Distributed Systems

I.Fundamental concepts of Distributed Systems

1. Introduction of Distributed Systems
2. Distributed Systems Architectures
3. Protocols
4. Client/Server Model

Introduction to Distributed Systems

DEFINITION

A distributed system is a collection of autonomous computers connected by a network and equipped with a distributed software system (FOROUZAN)

A distributed system is a collection of independent computers that appears to the user to be a single computer (TANENBAUM)

Dağıtılmış sistem, kullanıcıya tek bir bilgisayar gibi görünen bağımsız bilgisayarlar topluluğudur.

MOTIVATION

Technological advancements

Invention of high-speed computer networks (70s):

- Local Area Network (LAN)
- Global network (Wide Area Network - WAN)

Development of powerful microprocessors (80s)

- Intel 80386 processor (i386)
- Intel 80486 Processor (i486)

STATE OF ART

It is relatively easy to group together many CPUs, connecting them together over a high-speed network.

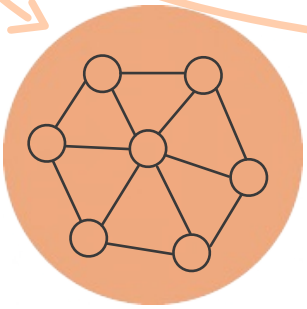
Software for distributed systems is completely different from software for centralized systems.

use client/server architecture where one or more client nodes are directly connected to a central server.
→client sends a request to a company server and receives the response

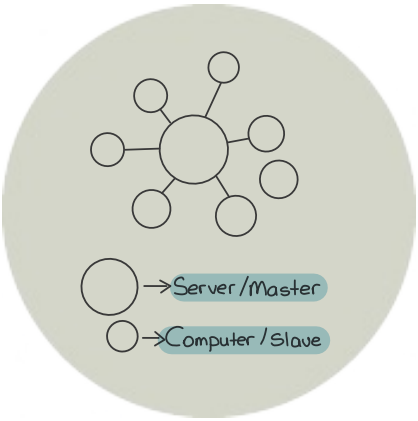
BENEFITS

Distributed Systems vs. Centralized Systems

- Best cost/benefit ratio
- Processing capacity beyond the practical limits of Centralized Systems
- Greater domain of applications
- Greater reliability and availability
- Gradual growth in processing capacity



Every node make its own decision. The final behavior of the system is the aggregate of the decisions of the individual nodes.
⇒There is no single entity that receives and responds to request



Distributed Systems vs. Independent Computers

- Sharing common data between users;
- Sharing hardware and software resources;
- Communication between people;
- Flexibility in the distribution of tasks according to applications.

DISADVANTAGES

- Lack of proper software
- Failures and saturation of the communication network can eliminate the advantages of Distributed Systems
- Security can be compromised with easy access to reserved data and resources

BASIC CHARACTERISTICS

1. Resource sharing
2. Extensibility
3. Competition
4. Scalability (smooth gradual growth)
5. Fault Tolerance
6. Transparency

1. RESOURCE SHARING

Hardware components

- Disks, printers, etc.

Software components

- Files, databases, etc.

Basic models

- Client/server model
- Object-based model

2. EXTENSIBILITY

Hardware Extensions

- Peripherals, memory, interfaces, etc.

Software Extensions

- OS functions, communication protocols, etc.

Key interfaces are public (system calls)

Uniform inter-process communication mechanism

3. COMPETITION

More than one process running at any given time

- Separate user activities
- resource independence
- Location of server processes on different computers

Concurrent access to shared resources requires synchronization

4. SCALABILITY

Amount of work involved in processing any request for access to a shared resource

- Regardless of network size

Techniques

- Replication, caching, multiple servers, etc.

5. FAULT TOLERANCE

Hardware and software failures (in CPUs and networks):

- Programs stop or produce wrong results

Approaches

Hardware redundancy

- Database replicated on multiple servers

Software recovery

- Keep permanent data always consistent

6. TRANSPARENCY

Hiding the separation of components in a distributed system from the user and the application programmer

- Seeks that this is seen as a centralized system

Forms of Transparency

- Access, localization, concurrency, replication, failure, migration, performance and scale

EXAMPLES OF DISTRIBUTED SYSTEMS

- Internet
- intranet
- SETI@home
- vCluster
- Amazon Web Services

INTERNET

Large collection of different interconnected computer networks

Communication through message exchange

Services

- www, ftp, mail, voip, etc.

