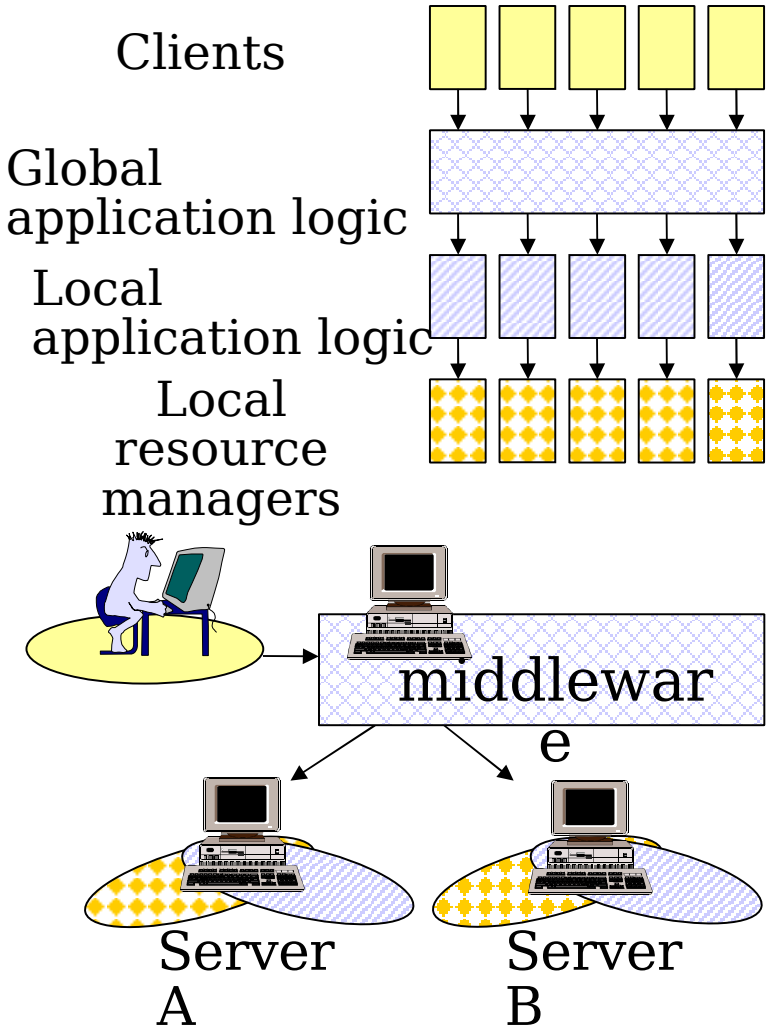
Separated application logic



Any combination thereof



Middleware



- 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.

Distributed IS

There are four basic problems to solve in a distributed information system. Everything else derives from these basic aspects:

- Use the appropriate abstraction for distribution in view of the communication infrastructure available. The abstraction must hide the network stack and provide high level primitives
- How to embed the service abstraction part of the programming language in a more or less transparent manner.
  - Don't forget this important aspect: whatever you design, others will have to program and use it
- How to exchange data between machines that might use different representations for different data types. This involves two aspects:
  - data type formats (e.g., byte orders in different architectures)
  - data structures (need to be flattened and then reconstructed)
- How to describe and find services:
  - so that clients can use them
  - to avoid tight integration

Real & artificial dependency and Standardisation

- Coupling is the dependency between interacting systems.
  - Real dependency
    - » Set of features or services consumed from other systems. Always exists and cannot be reduced.
  - Artificial dependency
    - » Factors that requesting party must comply with in order to consumed features or services (e.g., platform dependency, API dependency)
    - » Always exists but costs can be reduced.
- In each technology some of the following levels are standardized (as we can see further, web services is just one standard)
- Standardization levels (Sheth, 1998)
  - System level: Operating systems, bit-streams, protocols
  - Syntactic level: Programming languages, markup languages
  - Structural level: Data models, Process models
  - Semantic level: Concepts and states used in data interchange
- Loose coupling describes the configuration in which artificial dependency has been reduced to the minimum.
  - Abstraction of the system in standardization levels

SOA Summary

- Different level of integration is required depending of the economical / use-case scenario
- Standardization of the system is done in: System level, Syntactic level, Structural level and Semantic level
- Maintenance costs are reduced in each level: more abstraction, less maintenance
- Depending of the stage of the system life cycle we need either bottom-up- / top-down- or both design methodologies
- n-tier system architecture separates the resources, global application logic, application logic and presentation layer: more layers separated, more abstraction possible
- Middleware simplifies the design of the systems, makes the subsystems transparent by locating and executing resources (handles the abstraction layers)
- Many technologies have been proposed to support middleware based application development (RMI, Corba etc.). Web Services is a web based middleware

Web Services

Web Services as distributed systems
Why to use web services?

