Distributed Systems

Lesson 2
Architectures

University of New York in Tirana
Master of Science in Computer Science
Prof. Dr. Marenglen Biba

Lesson 2

01: Introduction
02: Architectures
03: Processes
04: Communication
05: Naming
06: Synchronization
07: Consistency & Replication
08: Fault Tolerance
09: Security
10: Distributed Object-Based Systems
11: Distributed File Systems
12: Distributed Web-Based Systems
13: Distributed Coordination-Based Systems

Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
  - Centralized Architectures
  - Decentralized Architectures
  - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
  - Wrappers
  - Interceptors
  - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
  - The Feedback Control Model

Organization of DSs

- Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines.
  - To master their complexity, it is crucial that these systems are properly organized.

- There are different ways on how to view the organization of a distributed system, but an obvious one is to make a distinction between:
  - the logical organization of the collection of software components
  - the actual physical realization.

- The organization of distributed systems is mostly about the software components that constitute the system.
  - These software architectures tell us how the various software components are to be organized and how they should interact.

System Architecture

- The actual realization of a distributed system requires that we instantiate and place software components on real machines.
  - There are many different choices that can be made in doing so.

- The logical organization of distributed systems into software components is referred to as software architecture.

- The final instantiation of a software architecture is also referred to as a system architecture.

Architectural Style

- Basic idea
  - Organize into logically different components, and distribute those components over the various machines.

- An architectural style is formulated in terms of:
  - components,
  - the way that components are connected to each other
  - the data exchanged between components
 - how these elements are jointly configured into a system.

Components and Connectors

- A component is a modular unit with well-defined required and provided interfaces that is replaceable within its environment.
  - The important issue about a component for distributed systems is that it can be replaced, provided we respect its interfaces.

- A connector is generally described as a mechanism that mediates communication, coordination, or cooperation among components
  - For example, a connector can be formed by the facilities for (remote) procedure calls, message passing, or streaming data.

- Using components and connectors, we can come to various configurations, which, in turn have been classified into architectural styles.

Architectural Styles

1. Layered architectures
2. Object-based architectures
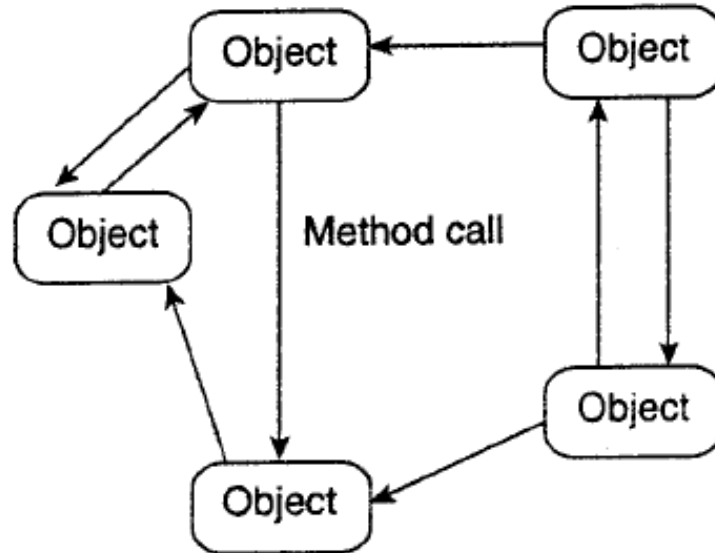3. Data-centered architectures
4. Event-based architectures

Layered architectures
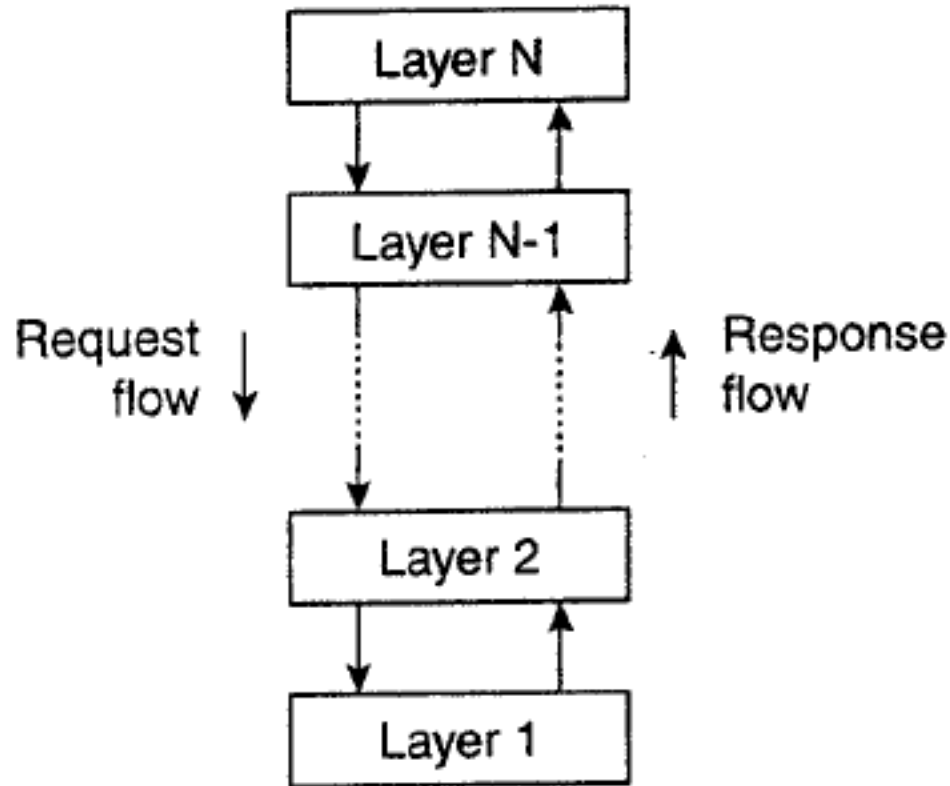
A component at layer L is allowed to call components at layer L_{i-1} but not the other way around.

This model has been widely adopted by the networking community

Object-based Architectures


• In essence, each object corresponds to what we have defined as a component, and these components are connected through a remote procedure call (RPC) mechanism.

• Not surprisingly, this software architecture matches a client-server system architecture.

• The layered and object-based architectures still form the most important styles for large software systems

Data Centered Architectures

- Data-centered architectures evolve around the idea that processes communicate through a common (passive or active) repository.

- It can be argued that for distributed systems these architectures are as important as the layered and object-based architectures.

- For example, a wealth of networked applications have been developed that rely on a shared distributed file system in which virtually all communication takes place through files.

- Likewise, Web-based distributed systems are largely data-centric: processes communicate through the use of shared Web-based data services.

Event-based Architectures
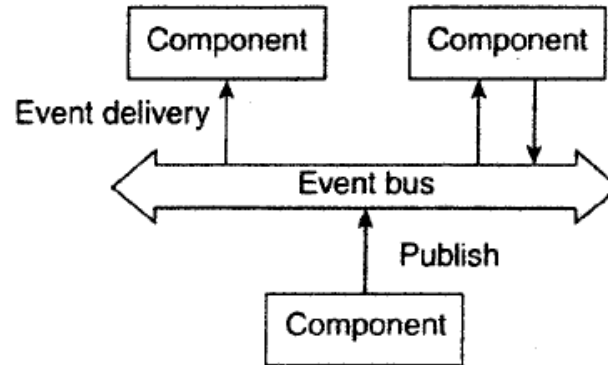
- In event-based architectures, processes essentially communicate through the propagation of events, which optionally also carry data.

- For distributed systems, event propagation has generally been associated with what are known as publish/subscribe systems.

Publish/subscribe Systems

- The basic idea is that processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.

- The main advantage of event-based systems is that processes are loosely coupled.
    - In principle, they need not explicitly refer to each other.
   - This is also referred to as being decoupled in space, or referentially decoupled.

Forms of coordination

	Temporally coupled	Temporally decoupled
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

Figure 2.9: Examples of different forms of coordination.

Shared Data Spaces

- Decoupling processes in space ("anonymous") and also time ("asynchronous") has led to alternative styles.

- Event-based architectures can be combined with data-centered architectures, yielding what is also known as shared data spaces.

- The essence of shared data spaces is that processes are now also decouple in time: they need not both be active when communication takes place.

