Distributed Systems Architectures

Slides from Ian Sommerville and Marco Autili 's presentations

Distributed systems

- Virtually all large computer-based systems are now distributed systems.
- Information processing is distributed over several computers rather than confined to a single machine.
- Distributed software engineering is therefore very important for enterprise computing systems.

System types

- **Personal** systems that are not distributed and that are designed to run on a personal computer or workstation.
- **Embedded** systems that run on a single processor or on an integrated group of processors.
- **Distributed** systems where the system software runs on a loosely integrated group of cooperating processors linked by a network.

Distributed system characteristics

- Resource sharing
  - Sharing of hardware and software resources.
- Openness
  - Use of equipment and software from different vendors.
- Concurrency
  - Concurrent processing to enhance performance.
- Scalability
  - Increased throughput by adding new resources.
- Fault tolerance
  - The ability to continue in operation after a fault has occurred.

Distributed system disadvantages

- Complexity
  - Typically, distributed systems are more complex than centralised systems.
- Security
  - More susceptible to external attack.
- Manageability
  - More effort required for system management.
- Unpredictability
  - Unpredictable responses depending on the system organisation and network load.

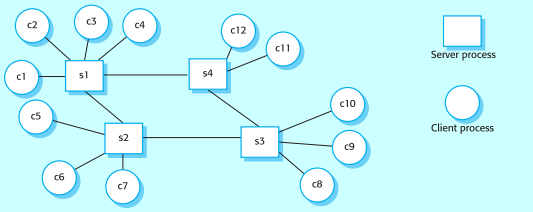
Distributed systems architectures

- Client-server architectures
  - Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services.
- Distributed object architectures
  - No distinction between clients and servers. Any object on the system may provide and use services from other objects.

Client-server architectures

- The application is modelled as a set of services that are provided by servers and a set of clients that use these services.
- Clients know of servers but servers need not know of clients.
- Clients and servers are logical processes
- The mapping of processors to processes is not necessarily 1 : 1.

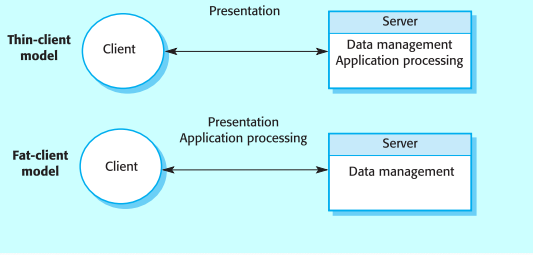
A client-server system



Thin and fat clients

- Thin-client model
  - In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.
- Fat-client model
  - In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system user.

Thin and fat clients



Thin client model

- Used when legacy systems are migrated to client server architectures.
  - The legacy system acts as a server in its own right with a graphical interface implemented on a client.
- A major disadvantage is that it places a heavy processing load on both the server and the network.

Fat client model

- More processing is delegated to the client as the application processing is locally executed.
- Most suitable for new C/S systems where the capabilities of the client system are known in advance.
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients.

Inter-organisational computing

- For security and inter-operability reasons, most distributed computing has been implemented at the enterprise level.
- Local standards, management and operational processes apply.
- Newer models of distributed computing have been designed to support inter-organisational computing where different nodes are located in different organisations.

Service Oriented Architecture

Services

- What are ***services*** ?
  - One of the abstract and context independent definition is:
    - Services are labors that, if accomplished, produce a non tangible useful benefit (Software Technologies Unit of the European Commission).
- What are ***software services*** ?
  - Software services are functionalities provided by software applications that supply:
    - computational resources on request;
    - informational resources on request.
  - Functionalities are contractually defined in the ***service description*** (Software Technologies Unit of the European Commission).

- **S**ervice **O**riented **A**rchitecture (SOA)

Service-oriented architectures

- Based around the notion of externally provided services (web services).
- A web service is a standard approach to making a reusable component available and accessible across the web
  - A tax filing service could provide support for users to fill in their tax forms and submit these to the tax authorities.

A generic service

- *An act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production.*
- Service provision is therefore independent of the application using the service.

