Chapters 12/31

Distributed Systems
Architectures & Service-Oriented Software Engineering
Distributed Systems Architectures

Architectural design for software that executes on more than one processor
Service-Oriented Software Eng

A way of developing distributed systems where the components are stand-alone services.

“Currently a hot topic…as important a development as object-oriented software engineering.”

(For Chap 31, skip sections: 31.2 and 31.3.1)
Objectives

- To explain the advantages and disadvantages of distributed systems architectures.
- To describe different approaches to the development of client-server systems.
- To explain the differences between client-server and distributed object architectures.
- To describe object request brokers and the principles underlying the CORBA standards.
- To introduce service-oriented SE, an increasingly important approach in business application development.
Topics covered

- Multiprocessing systems
- Distributed systems architectures
  - Client-server architectures
  - Distributed object architectures
- CORBA (Common Object Request Broker Architecture)
- Service-oriented SE
Topics covered

- Multiprocessing systems
- Distributed systems architectures
  - Client-server architectures
  - Distributed object architectures
- CORBA (Common Object Request Broker Architecture)
- Service-oriented SE
Multiprocessor architectures

- System composed of multiple *processes* which may or may not execute on different *processors*.
- Distribution of process to processor may be pre-determined (e.g., by type of process) or may be under the control of a dispatcher.
A multiprocessor traffic control system
Types of multiprocessing systems

- **Personal systems** that are not distributed and that are designed to run on a personal computer or workstation. (e.g., word processors)

- **Embedded systems** that run on a single processor or on an integrated group of processors. (e.g., control systems, real-time systems)

- **Distributed systems** where the system software runs on a loosely integrated group of processors linked by a network. (e.g., ATM systems)
Distributed systems

- Virtually all large, computer-based systems are now distributed systems.
- Processing is distributed over several computers rather than confined to a single machine.
- Distributed software engineering is now very important.
Distributed system characteristics / advantages

- Resource sharing *(hardware and software)*
- Openness *(standard protocols allow equipment and software from different vendors to be combined)*
- Concurrency *(parallel processing)*
- Scalability *(up to capacity of network)*
- Fault tolerance *(potential)*
- Transparency *(resources can be accessed without knowledge of their physical or network location – item deleted after 6th Ed.)*
Distributed system disadvantages

- Complexity *(number of factors affecting emergent properties)*
- Security *(multiple access points, network eavesdropping, etc.)*
- Manageability *(heterogeneity)*
- Unpredictable responsiveness
<table>
<thead>
<tr>
<th>Design issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource identification</td>
<td>The resources in a distributed system are spread across different computers and a naming scheme has to be devised so that users can discover and refer to the resources that they need. An example of such a naming scheme is the URL (Uniform Resource Locator) that is used to identify WWW pages. If a meaningful and universally understood identification scheme is not used then many of these resources will be inaccessible to system users.</td>
</tr>
<tr>
<td>Communications</td>
<td>The universal availability of the Internet and the efficient implementation of Internet TCP/IP communication protocols means that, for most distributed systems, these are the most effective way for the computers to communicate. However, where there are specific requirements for performance, reliability etc. alternative approaches to communications may be used.</td>
</tr>
<tr>
<td>Quality of service</td>
<td>The quality of service offered by a system reflects its performance, availability and reliability. It is affected by a number of factors such as the allocation of processes to processes in the system, the distribution of resources across the system, the network and the system hardware and the adaptability of the system.</td>
</tr>
<tr>
<td>Software architectures</td>
<td>The software architecture describes how the application functionality is distributed over a number of logical components and how these components are distributed across processors. Choosing the right architecture for an application is essential to achieve the desired quality of service.</td>
</tr>
</tbody>
</table>

## Issues in distributed system design
Distributed systems architectures

- **Client-server architectures** – distributed services are called on by clients. Servers that provide services are treated differently from clients that use services.