- Furthermore, many shared data spaces use a SQL-like interface to the shared repository in that sense that data can be accessed using a description rather than an explicit reference, as is the case with files.
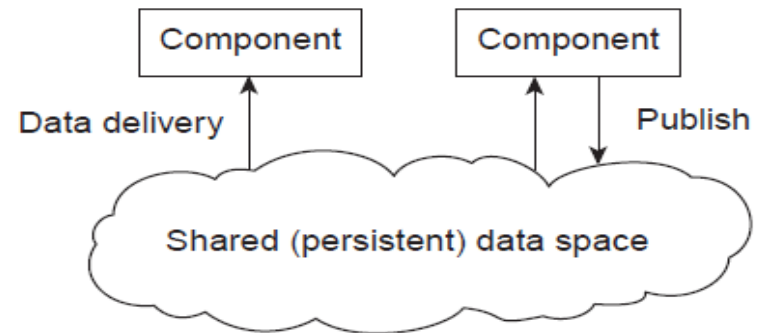
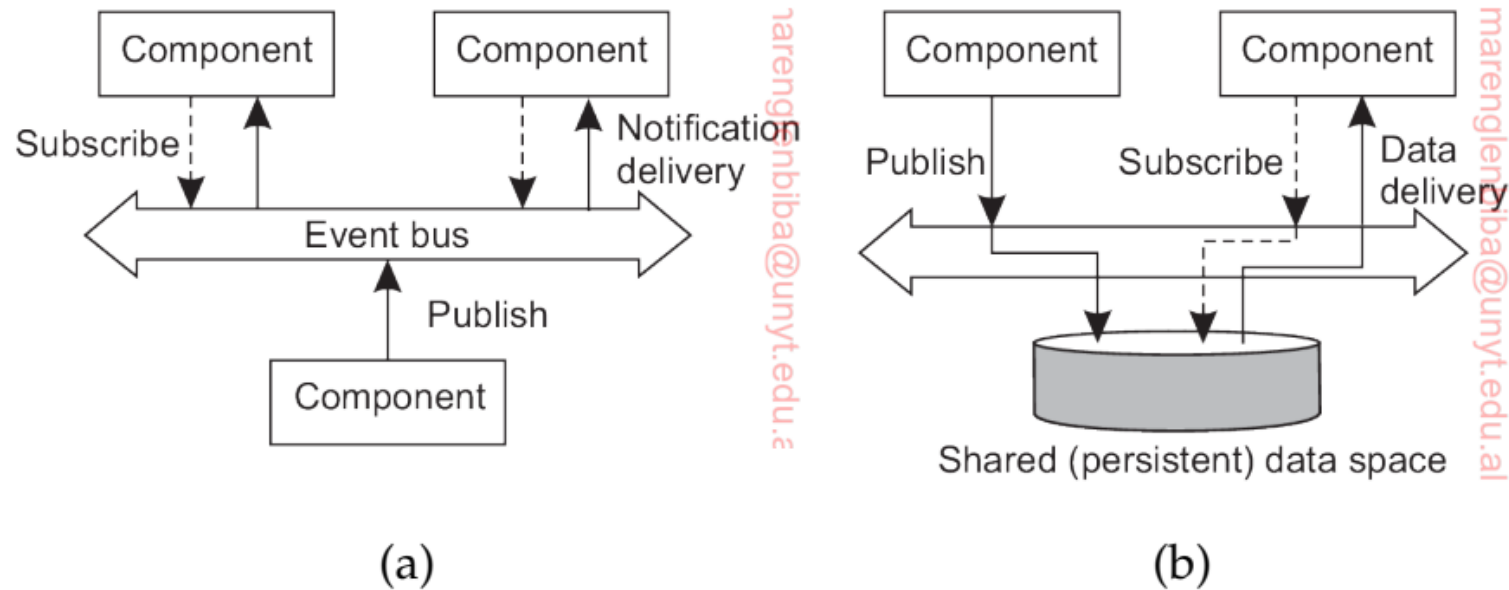# Architectural styles based on coordination

Figure 2.10: The (a) event-based and (b) shared data-space architectural style.

Distribution Transparency and Architectural Styles

- What makes the software architectures important for distributed systems is that they all aim at achieving (at a reasonable level) distribution transparency.

- However, distribution transparency requires making trade-offs between performance, fault tolerance, ease-of-programming, and so on.

- As there is no single solution that will meet the requirements for all possible distributed applications, researchers have abandoned the idea that a single distributed system can be used to cover 90% of all possible cases.

Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
  - Centralized Architectures
  - Decentralized Architectures
  - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
  - Wrappers
  - Interceptors
  - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
 - The Feedback Control Model

System Architectures

- Deciding on software components, their interaction, and their placement leads to an instance of a software architecture also called a system architecture

- These are of three types:
    - Centralized Architectures
    - Decentralized Architectures
    - Hybrid forms

Centralized Architectures

Basic Client–Server Model

- Characteristics:
  - There are processes offering services (servers)
  - There are processes that use services (clients)
  - Clients and servers can be on different machines
  - Clients follow request/reply model wrt to using services
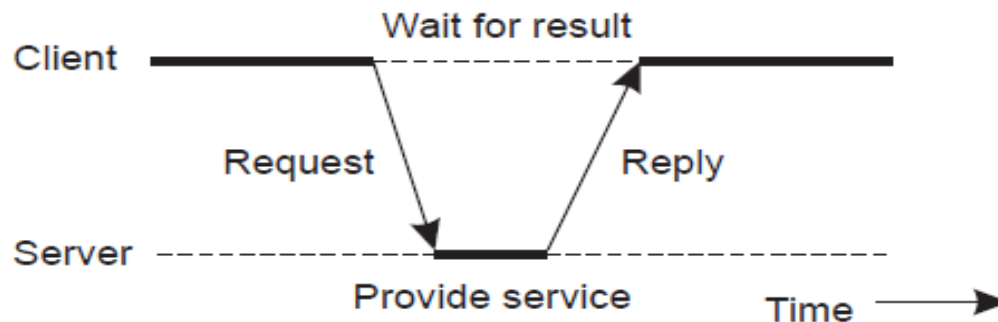
Client-Server in Layered Architectural Style

- Considering that many client-server applications are targeted toward supporting user access to databases, many people have advocated a distinction between the following three levels, essentially following the layered architectural style we discussed previously:
  - The user-interface level
  - The processing level
  - The data level

Example: Internet Search Engine
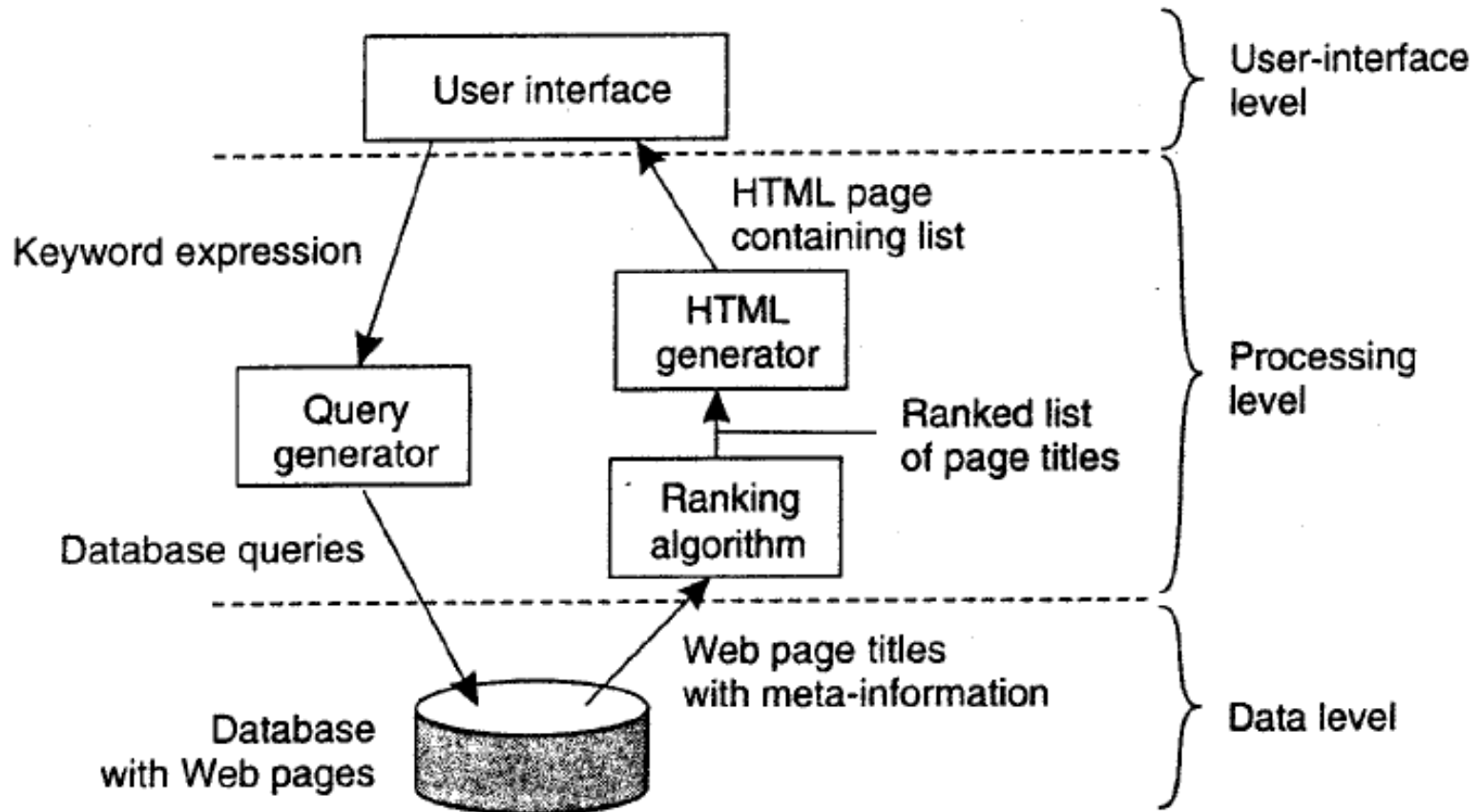
Example: Decision support for stock brokerage

- As a second example, consider a decision support system for a stock brokerage.