Distributed Web-based middleware = Web Services

- 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 globalisation 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.
- Many technologies have been proposed for distributed middleware
  - Java RMI, DCOM,.NET, Corba, WebServices, etc...
  - Each of these try to standardize the system interfaces in different levels
  - Web services seem to be the most promising technology platform to do the trick
    - » Standards are developed on every dependency level
    - » Strong industrial support

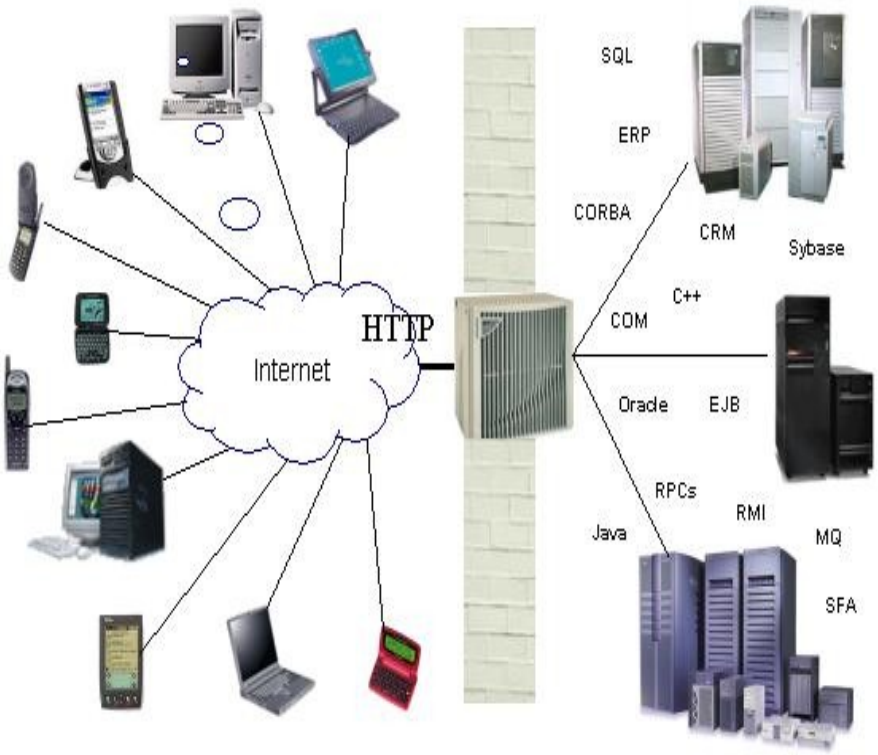
Web Services – How do they work

- The Web services architecture represented by SOAP, UDDI, and WSDL is a direct descendant of conventional middleware platforms. They can be seen as the most basic extensions that are necessary to allow conventional synchronous middleware to interact with each other.
- The model and even the notation followed in this architecture mimics to a very large extent what has been done in RPC, TP-Monitors, CORBA, etc.
- This dependency gives a very good hint of what can be done with these technologies today and what is missing to obtain a complete platform for electronic commerce
- First implementations are just extensions of existing platforms to accept invocations through a Web service interface (e.g., database stored procedure published as Web services)