INTRANET

Part of the Internet administered separately

Local security policies

Needs:

- file sharing service
- Firewalls for protection
- Ease of installation
- software support

SETI@HOME

Search for Extra-terrestrial Intelligence at Home

Objective

- Using the processing power of hundreds of thousands of computers connected to the Internet in the search for extra-terrestrial intelligence

VCLUSTER

Developed by CPAD

Objective

- Underutilization of computational resources in academic laboratories
- Scientific applications with demand for computational resources

Focus

- Local user has priority over running tasks on the machine

AMAZON WEB SERVICES

Amazon web services solution

Seeks to offer web services with:

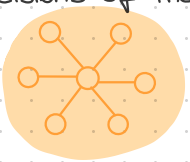
- Low cost
- Agility
- Flexibility
- Safety

Uses cloud computing concepts

Definition: A DS is a collection of independent computers that appears to the user to be a single computer
→ DS, kullanıcıya tek bir bilgisayar gibi görünen bağımsız bilgisayarlar topluluğudur.

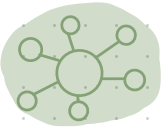
Distributed Systems: Every node make its own decision. The final behavior of the system is the aggregate of the decisions of the individual nodes.

- there is no single entity that receives and responds to the request



Centralized System: Use client/server architecture where one or more client nodes are directly connected to a central server

- client sends a request to a company server and receives the respond



Disadvantagem: → lack of proper software
→ security can be compromised with easy access to reserved data and resources

Basic Characteristic:

1. Resource Sharing
   - Hardware components
     - Disk, printers
   - Software components
     - Files, databases
   - Basic models
     - Client/server model
     - Object-based model

2. Extensibility
   - Hardware extensions
     - peripherals, memory, interfaces
   - Software Extensions
     - OS functions, communication protocols
   - Key interfaces are public (system calls)
     - uniform inter-process communication mechanism

3. Competition
   → more than one process running at any giving time
     - separate user activities
     - resource independence
     - location of server processes on different computers
   → concurrent access to shared resources requires syncronization

4. Scability
→ Amount of work involved in processing any request for access to shared resources
  • regardles of network size
- Tecniques
  • replication, caching, multiple servers
-
-

5. Fault Tolerance
→ Hardware and software failures (in CPUs and networks)
  • program stop or produce wrong results
- Approaches
  - Hardware redundancy
    • database replicated on multiple servers
  - Software recovery
    • keep permanent data always consistent
.
.

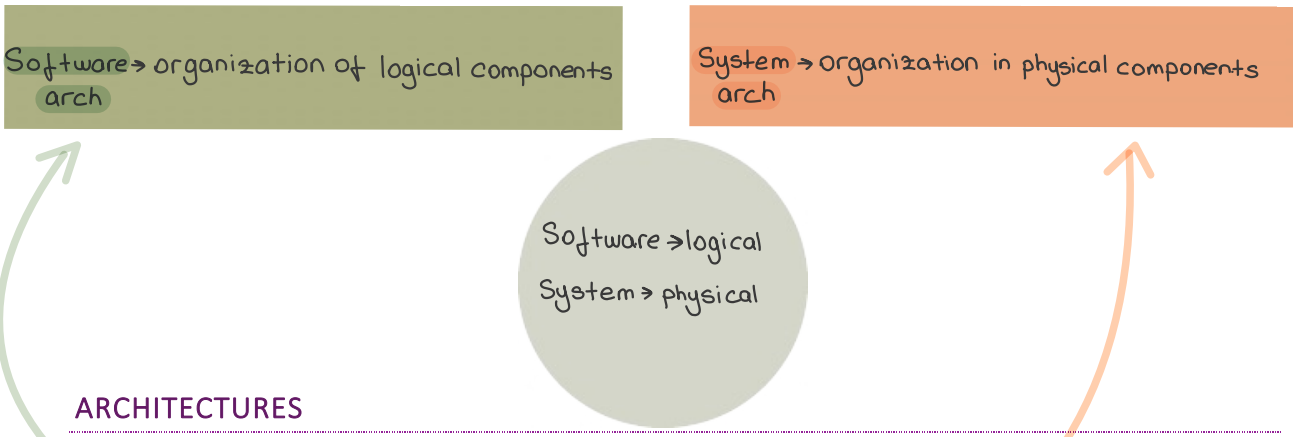
6. Transparency
→ Hiding the separation of components in a distributed system from the user and the application
  • seeks that this is seen as a centralized system
→ Forms of transparency
  • access, localization, concurrency, replication, failure, migration