- Such a system can be divided into:
  - a front end implementing the user interface,
  - a back end for accessing a database with the financial data
  - the analysis programs between these two.

- Analysis of financial data may require sophisticated methods and techniques from statistics and artificial intelligence.

- In some cases, the core of a financial decision support system may even need to be executed on high-performance computers in order to achieve the throughput and responsiveness that is expected from its users.

Multi-tiered Architectures

- The distinction into three logical levels as discussed so far, suggests a number of possibilities for physically distributing a client-server application across several machines.

- The simplest organization is to have only two types of machines:

  1. A client machine containing only the programs implementing (part of) the user-interface level

  2. A server machine containing the rest, that is the programs implementing the processing and data level

Three-tiered Architecture

- **Single-tiered**: dumb terminal/mainframe configuration
- **Two-tiered:** client/single server configuration
- **Three-tiered**: each layer on separate machine
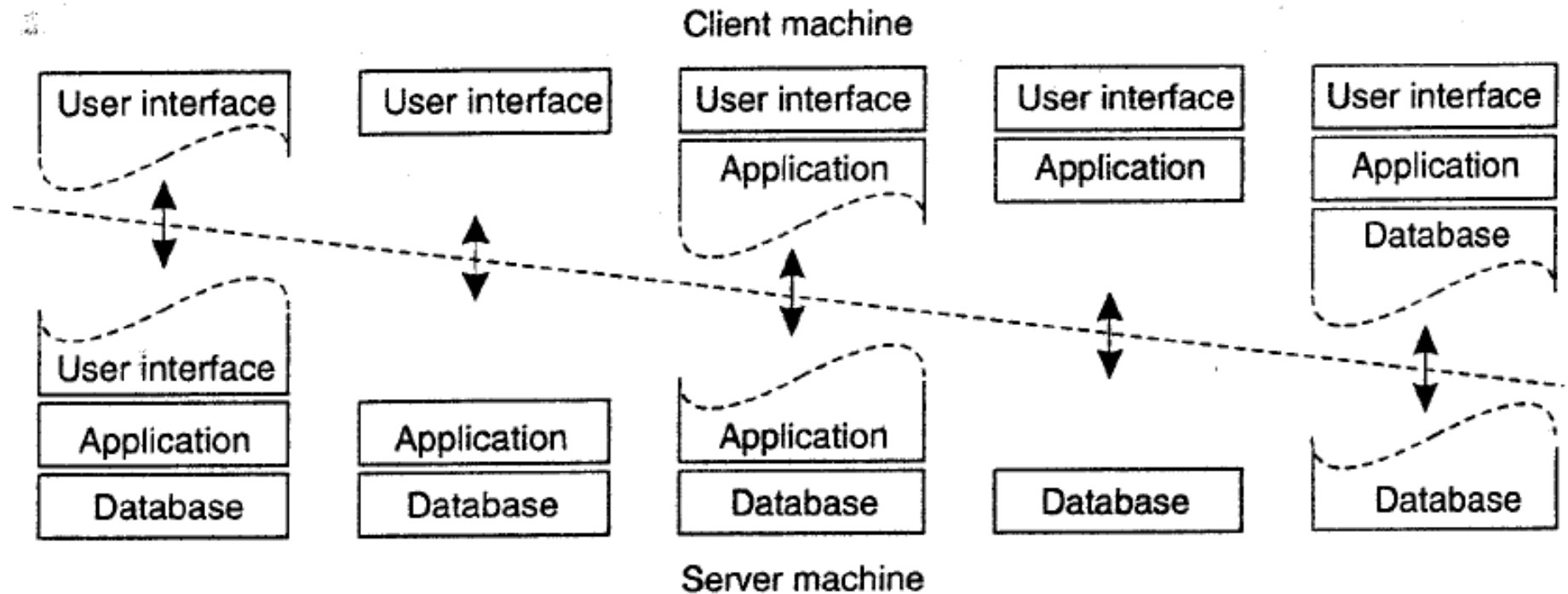
Three-tiered Architecture
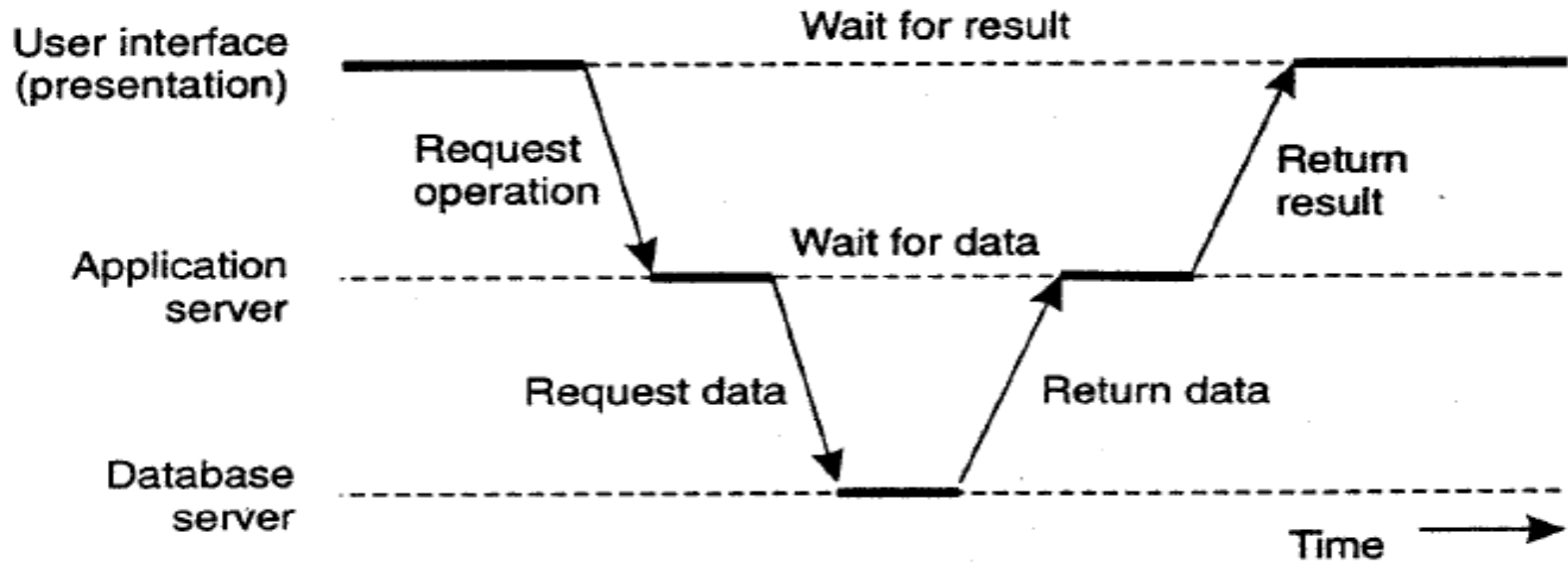
Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
  - Centralized Architectures
  - Decentralized Architectures
  - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
  - Wrappers
  - Interceptors
  - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
  - The Feedback Control Model

Vertical Distribution

- Multitiered client-server architectures are a direct consequence of dividing applications into a user-interface, processing components, and a data level.
  - The different tiers correspond directly with the logical organization of applications.

- In many business environments, distributed processing is equivalent to organizing a client-server application as a multitiered architecture.

- We refer to this type of distribution as vertical distribution.
  - The characteristic feature of vertical distribution is that it is achieved by placing logically different components on different machines.

Horizontal Distribution

- From a system management perspective, having a vertical distribution can help: functions are logically and physically split across multiple machines, where each machine is tailored to a specific group of functions.

- In modem architectures, it is often the distribution of the clients and the servers that counts, which we refer to as horizontal distribution.

- In this type of distribution, a client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load.

- A class of modern system architectures that support horizontal distribution is known as peer-to-peer systems (P2P).