Web Services – How do they work

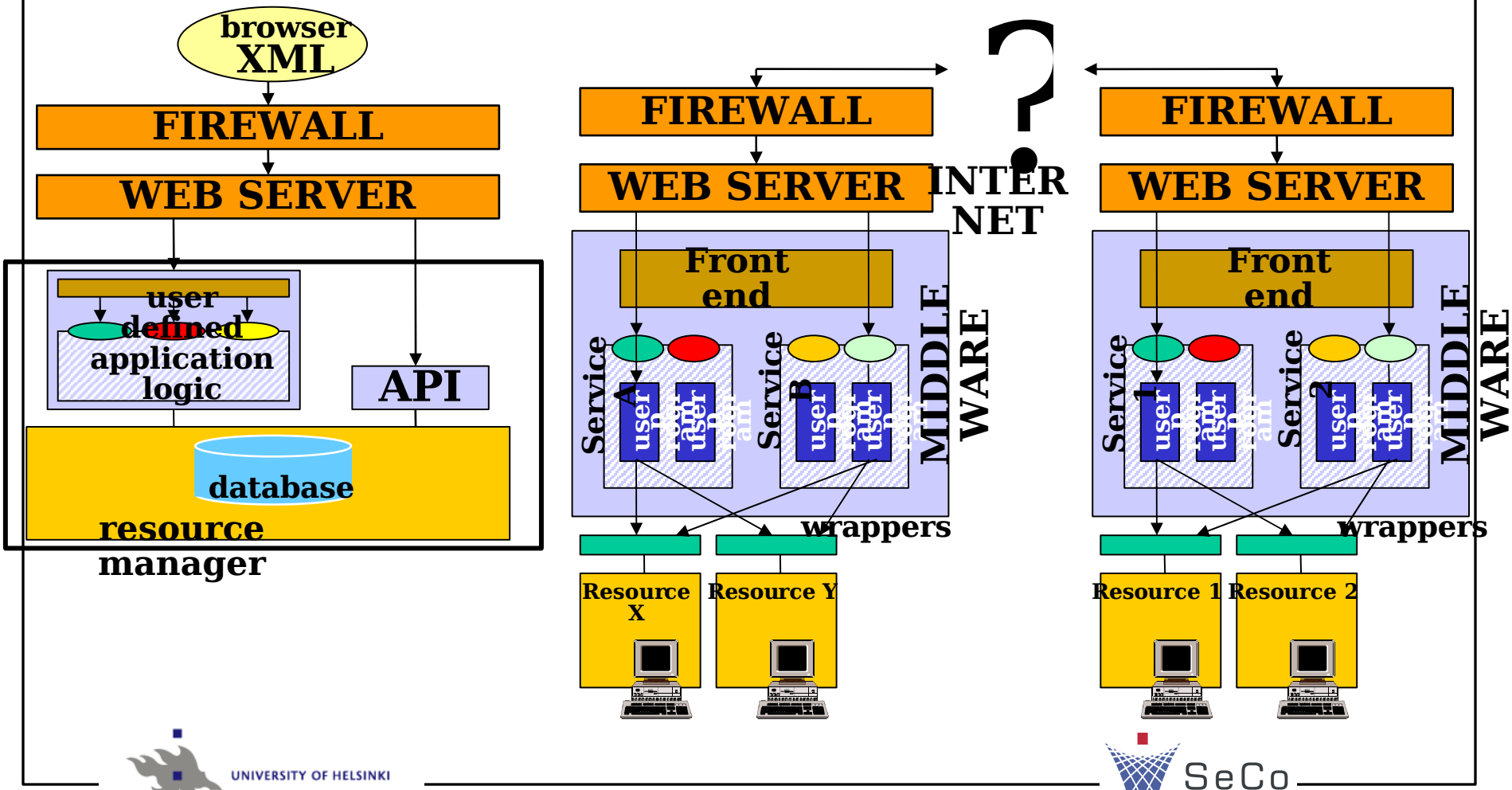
- The next step in that progression leads immediately to the notion of Web services as considered in IBM's Web service architecture.
  - The notion of service in the conventional middleware is now translated into the notion of Web service based on the access channel to that service (the service in fact 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 interface descriptions
    - » UDDI as the WWW version of basic name and directory services