Distributed System

Architectures

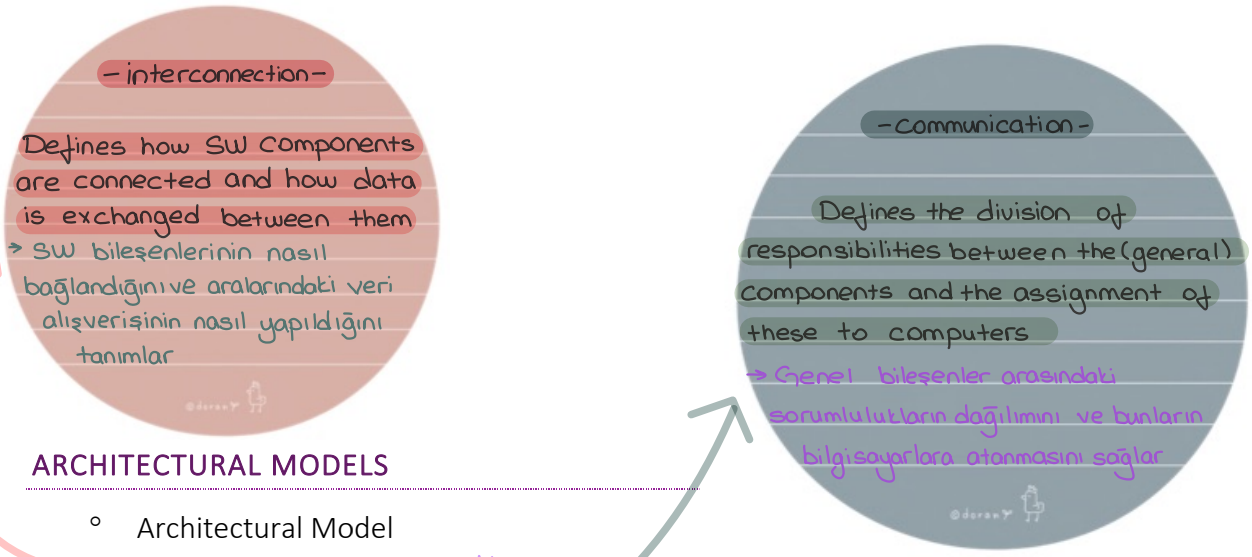
GOALS

- Understand possibilities of organizing SD components in architectures
- Understand the advantages and disadvantages of architectures
- Become familiar with most common models of SDs



ARCHITECTURES

- **Architecture**: organization and interaction of components
- **Software architecture** vs. **system architecture**
  - Software: organization of logical components
  - System: organization in physical components
- System architecture depends on software architecture



ARCHITECTURAL MODELS

- Architectural Model
- **Interconnection**:
  - Defines how SW (software) components are connected and how data is exchanged between them.
- **Communication**:
  - Defines the division of responsibilities between the (general) components and the assignment of these to computers.

ARCHITECTURAL MODEL REGARDING INTERCONNECTION

Defines software architectures

- Defines the logical organization of distributed systems into software components.
- Style is defined from components:
  - How these components are connected to each other
  - Of the data exchanged between them
  - The way they are set up in sets to form a system

interface

Component is a modular unit with well-defined interfaces

As it is very replaceable, it is important that we respect its interfaces.

- That is why the concept of Connector was created.

ara bulucu

Connector is a mediator of communication or cooperation between components.

mediator

- Defines the format of the integration
- Facilitates communication

The most important models are

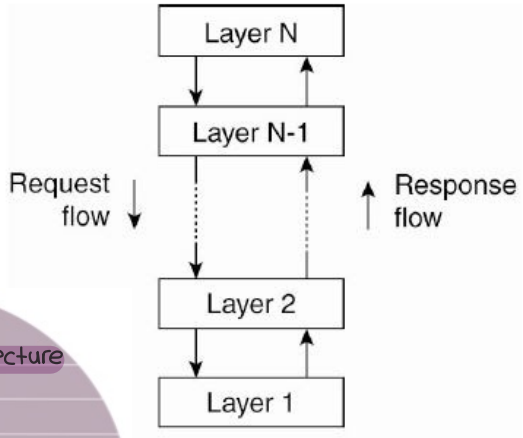
1. Layered Architecture
2. Object-Based Architecture
3. Event-Based Architecture
4. Architecture Based on Shared Data

1. LAYERED ARCHITECTURE

The basic idea is to divide application components into several logical layers.