Peer-to-Peer Systems

- From a high-level perspective, the processes that constitute a P2P system are all equal.

- This means that the functions that need to be carried out are represented by every process that constitutes the distributed system.

- As a consequence, much of the interaction between processes is symmetric: each process will act as a client and a server at the same time (which is also referred to as acting as a servent).

Overlay Networks

- Given the symmetric behavior, P2P architectures evolve around the question how to organize the processes in an overlay network, that is, a network in which the nodes are formed by the processes and the links represent the possible communication channels (which are usually realized as TCP connections).

- In general, a process cannot communicate directly with an arbitrary other process, but is required to send messages through the available communication channels.

- Types of overlay networks:
  - Structured P2P: nodes are organized following a specific distributed data structure
  - Unstructured P2P: nodes have randomly selected neighbors
  - Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

Structured P2P

- In a structured peer-to-peer architecture, the overlay network is constructed using a deterministic procedure.
- By far the most-used procedure is to organize the processes through a distributed hash table (DHT).
  - In a DHT-based system, data items are assigned a **random key** from a large identifier space, such as a 128-bit or 160-bit identifier.
  - Likewise, nodes in the system are also assigned a **random number** from the same identifier space.
- The effort of every DHT-based system is then to implement an efficient and deterministic scheme that uniquely maps the key of a data item to the identifier of a node based on some distance metric
- Most importantly, when looking up a data item, the network address of the node responsible for that data item is returned.
- Effectively, this is accomplished by *routing a request* for a data item to the responsible node.

Unstructured P2P Networks

- Unstructured peer-to-peer systems largely rely on randomized algorithms for constructing an overlay network.

- The main idea is that each node maintains a list of neighbors
  - This list is constructed in a more or less random way.
    - Also known as random graph

- Likewise, data items are assumed to be randomly placed on nodes.

- As a consequence, when a node needs to locate a specific data item, the only thing it can effectively do is flood the network with a search query.

Superpeers

- Notably in unstructured peer-to-peer systems, locating relevant data items can become problematic as the network grows.

- The reason for this scalability problem is simple:
  - As there is no deterministic way of routing a lookup request to a specific data item, essentially the only technique a node can resort to is flooding the request.

- Many peer-to-peer systems have proposed to make use of special nodes that maintain an index of data items.

Superpeers Example: CDNs

- In a collaborative content delivery network (CDN), nodes may offer storage for hosting copies of Web pages allowing Web clients to access pages nearby, and thus to access them quickly.
    - In this case a node P may need to seek for resources in a specific part of the network.

- In that case, making use of a broker that collects resource usage for a number of nodes that are in each other's proximity will allow to quickly select a node with sufficient resources.

- This broker is called superpeer.

Superpeers
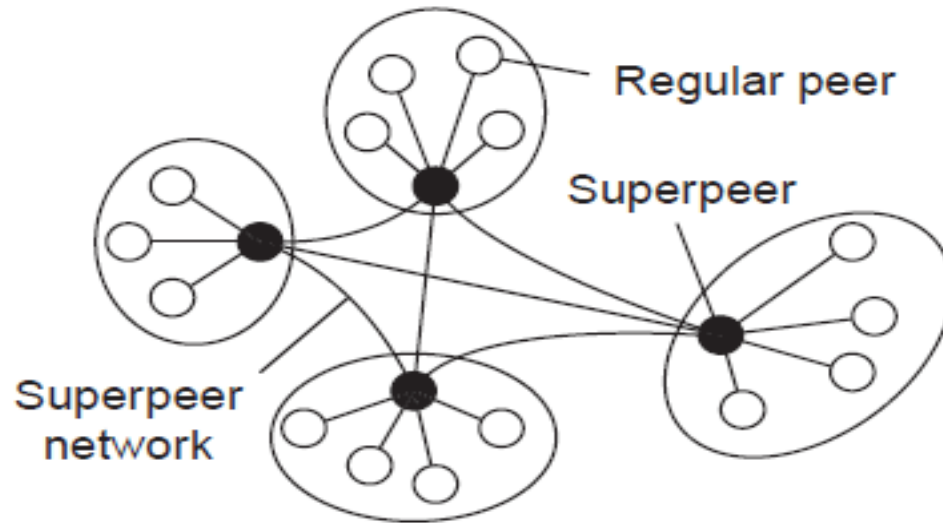
Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
    - Centralized Architectures
    - Decentralized Architectures
    - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
    - Wrappers
    - Interceptors
    - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
    - The Feedback Control Model

Hybrid Architectures
Client-Superpeer relation

- In many cases, the client-superpeer relation is fixed: whenever a regular peer joins the network, it attaches to one of the superpeers and remains attached until it leaves the network.

- Having a fixed association with a superpeer may not always be the best solution.
  - For example, in the case of file-sharing networks, it may be better for a client to attach to a superpeer that maintains an index of files that the client is generally interested in.
  - In that case, chances are bigger that when a client is looking for a specific file, its superpeer will know where to find it.

Edge-server Systems

- An important class of distributed systems that is organized according to a hybrid architecture is formed by edge-server systems.

  - These systems are deployed on the Internet where servers are placed "at the edge" of the network.

- This edge is formed by the boundary between enterprise networks and the actual Internet, for example, as provided by an Internet Service Provider (ISP).

  - Likewise, where end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet.

Hybrid Architectures
Client-server combined with P2P

- Edge-server architectures, which are often used for Content Delivery Networks.
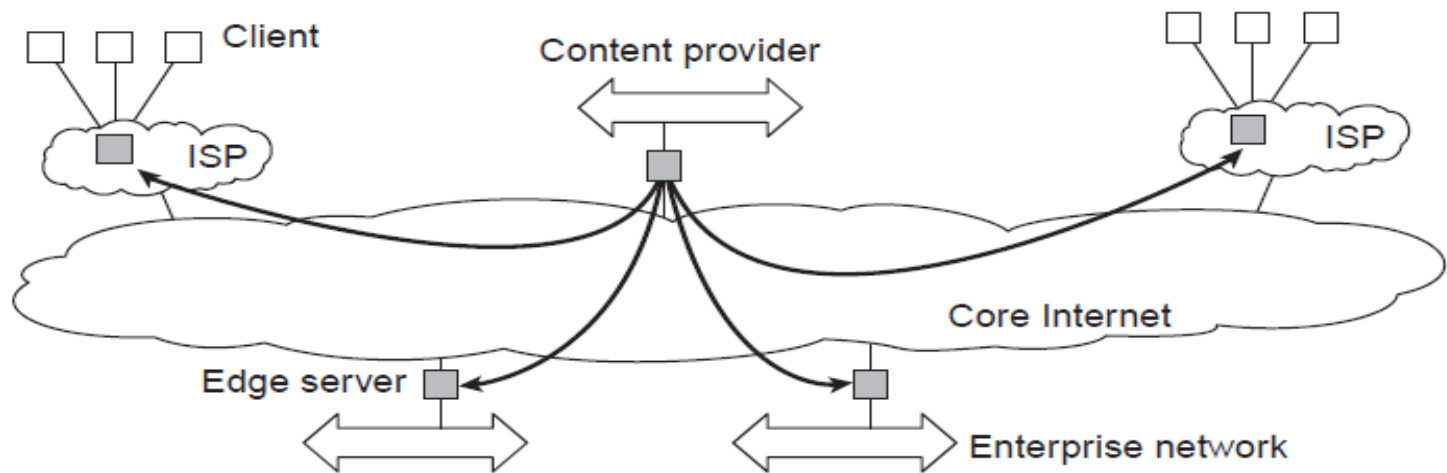
Collaborative Distributed Sytems

- Hybrid structures are notably deployed in collaborative distributed systems.

- Once a node has joined the system, it can use a fully decentralized scheme for collaboration.