- **Distributed object architectures** – removes distinction between clients and servers. Any object in the system may provide and use services from other objects.
Middleware

- Software that manages and enables communication between the diverse components of a distributed system.
- Usually off-the-shelf rather than specially written.
- Distributed system frameworks, e.g. CORBA and DCOM, is an important class of middleware described later.
Client-server architectures

- The application is modelled as a set of **services that are provided by servers** and a set of **clients that use the services**.
- Clients must know of servers but servers need not know of clients. (e.g., “installing” a network printer)
- Clients and servers are **logical processes** as opposed to physical machines.
- The **mapping of processors to processes is not necessarily 1:1**.
Processes in a client-server system
Computers in a client-server network
Application layers model

- **Presentation layer** – concerned with presenting the results of a computation to system users and with collecting user inputs.
  - implemented on client only

- **Application processing layer** – concerned with implementing the logic (functionality) of the application.
  - implemented on client or server

- **Data management layer** – concerned with all system database operations.
  - implemented on server only
Application layers

- Presentation layer
- Application processing layer
- Data management layer
Distributing layers to processors

- **Two-tier C/S architecture** — the three layers are distributed between a server processor and a set of clients.

- **Three-tier C/S architecture** — the layers are distributed among two server processes / processors and a set of clients.

- **Multi-tier C/S architectures** — the layers are distributed among multiple server processes / processors (each associated with a separate database, for example) and a set of clients.
Two-tier architectures: thin and fat clients

- **Thin-client model** — the *application logic and data management* are implemented on the server; clients only handle the *user interface*.

- **Fat-client model** — the server is only responsible for *data management*; clients handle the *application logic and the user interface*. 
Thin and fat clients

Thin-client model

Fat-client model

Presentation

Server

Data management

Application processing

Presentation

Application processing

Server

Data management
Thin-client model

- Often **used when centralized legacy systems evolve to a C/S architecture**; the user interface is migrated to workstations or terminals and the legacy system acts as a server.

- Places a **heavy processing load on both the server and the network**; this is **wasteful** when clients are powerful workstations.
Fat-client model

- More processing is delegated to the client as the application logic is executed locally (i.e., on the client).
- But system management is more complex, especially when application function changes and new logic must be installed on each client.
A client-server ATM system

fat- or thin-client model?
Three-tier architectures

- Each application architecture layer may execute on a separate processor.

- Allows for better performance and scalability than a thin-client approach and is simpler to manage than a fat-client approach. (Why?)
A 3-tier C/S architecture
An internet banking system

- **Client**
  - HTTP interaction

- **Web server**
  - Account service provision

- **Database server**
  - SQL query
  - SQL
  - Customer account database

**Presentation**

©Ian Sommerville 2000  Software Engineering. Chapters 12/31  Slide 29
## Use of C/S architectures

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-tier C/S architecture with thin clients</td>
<td>- Legacy system applications where separating application processing and data management is impractical</td>
</tr>
<tr>
<td></td>
<td>- Computationally-intensive applications such as compilers with little or no data management</td>
</tr>
<tr>
<td></td>
<td>- Data-intensive applications (browsing and querying) with little or no application processing.</td>
</tr>
<tr>
<td>Two-tier C/S architecture with fat clients</td>
<td>- Applications where application processing is provided by COTS (e.g. Microsoft Excel) on the client</td>
</tr>
<tr>
<td></td>
<td>- Applications where computationally-intensive processing of data (e.g. data visualisation) is required.</td>
</tr>
<tr>
<td></td>
<td>- Applications with relatively stable end-user functionality used in an environment with well-established system management</td>
</tr>
<tr>
<td>Three-tier or multi-tier C/S architecture</td>
<td>- Large scale applications with hundreds or thousands of clients</td>
</tr>
<tr>
<td></td>
<td>- Applications where both the data and the application are volatile.</td>
</tr>
<tr>
<td></td>
<td>- Applications where data from multiple sources are integrated</td>
</tr>
</tbody>
</table>
Review of C/S system models/types