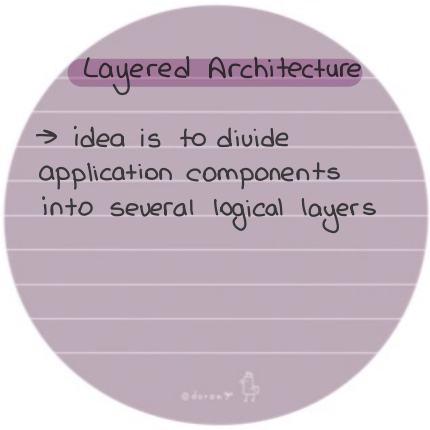
- A component at layer N will make a request to layer N-1, but usually not the other way around.
- Each layer is geared towards addressing some specific purpose.

Operation



Application Example

- WEB application using AJAX

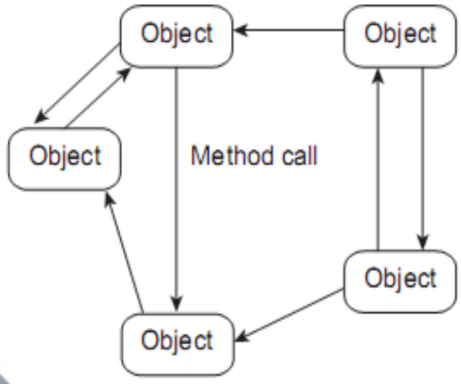


2. OBJECT BASED ARCHITECTURE

Each object corresponds to a component

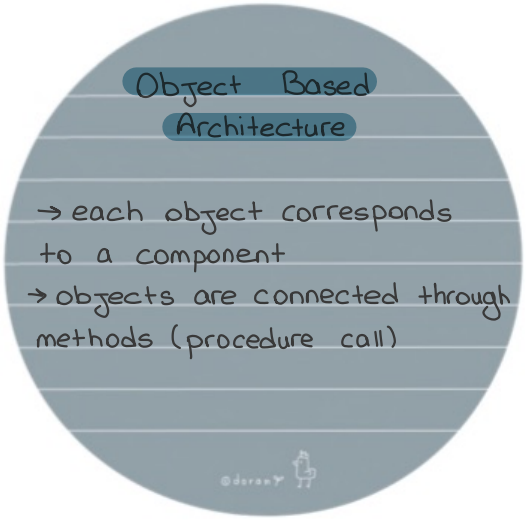
Objects are connected through procedure calls (methods)

Operation



Application Example

- Java RMI (Remote Method Invocation) application



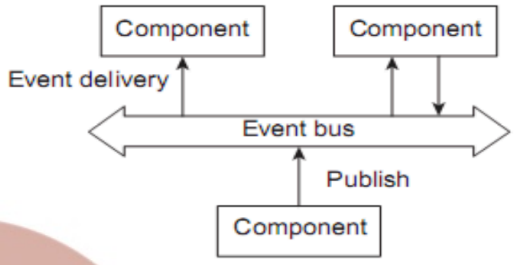
3. EVENT BASED ARCHITECTURE

Components communicate through event propagation and can optionally carry data.

Are based on publish/subscribe systems.

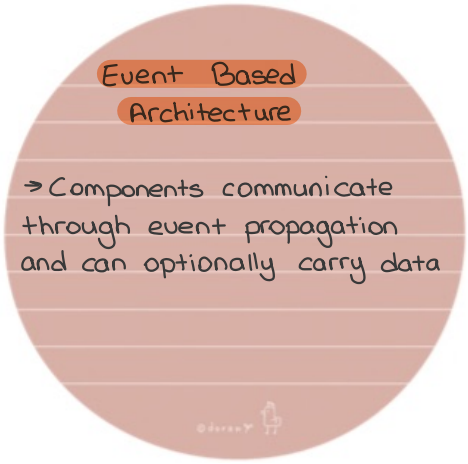
- Components publish events and only those who subscribe (subscribers' idea) to these events will receive them.