- Example: BitTorrent file-sharing system
  - BitTorrent is a peer-to-peer file downloading system

BitTorrent

- To download a file, a user needs to access a global directory, which is just one of a few well-known Web sites.
    - Such a directory contains references to what are called *.torrent* files. A *.torrent* file contains the information that is needed to download a specific file.
    - In particular, it refers to what is known as a tracker, which is a server that is keeping an accurate account of active nodes that have (chunks) of the requested file.
    - An active node is one that is currently downloading another file.
- Obviously, there will be many different trackers, although (there will generally be only a single tracker per file (or collection of files).
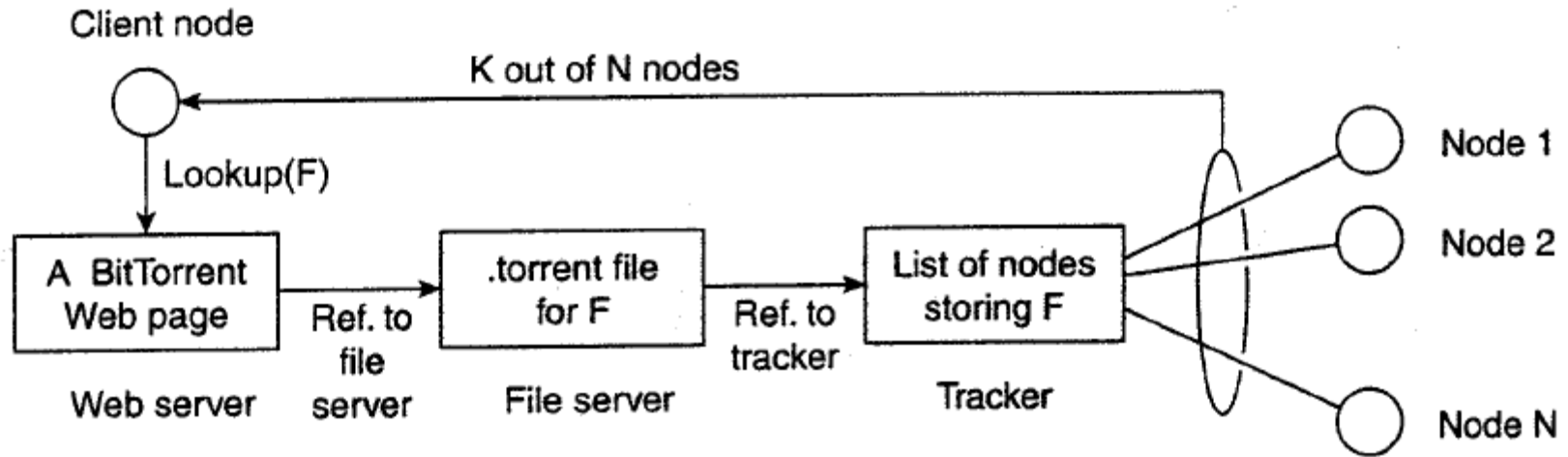
Hybrid Architectures: C/S with P2P – BitTorrent

Figure 2-14. The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.

Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
  - Centralized Architectures
  - Decentralized Architectures
  - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
  - Wrappers
  - Interceptors
  - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
 - The Feedback Control Model

Architectures Vs. Middleware

- In many cases, distributed systems/applications are developed according to a specific architectural style.

- For example, many middleware solutions have adopted an object-based architectural style, such as CORBA.

- Others, like TIB/Rendezvous (TIBCO) provide middleware that follows the event-based architectural style.

CORBA

- The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that:
  - enables software components written in multiple computer languages and running on multiple computers to work together, i.e. it supports multiple platforms.
- DCOM and Web Services
  - Microsoft's counterpart to CORBA is its COM-based Distributed COM (DCOM) architecture. COM/CORBA interoperability is required to integrate Windows desktops into a CORBA-based system.
- Although CORBA and DCOM have achieved some success, Web services on the Internet have become far more triumphant.
- CORBA software from different vendors may not always interoperate at all levels, and DCOM is a Windows-based solution only.

CORBA: simple schema
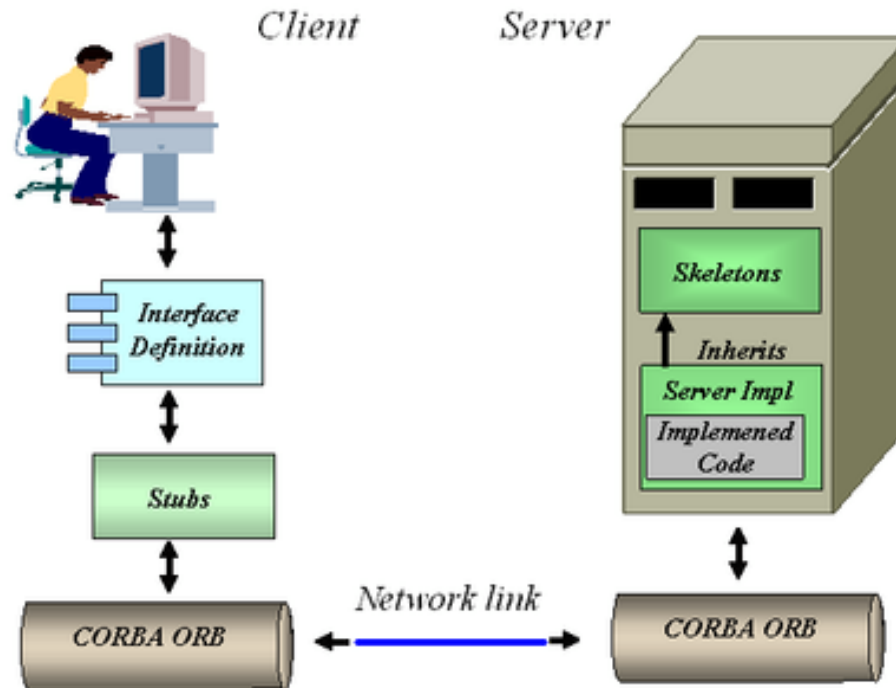
Event-based architecture
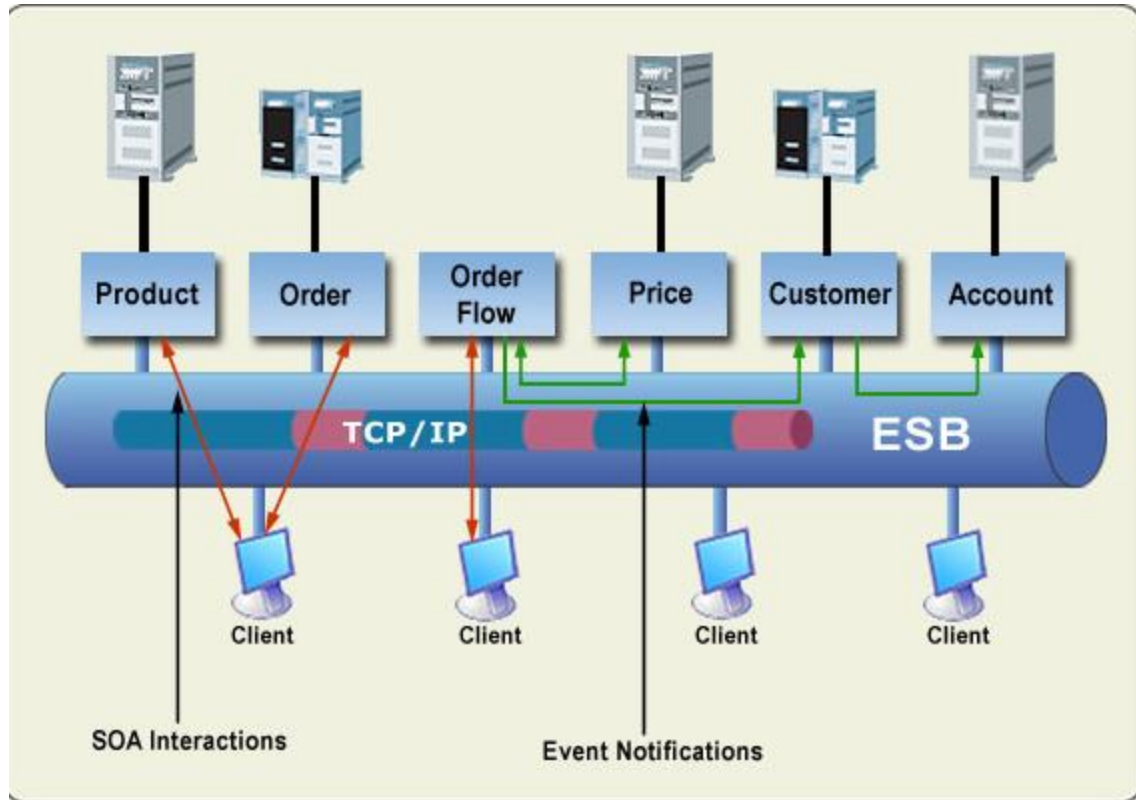
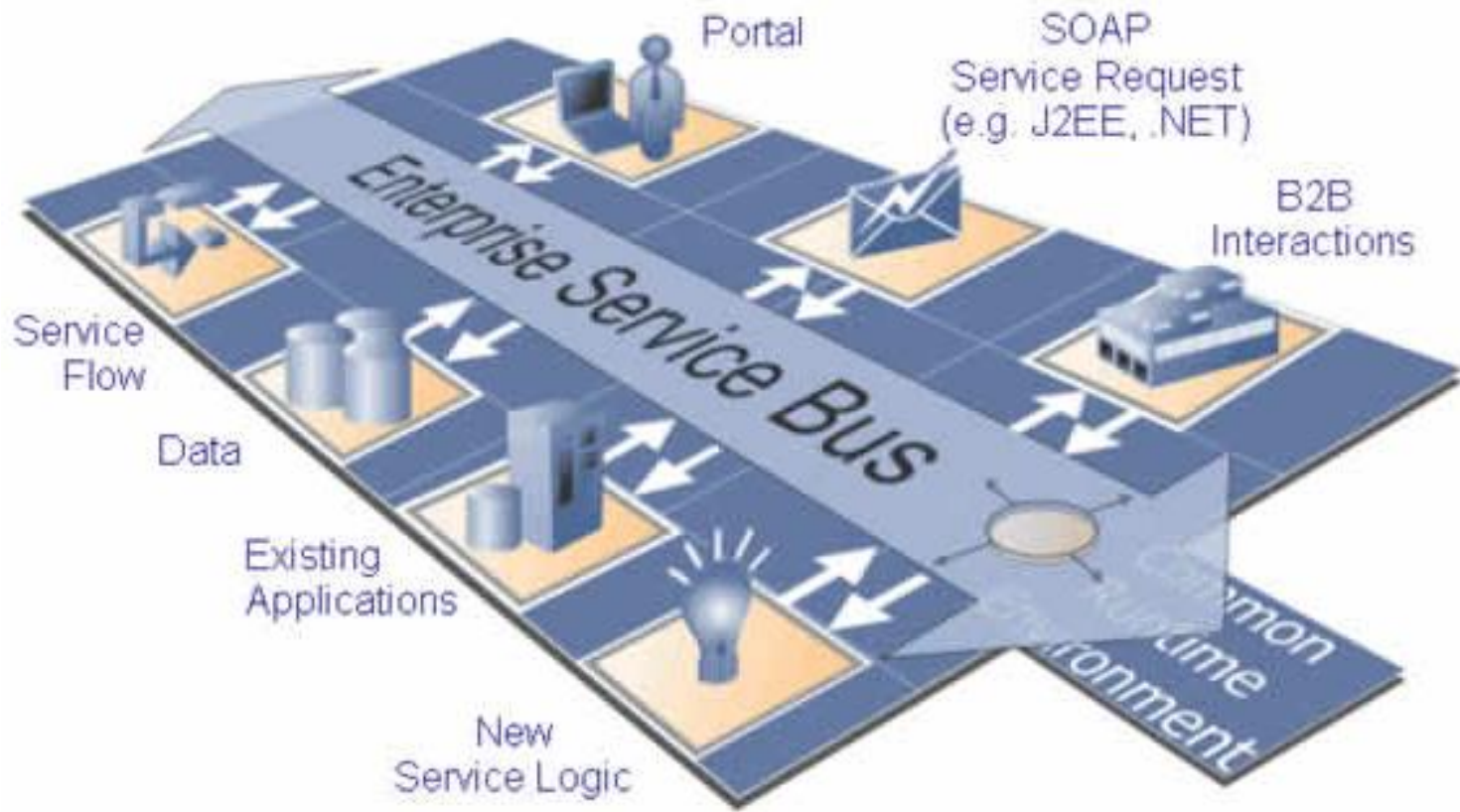# TIBCO