- **Two logical process types**
  - servers (providing services)
  - clients (using services)

- **Three application layers**
  - presentation (always associated with clients)
  - processing logic (thin model: associated with server; fat model: associated with clients)
  - data management (always associated with servers)

- **Three layer-to-processor mappings**
  - 2-tier (1 server + a set of clients)
  - 3-tier (2 servers + a set of clients)
  - multi-tier (n servers + a set of clients)
Distributed object architectures

- Eliminates the distinction between clients and servers.
- System components are objects that provide services to, and receive services from, other objects.
- Object communication is through middleware called an object request broker (effectively a software bus)
Distributed object architectures

Object name

Services

Software bus (ORB)
Advantages of distributed object architectures

- Allows system designer to delay decisions on where and how services should be provided. (Service-providing objects may execute on any network node.)
- Very open architecture – allows new resources to be added as required.
- Flexible and scaleable.
- System is dynamically reconfigurable – objects can migrate across the network as required. (Thus improving performance.)
Uses of distributed object architectures

1. As a **logical model for structuring and organizing a system** solely in terms of services provided by a number of distributed objects. (E.g., a data mining system.)

2. As a flexible **means of implementing C/S systems**. Both clients and servers are realised as distributed objects communicating through a software bus.
A data mining system
Disadvantages of distributed object architectures

- Complex to design due to model generality and flexibility in service provision.
- C/S systems seem to be a more natural model for most systems. (They reflect many human service transactions.)
CORBA (Common Object Request Broker Architecture) – est. 1991

- International standards for an Object Request Broker (ORB) – middleware to manage communications between distributed objects. (Version 3.1, 2008)
- Several implementations of CORBA are available (Unix and MS OS’s). www.corba.org
- Defined by the Object Management Group (> 500 companies, including Sun, HP, IBM). (OMG → UML)
- DCOM (Distributed Component Object Model) is an alternative standard developed and implemented by Microsoft – now deprecated in favor of Microsoft .NET
Components of the OMG vision of a distributed application

- **Application objects** designed and implemented for this application.

- **Domain-specific objects** defined by the OMG. (e.g., finance/insurance, healthcare)

- **Fundamental CORBA distributed computing services** such as directories and security management.

- **Horizontal facilities** (such as user interface facilities) used in many different applications. (cross domain)
CORBA application structure

- Application objects
- Domain facilities
- Horizontal CORBA facilities

Object request broker

CORBA services
The OMG Vision

- Application specific
- Horizontal facilities
- Domain specific facilities
- CORBA services
Major elements of the CORBA standards

- **An application object model** where objects encapsulate state and have a well-defined, language-
  neutral interface defined in an IDL (interface definition language).

- **An object request broker (ORB)** that locates objects providing services, sends service requests, and returns results to requesters.

- **A set of general object services** of use to many distributed applications. (Horizontal facilities)

- **A set of common components** built on top of these services. (Task forces are currently defining these.) (domain specific facilities)
CORBA objects

- **Objects that provide services** have IDL skeletons that link their interface to an implementation.
- **Calling objects** have IDL stubs for every object being called.
- IDL stubs and skeletons are “visible” to ORB.
- Objects do not need to know the location or implementation details of other objects.
Object request broker (ORB)

- The ORB handles object communications. It knows of all objects in the system and their interfaces.
- The calling object binds an IDL stub that defines the interface of the called object.
- Calling this stub results in calls to the ORB which then calls the required object through a published IDL skeleton that links the interface to the service implementation.
ORB-based object communications

Calling object

Object providing service

Object Request Broker

IDL stub
(For o2)

IDL skeleton

Object

s (o2)

Object

s (o1)

Calling

object

IDL

stub

(For 02)
Inter-ORB communications

- ORBs handle communications between objects executing on the same machine.
- Inter-ORB communications are used for distributed (remote) object calls.
- CORBA supports ORB-to-ORB communication by providing all ORBs access to all IDL interface definitions and by implementing the OMG’s standard Generic Inter-ORB Protocol (GIOP).
Inter-ORB communications

Object Request Broker

IDL

Network

GIOP

Object Request Broker

IDL

©Ian Sommerville 2000
Software Engineering. Chapters 12/31
Slide 47
Examples of general CORBA services

- **Naming and trading services** – these allow objects to discover and refer to other objects on the network.