Why SOA? (1/2)

- SA have attempted to deal with **increasing** levels of software **complexity** ...
- ... traditional architectures seem to be **reaching the limit** of their ability to deal with such a complexity.
- At the same time, traditional needs of Information Technology (IT) organizations persist:
  - The **need to respond quickly to new requirements** of business;
  - the need to continually **reduce the cost of IT** to the business;
  - the **ability to absorb and integrate** new business partners and new customer sets, ecc.

Why SOA? (2/2)

- We need an architectural framework which allows the **assembly of services** for the **rapid**, and even **dynamic**, **delivery of solutions**.
- SOA is being promoted in the industry as **the next evolutionary step in SA** to help IT organizations meet their evermore complex set of challenges.
- SOA "**could be a 2010 phenomenon**"
  - From: Financial Times, 4 May 2005, Richard Waters: "Plugging together software may soon painless"

SOA is not new!

- It is an alternative *loosely-coupled* model to the more traditionally *tightly-coupled* object-oriented models that have emerged in the past decades.
- The overall design is service-oriented but do not exclude the fact that individual services can themselves be built with object-oriented designs.
- SOA is *object-based*, but it is not, as a whole, *object-oriented*.
- The difference lies in the **"interfaces"** themselves. A classic example of a proto-SOA system that has been around for a while is the (CORBA), which defines similar concepts to SOA

SOA is not new!

- ... SOA is different in that it relies on a more recent advance based upon XML.
- By describing interfaces in an XML-based language called WSDL, services have moved to a more dynamic and flexible interface system than the older IDL found in CORBA.
- Web services aren't the only way to implement SOA.
- CORBA, as just explained earlier is one other way and so are Message-Oriented Middleware systems such as the IBM MQ Series.
- .. to become an architecture model, you need more than just a service description:
  - you need to define how the overall application performs its workflow between services.

SOA characteristics: modularity

- **Modular Decomposability:**
  - break of an application into many smaller services.
  - each module is responsible for a single, **distinct functionality**.
  - Within a "**top-down design approach**", the goal is to obtain the smallest unit of service that can be reused in different contexts.

SOA characteristics: modularity

- **Modular Composability:**
  - software services may be freely combined as a whole with other services to produce new **composed services.**
  - each service is still responsible for a single, **distinct functionality**.
  - Within a "**bottom-up design approach**", the goal is to create services sufficiently independent to be reused in entirely different applications from the ones for which they were originally intended.

SOA characteristics: modularity

- **Modular Understandability:**
  - A service should be described in such a way that a consumer/developer is able **to understand the functionality and other specification** of the service without having any knowledge of other services.
  - If the **functionality provided** from the service is not understandable, the **consumer** deciding whether to use the service will have a difficult time making a decision.
  - If the **behavioral specification** of a service is tied to other specification, **developer** will have hard time to reuse the service.

SOA characteristics: modularity

- **Modular Continuity:**
  - the impact of **changes in one service should not require changes** in other services or in the consumers of the service.
  - An service description that **does not sufficiently hide the implementation details** of the service creates a domino effect when changes are needed:
    - other service may have assumption on this implementation details.

SOA characteristics

- **Interoperability:**
  - SOA stresses **interoperability**:
    - the ability of systems using different platforms and languages to communicate with each other.
  - Protocol and a data format should be standardized:
    - For instance JAX-RPC and JAXM map Java data types to SOAP.

SOA characteristics

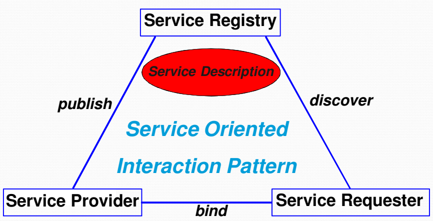
- **Loose Coupling:**
  - The **binding** from the service requester to the service provider **should loosely couple the service**.
  - This means that the service **requester has no knowledge of the technical details** of the provider's implementation, such as the programming language, deployment platform.
  - Implementation on each side of the conversation can change without impacting the other, provided that **the message schema stays the same**.
  - Legacy code, such as COBOL, can be replaced with new Java code without having any impact on the service requester.