- TIBCO Software Inc. is a global company that develops integration software for companies including those in the energy, manufacturing, retail, healthcare, and financial services industries. Its headquarters is in Palo Alto, California, with offices in North America, Europe, Asia, the Middle East, Africa and South America.
  - The company's major commercial competitors are IBM, Oracle Corporation, and SAP AG.

- Enterprise application integration
- Business Process Management
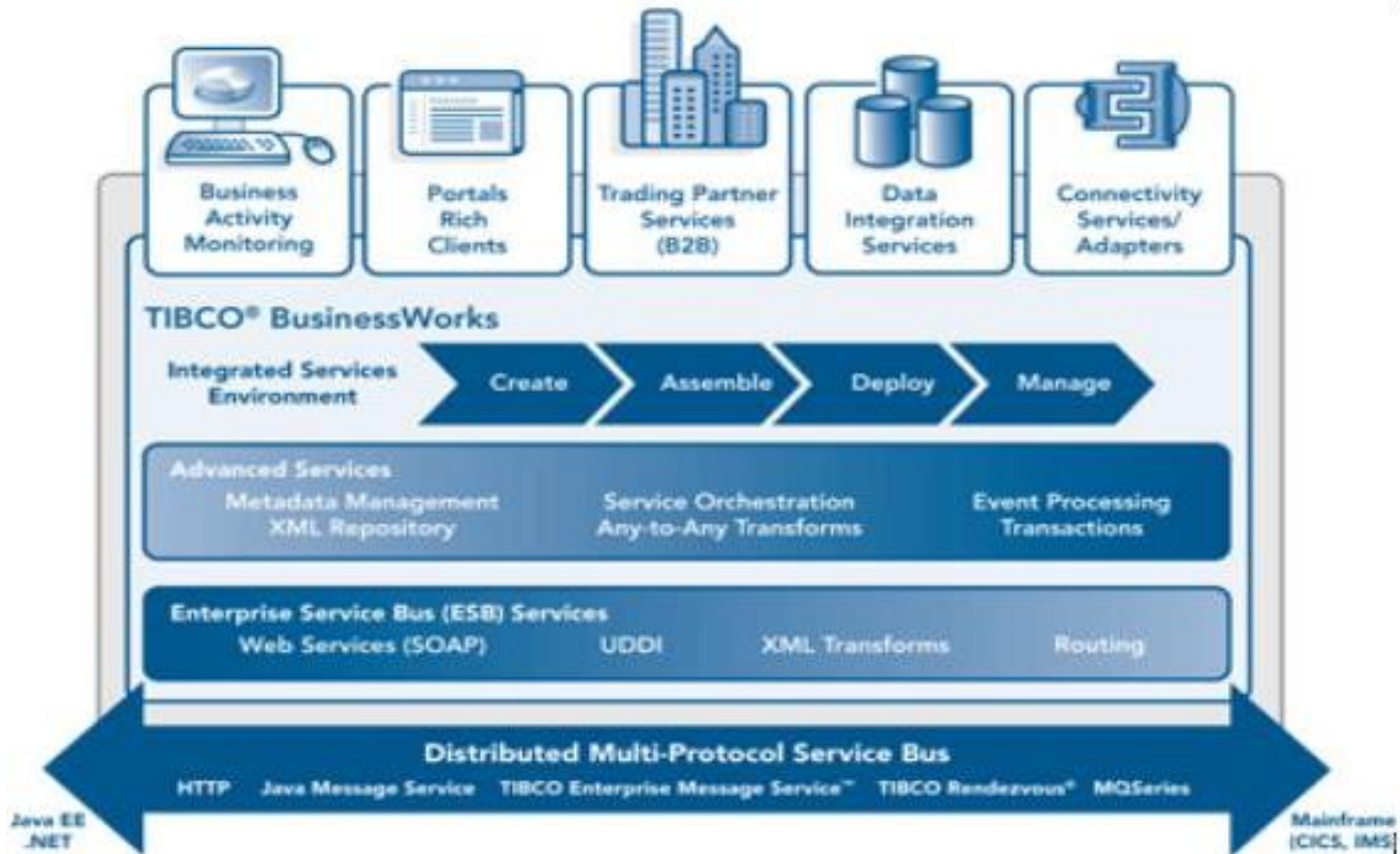- Enterprise service bus
- And many more….

- http://www.tibco.com/

TIBCO: message-oriented middleware

- In addition, TIBCO offers the message-oriented middleware product Rendezvous.

- TIBCO provides a message bus for enterprise application integration (EAI).

- TIBCO provides messaging APIs in C, C++, Java, Visual BASIC , Perl and .NET to receive data feeds on MS Excel spreadsheets and other applications of choice.