- **Notification services** – these allow objects to notify other objects that an event has occurred. *(selective broadcasting)*

- **Transaction services** – these support atomic transactions and rollback on failure. *(recovery functions)*
The future of CORBA support for Internet computing

- Proponents of CORBA envisioned it as becoming the dominant model for code and service-reuse over the Internet.
- However, difficulties encountered with firewalls, insecure machines, etc., have resulted in an increased use of normal HTTP requests in combination with web browsers, as well as interest in other models of inter-organizational computing...
Inter-organizational 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 organizations.
Peer-to-peer architectures

- Peer to peer (p2p) systems are decentralised systems where computations may be carried out by any node in the network.
- The overall system is designed to take advantage of the computational power and storage of a large number of networked computers.
- Most p2p systems have been personal systems but there is increasing business use of this technology.
Service-oriented architectures (Ch. 31)

- A means of developing distributed systems where the components are stand-alone (usually web-based) services
- Standard, XML-based protocols have been developed to support service communication and information exchange ("perhaps the key reason for success")
- Services are platform and language independent.
- Systems can be constructed using services from different providers (using a workflow language) with seamless interaction.
Web services

Service registry

Find

Publish

Service requestor

Bind

Service provider

Service
Web service standards

- XML technologies (XML, XSD, XSLT, ...)
- Support (WS-Security, WS-Addressing, ...)
- Process (WS-BPEL)
- Service definition (UDDI, WSDL)
- Messaging (SOAP)
- Transport (HTTP, HTTPS, SMTP, ...)

©Ian Sommerville 2000 Software Engineering. Chapters 12/31 Slide 54
Principal XML-based standards

- **Messaging**: SOAP (Simple Object Access Protocol): message exchange standard that supports service communication

- **Service Definition**:
  - WSDL (Web Service Definition Language): defines a service interface and its bindings
  - UDDI (Universal Description, Discovery and Integration): defines the components of a service that may be used to discover its existence

- **Process**: WS-BPEL (Web Service Business Process Execution Language): standard for workflow languages used to define service composition
Web services

(UDDI)

Service registry

Find

Publish

Service requestor

Service provider

Bind (SOAP)

©Ian Sommerville 2000
Software Engineering. Chapters 12/31 Slide 56
Principal XML-based standards

- **Messaging**: SOAP (Simple Object Access Protocol): message exchange standard that supports service communication

- **Service Definition**:
  - WSDL (Web Service Definition Language): defines a service interface and its bindings
  - UDDI (Universal Description, Discovery and Integration): defines the components of a service that may be used to discover its existence

- **Process**: WS-BPEL (Web Service Business Process Execution Language): standard for workflow languages used to define service composition
Web services

(UDDI)

Service registry

Find

Publish

Service requestor

Bind (SOAP)

Service provider

Service

(WSDL)
Principal XML-based standards

- **Messaging**: SOAP (Simple Object Access Protocol): message exchange standard that supports service communication

- **Service Definition**:
  - WSDL (Web Service Definition Language): defines a service interface and its bindings
  - **UDDI (Universal Description, Discovery and Integration)**: defines the components of a service that may be used to discover its existence

- **Process**: WS-BPEL (Web Service Business Process Execution Language): standard for workflow languages used to define service composition
Web services

Service registry

(UDDI)

Find

Publish

Service requestor

Service provider

Service

Bind (SOAP)