Web Services -– How do they work



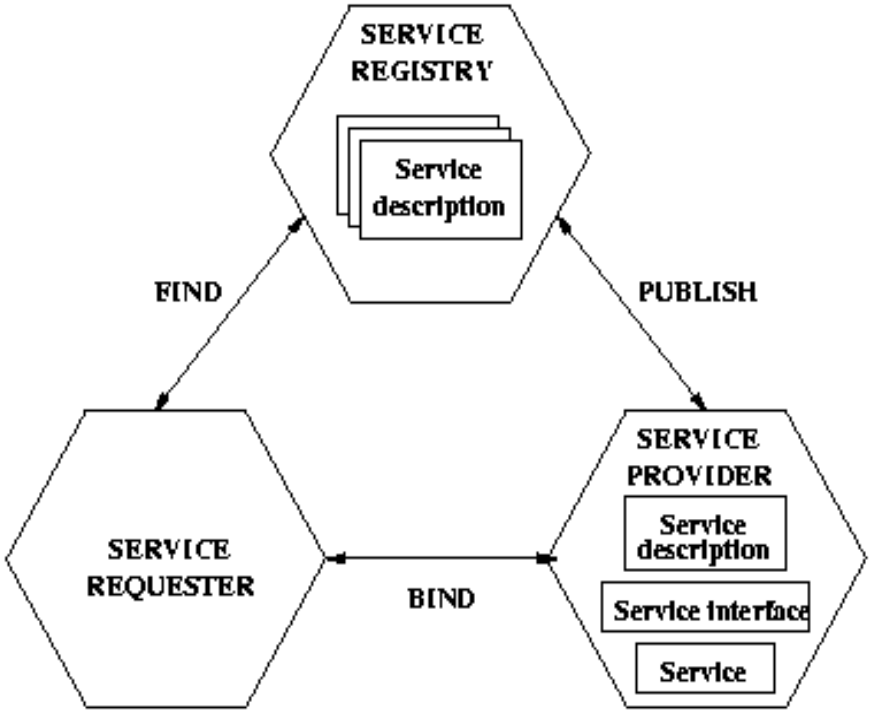
- In context of Web Services, artificial dependency is reduced by making the services
  - System: Hardware and operating system independent (http, ftp, TCP/IP etc.)
  - Syntax: Programming language independent (XML/SOAP, WSDL)
  - Structural: Object model independent (XML Schema)
  - Semantics: “independent” (Ontologies / Workflows: BPEL, ebXML, RosettaNet, RDF/OWL, OWL-S, etc.)
    - » Level of semantics vary dramatically