- Considerable "Enterprise" functionality is layered onto this.
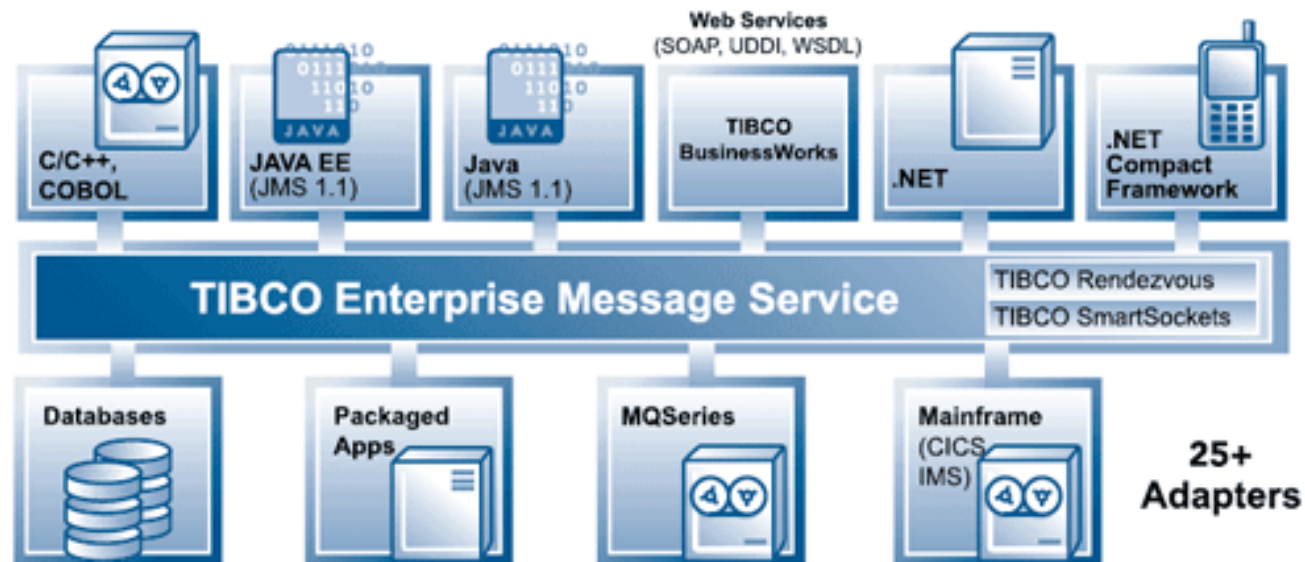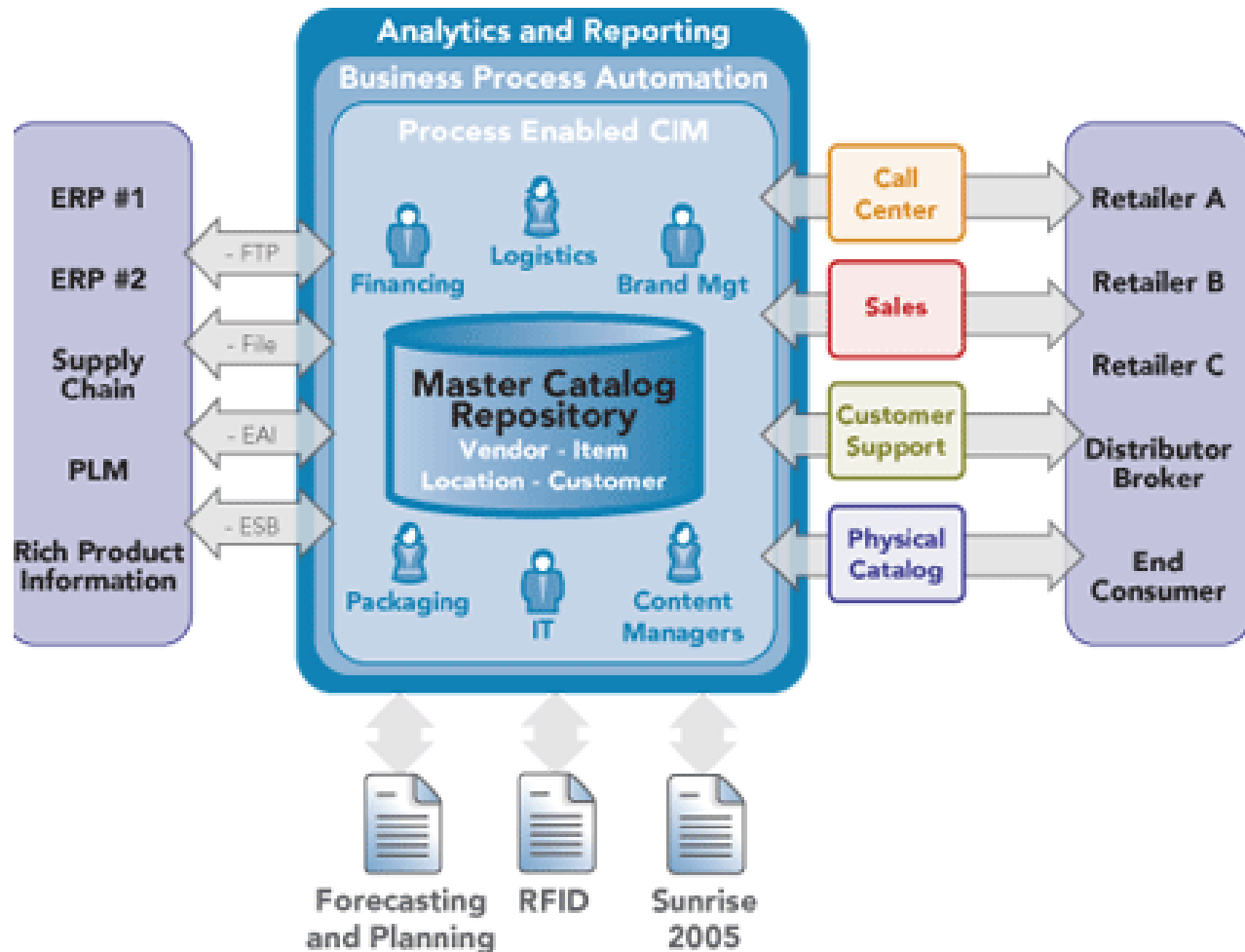
Enterprise Service Bus

TIBCO: Enterprise Service Bus

TIBCO: Enterprise Message Service

TIBCO: Business Process Automation

Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
  - Centralized Architectures
  - Decentralized Architectures
  - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
  - Wrappers
  - Interceptors
  - General Approaches to Adaptive Software
  - Discussion
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
  - The Feedback Control Model

Wrappers

- When building a distributed system out of existing components, we immediately bump into a fundamental problem: the interfaces offered by the legacy component are most likely not suitable for all applications.

- A **wrapper** or **adapter** is a special component that offers an interface acceptable to a client application, of which the functions are transformed into those available at the component. In essence, it solves the problem of incompatible interfaces.

- Facilitating a reduction of the number of wrappers is typically done through middleware. One way of doing this is implementing a so called **broker**, which is logically a centralized component that handles all the accesses between different applications.
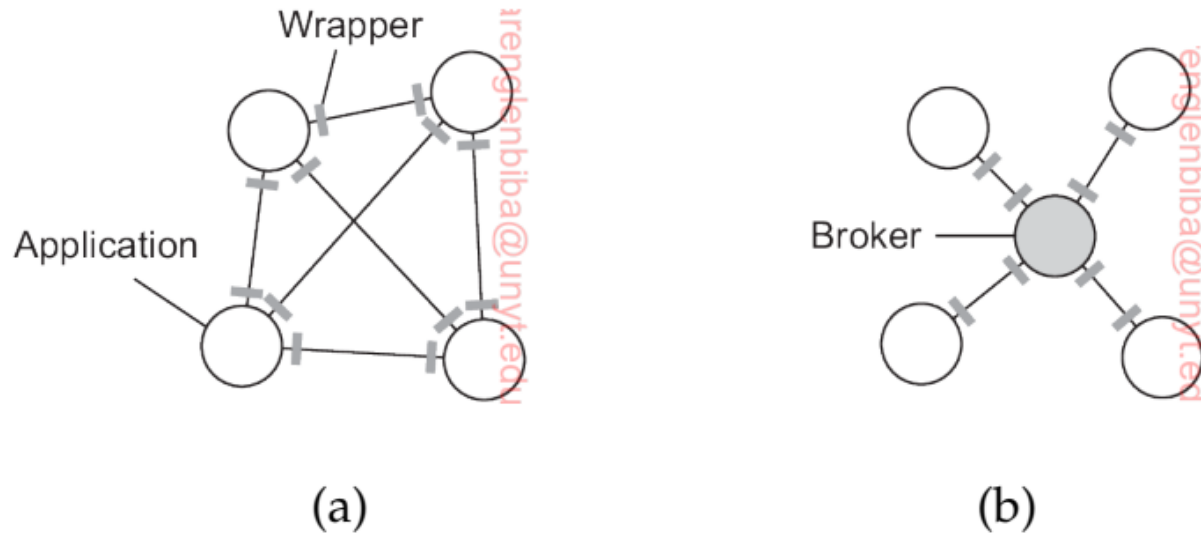
Wrappers

Figure 2.13: (a) Requiring each application to have a wrapper for each other application. (b) Reducing the number of wrappers by making use of a broker.

Architectures Vs. Middleware

- **Problem**
  - The chosen style may not be optimal in all cases.
  - Need to (dynamically) adapt the behavior of the middleware.
- **Interceptors**
  - Intercept the usual flow of control when invoking a remote object.

Adaptive Middleware

- Although middleware is meant to provide distribution transparency, it is generally felt that specific solutions should be adaptable to application requirements.

- One solution to this problem is to make several versions of a middleware system, where each version is tailored to a specific class of applications.

- An approach that is generally considered better is to make middleware systems such that they are easy to configure, adapt, and customize as needed by an application.