Service Orientation (1/5)

- Applications are ***assembled*** from reusable building *"blocks"*, but in this case the blocks are ***services***.
- Services are ***functionalities*** defined by contractual descriptions:
  - Services functionality as mix of syntax, semantics and behavior.
- Application assembly is ***based only*** on ***service description***.
- ***The most interesting aspects are:***
  - the actual service provider is *located later*:
  - integration ***prior*** or ***during execution***.
  - the focus is on how the service are described in order to support ***dynamic discovery***.

Service Orientation (2/5)

- To support dynamic discovery:
  - [illegible] functionalities is returned.



Service Orientation (3/5)

- Service requestor ***is not tied*** to a particular service provider:
  - service ***providers may be substituted*** whenever they ***obey to the contract*** imposed by the service description.
- Due to ***dynamic availability***, a service may be removed from the registry at any time:
  - running application must be able to ***releasing*** (***incorporating***) ***departing*** (***arriving***) services.

Service Orientation (3/5)

- To deal with *dynamic availability:*
  - some service platforms provide ***notification mechanisms*** used to inform service requesters of the arrival or departure of the services.
  - others use the concept of a ***service lease***, which means that service availability is guaranteed only for a determined time period after which the lease must be renewed.

Service Orientation (4/5)

- *Service composition* is an abstract composition:
  - it is based ***only on service descriptions*** and it ***concretizes*** only at run-time.
- *Hierarchical service composition* is achieved when a service composition itself has a service description.

Service Orientation (5/5)

- Development platforms such as the one of PLASTIC should respect the *service orientation principles*.
- Examples of other platforms include the CORBATrader, Jini, OSGi, IBM WebSphere, ecc.

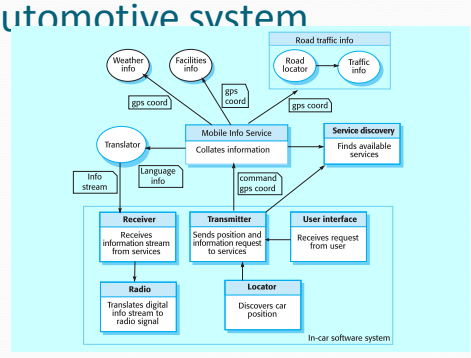
Services and distributed objects

- Provider independence.
- Public advertising of service availability.
- Potentially, run-time service binding.
- Opportunistic construction of new services through composition.
- Pay for use of services.
- Smaller, more compact applications.
- Reactive and adaptive applications.

Services scenario

- An in-car information system provides drivers with information on weather, road traffic conditions, local information etc. This is linked to car radio so that information is delivered as a signal on a specific radio channel.
- The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services. Information may be delivered in the driver's specified language.

Automotive system



eHealth scenario

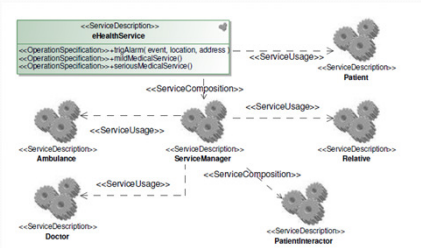
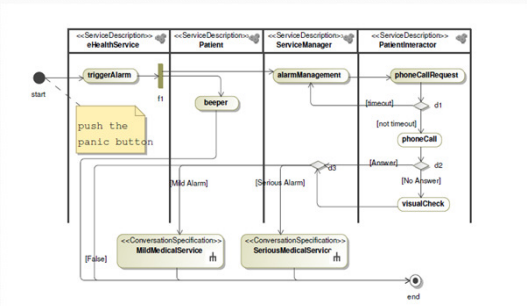


Fig. 5. The eHealth Service Description Diagram

eHealth scenario



Services standards

- Services are based on agreed, XML-based standards so can be provided on any platform and written in any programming language.
- Key standards
  - SOAP - Simple Object Access Protocol;
  - WSDL - Web Services Description Language;
  - UDDI - Universal Description, Discovery and Integration.