Web Services -- How do they work



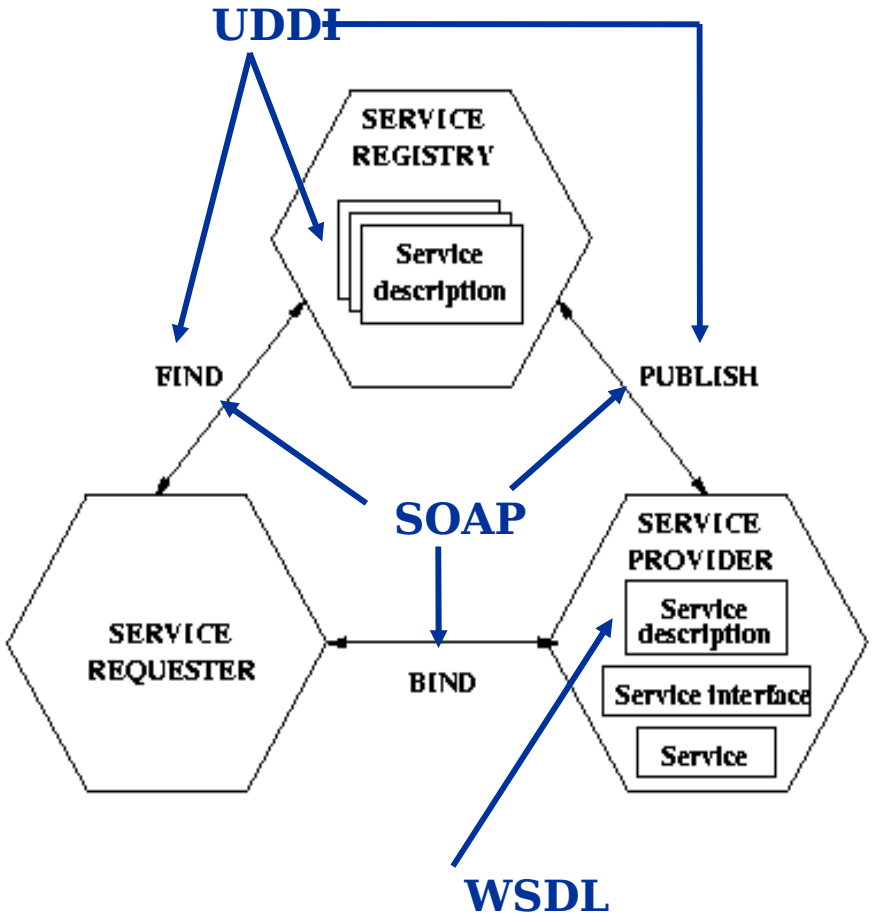
Web Services -– How do they work

- 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

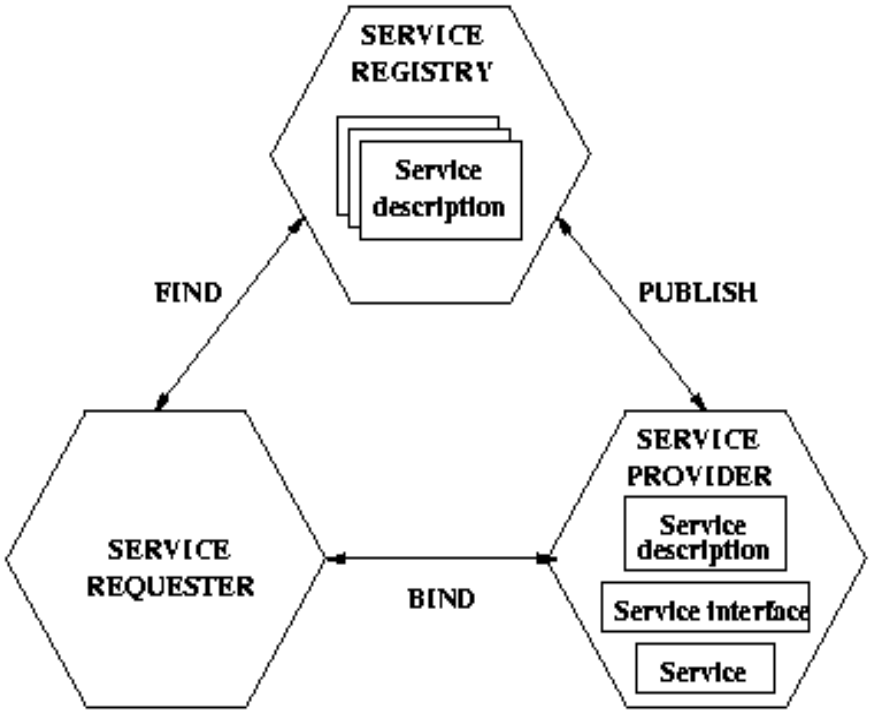


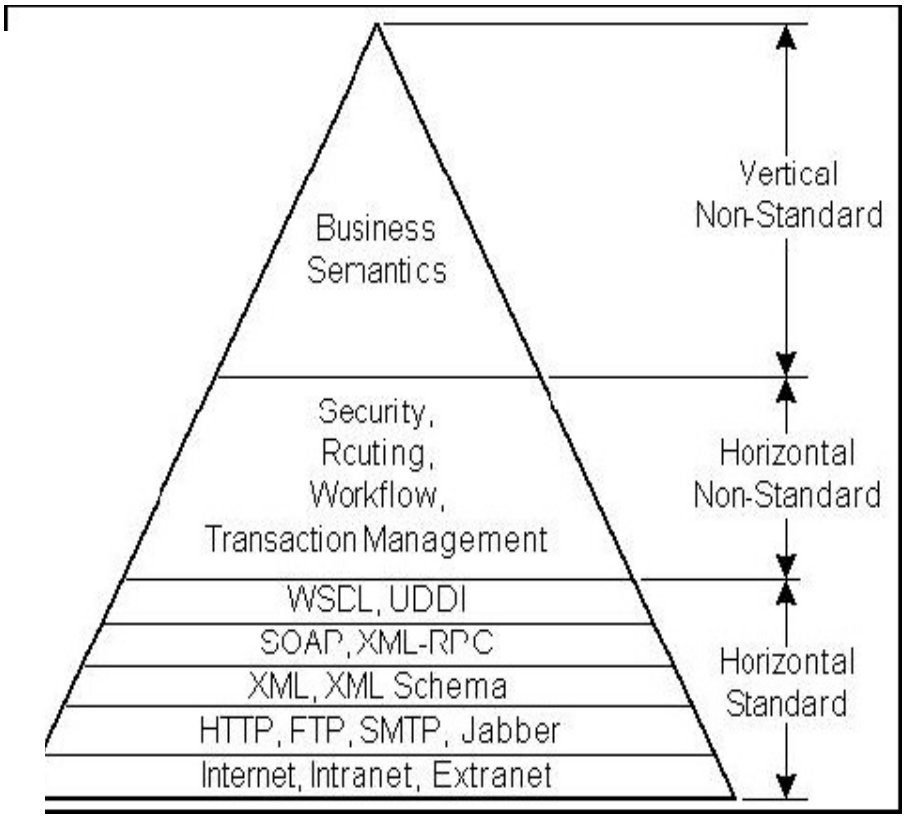
Web Services -– How do they work

- 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)



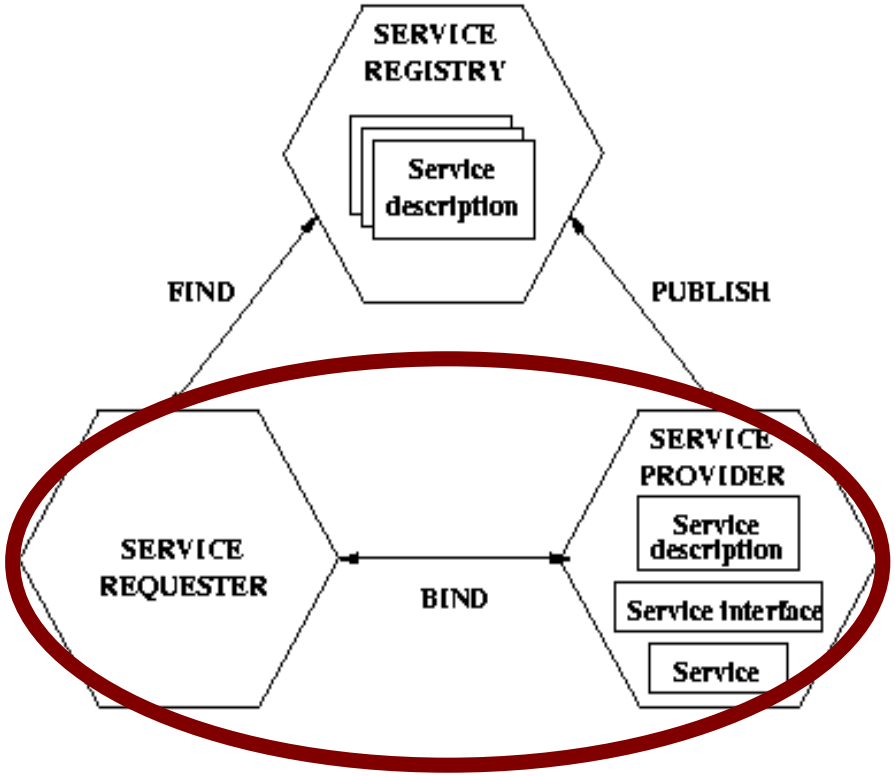
Web Services – The Big Picture





Web Services binding - levels of dependency

- RPC (Remote Procedure Calls)
  - Call the procedures over the wan (internet) and return a response
  - Basically synchronous method calls
- Message queues
  - Send documents rather than call methods
- Workflow descriptions
  - Processes rather than fixed clients
- Semantic Descriptions of data
  - Machine understandable descriptions of services, data and processes



Web Services Summary

- Web Services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. They can be local, distributed, or Web-based. Web services are built on top of open standards such as TCP/IP, HTTP, HTML, and XML. Web services use new standard technologies such as SOAP (Simple Object Access Protocol) for messaging, and UDDI (Universal Description, Discovery and Integration) and WSDL (Web Service Description Language) for publishing.
- These technologies are used to represent abstractions of programming language interfaces in the web and to provide an find/publish and bind/execute framework to enable service oriented computing
- What we have now (IBM WS architecture) is not the final e-commerce architecture but a good start
- Semantic web services are defined using levels of: configurability and automation in data and process interoperability
  - We will further see where the current technologies go wrong and in which presented use cases they are not enough