Interceptors

- Conceptually, an interceptor is nothing but a software construct that will break the usual flow of control and allow other (application specific) code to be executed.

- To make interceptors generic may require a substantial implementation effort, and it is unclear whether in such cases generality should be preferred over restricted applicability and simplicity.

- Also, in many cases having only limited interception facilities will improve management of the software and the distributed system as a whole.

Interceptors in Object-based DSs

- Imagine that object *B* is replicated.
- In that case, each replica should actually be invoked.
- This is a clear point where interception can help.
- What the request-level interceptor will do is simply call invoke(B, &do_something, value) for each of the replicas.
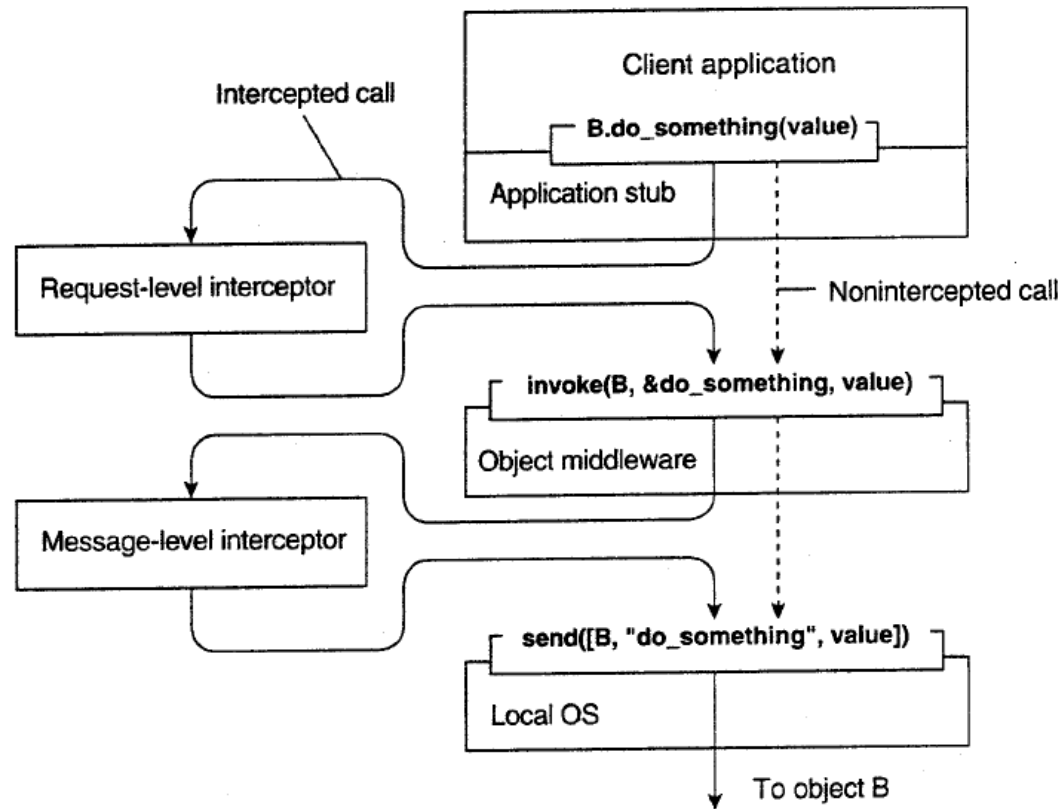- The beauty of this is that the object *A* need not be aware of the replication of *B*

Figure 2-15. Using interceptors to handle remote-object invocations.

Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
    - Centralized Architectures
    - Decentralized Architectures
    - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
    - Wrappers
    - Interceptors
    - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
    - The Feedback Control Model

Adaptive Software

- What interceptors actually offer is a means to adapt the middleware.

- The need for adaptation comes from the fact that the environment in which distributed applications are executed changes continuously.

- Rather than making applications responsible for reacting to changes, this task is placed in the middleware.

- These strong influences from the environment have brought many designers of middleware to consider the construction of adaptive software.

- However, adaptive software has not been as successful as anticipated.

Do we need Adaptive software?

- The underlying assumption is that we need adaptive software in the sense that the software should be allowed to change as the environment changes.
  - However, one should question whether adapting to a changing environment is a good reason to adopt changing the software.
  - Faulty hardware, security attacks, energy drainage, and so on, all seem to be environmental influences that can (and should) be anticipated by software.
- The strongest, and certainly most valid, argument for supporting adaptive software is that many distributed systems cannot be shut down.
- This constraint calls for solutions to replace and upgrade components on the fly, but is not clear whether any of the solutions proposed are the best ones to tackle this maintenance problem.

Lesson 2: Architectures

- ARCHITECTURAL STYLES
- SYSTEM ARCHITECTURES
  - Centralized Architectures
  - Decentralized Architectures
  - Hybrid Architectures
- ARCHITECTURES VERSUS MIDDLEWARE
  - Wrappers
  - Interceptors
  - General Approaches to Adaptive Software
- SELF-MANAGEMENT IN DISTRIBUTED SYSTEMS
  - The Feedback Control Model

Self-managing Distributed Systems

- When adaptation needs to be done automatically, we see a strong interplay between system architectures and software architectures.
  - On the one hand, we need to organize the components of a distributed system such that monitoring and adjustments can be done, while on the other hand we need to decide where the processes are to be executed that handle the adaptation.
- Explicit attention is dedicated to organizing distributed systems as high-level feedback-control systems allowing automatic adaptations to changes.
- This phenomenon is also known as autonomic computing

Autonomic Computing

- This name indicates the variety by which automatic adaptations are being captured:
  - self-managing
  - self-healing
  - self-configuring
  - self-optimizing
  - Self-*
- We resort simply to using the name self-managing systems as coverage of its many variants.
- In many cases, self-* systems are organized as a feedback control system.
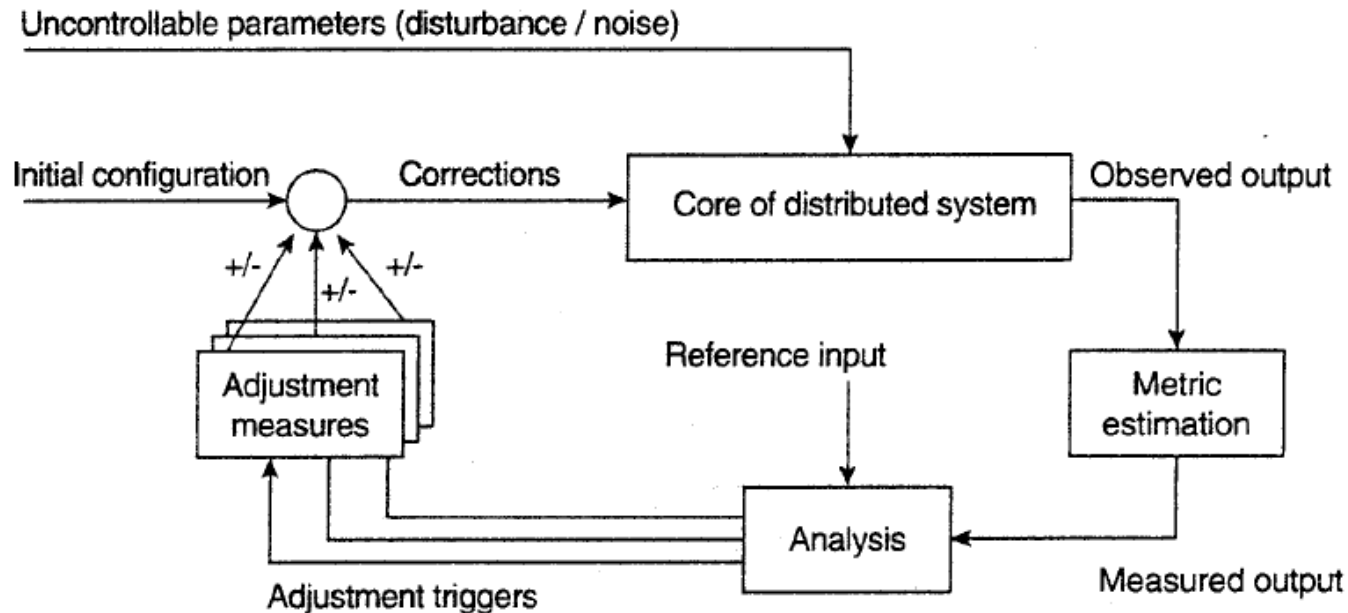
Feedback control system

Figure 2-16. The logical organization of a feedback control system.

This shows the logical organization of a self-managing system, and as such corresponds to what we have seen when discussing software architectures. However, the physical organization may be very different. For example, the analysis component may be fully distributed across the system.

End of Lesson 2

- Readings
  - Distributed Systems, Chapter 2.