(WSDL)
Principal XML-based standards

- **Messaging**: SOAP (Simple Object Access Protocol): message exchange standard that supports service communication

- **Service Definition**:
  - **WSDL** (Web Service Definition Language): defines a service interface and its bindings
  - **UDDI** (Universal Description, Discovery and Integration): defines the components of a service that may be used to discover its existence

- **Process**: **WS-BPEL** (Web Service Business Process Execution Language): standard for workflow languages used to define service composition
Examples of “Supporting standards”

- **WS-Reliable Messaging**: ensures messages will be delivered once and once only
- **WS-Security**: specify security policies and standards for use of digital signatures
- **WS-Addressing**: specifies representation of addresses in SOAP messages
- **WS-Transactions**: specifies how transactions should be coordinated across services
A service can be defined as:

A loosely-coupled, reusable software component that encapsulates discrete functionality which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet (HTTP, TCP/IP) and XML-based protocols.
Services as reusable components (cont’d)

- A critical distinction between a service and a component as defined in CBSE is that services are independent.
  - Services do not have a ‘requires’ interface
  - Services rely on message-based communication using XML (versus synchronous method calls)

(CBSE covered in Chap. 19)
Message-based communication vs. interaction via method calls

restaurant order via **synchronous method calls**
Message-based communication vs. interaction via method calls (cont’d)

restaurant order via an XML message

<starter>
    <dish name = “soup” type = “tomato” />
    <dish name = “soup” type = “fish” />
    <dish name = “pigeon salad” />
</starter>

<main course>
    <dish name = “steak” type = “sirloin” cooking = “medium” />
    <dish name = “steak” type = “fillet” cooking = “rare” />
    <dish name = “sea bass”/>
</main course>

<accompaniment>
    <dish name = “french fries” portions = “2” />
    <dish name = “salad” portions = “1” />
</accompaniment>
Software development with services

- Existing services are composed and configured to create new composite services and applications.
- The basis for service composition is often a workflow.
  - Workflows are logical sequences of activities that, together, model a coherent business process.
  - For example, consider a workflow providing a travel reservation services which allows flights, car hire and hotel bookings to be coordinated…
Suppose an airline wishes to compose its own booking services with services offered by a hotel booking agency, car hire and taxi companies, and local attraction providers.

The sequence of steps for this “vacation package” application could be modeled (in simplified form) as the following workflow:
Note that this simplified model does NOT reflect the possibility of service failure, exceptional user requests, etc.
Construction by composition

The process of designing new services by composing existing services is, essentially, a process of software design with reuse.
External service testing issues

- Services are “black-boxes,” so code-based (i.e., “white-box”) testing techniques cannot be used at the service level.
- Services may be modified by the service provider.
- Dynamic binding means that the service used in an application may vary.
- Non-functional behavior is unpredictable as it depends on other (concurrent) uses of the service.
- If services have to be paid for as used, testing may be expensive.
- It may be difficult to test compensating actions which rely on the failure of services.
Key points

- Almost all new large systems are distributed systems.

- Distributed systems support resource sharing, openness, concurrency, scalability, fault tolerance, and transparency.

- Client-server architectures involve services being delivered by servers to programs operating on clients.

- User interface software always runs on the client and data management on the server.
Key points (cont’d)

- In a distributed object architecture, there is no distinction between clients and servers.
- Distributed object systems require **middle-ware** to handle object communications.
- The **CORBA** standards are a set of middle-ware standards that support distributed object architectures.
- **Peer to peer architectures** are decentralised architectures where there is no distinction between clients and servers.
Key points (cont’d)

- **Service-oriented software engineering** is based on the notion that programs can be constructed by composing independent services which encapsulate reusable functionality.
- Service interfaces are defined in **WSDL**. A WSDL specification includes a definition of the interface types and operations, the binding protocol used by the service, and the service location.
- Service-level code-based testing cannot be used in service-oriented systems that rely on externally provided services.