Operation



Application Example

- Systems with multiple sensors



4. ARCHITECTURE BASED ON SHARED DATA

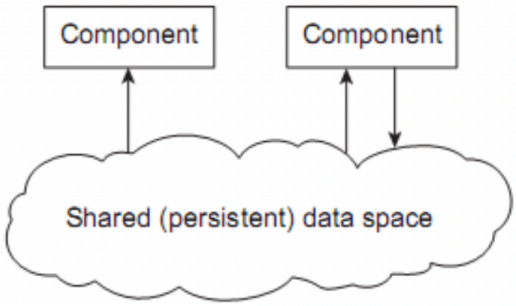
The advantage of this architecture is that all components are loosely coupled

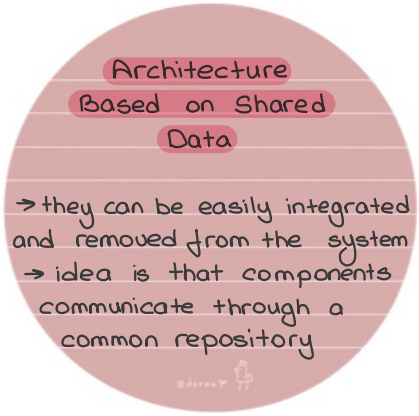
- In other words: they can be easily integrated and removed from the system.
- This is because it does not have an explicit reference to another system.

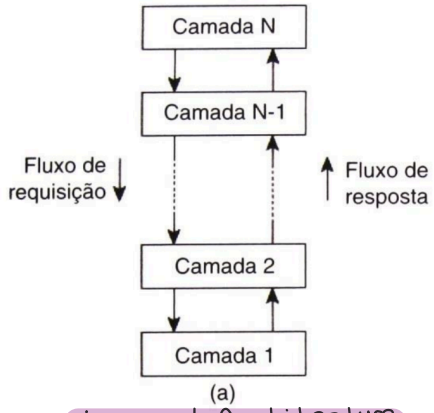
The idea is that components communicate through a common repository

- Repository can be passive or active.

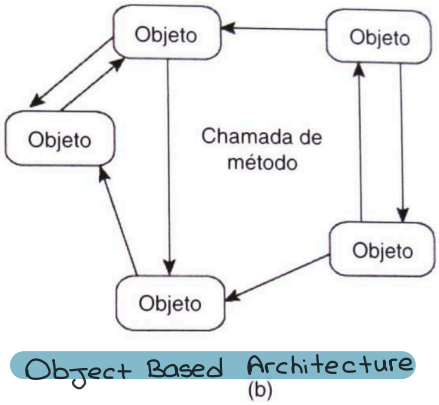
Operation





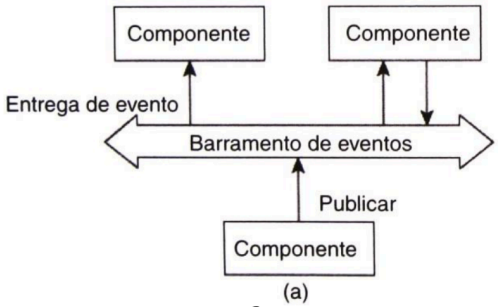


Layered Architecture

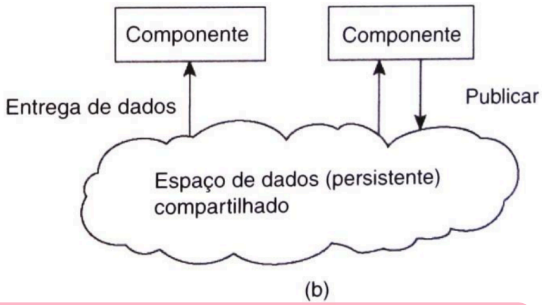


Object Based Architecture

Figura 2.1 Estilo arquitetônico (a) em camadas e (b) baseado em objetos.



Event Based Architecture



Architecture Based on Shared Data

Figura 2.2 Estilo arquitetônico (a) baseado em eventos e (b) de espaço de dados compartilhado.

ARCHITECTURE MODEL REGARDING COMMUNICATION

It defines the logical structure of process communication and its arrangement on computers on a network.

- Defines the distribution of responsibilities and the assignment of processes to computers.

Each process has well-defined responsibilities and interact to carry out a useful activity

Two existing basic models

- Client-Server (Centralized)
- Peer to Peer (Decentralized)

There is a model with centralized + decentralized communication

- Hybrid

1. CENTRALIZED ARCHITECTURE

Server:

- A process waits for requests from other processes (on other computers)
- When accepting an order, perform a service and respond appropriately.

Client:

- Order Requester. Requires an operation on the server.

Servers can, in turn, be clients of other servers.

DIVIDING RESPONSIBILITIES

What stays on the client, what stays on the server?

Let's consider a 3-tier application:

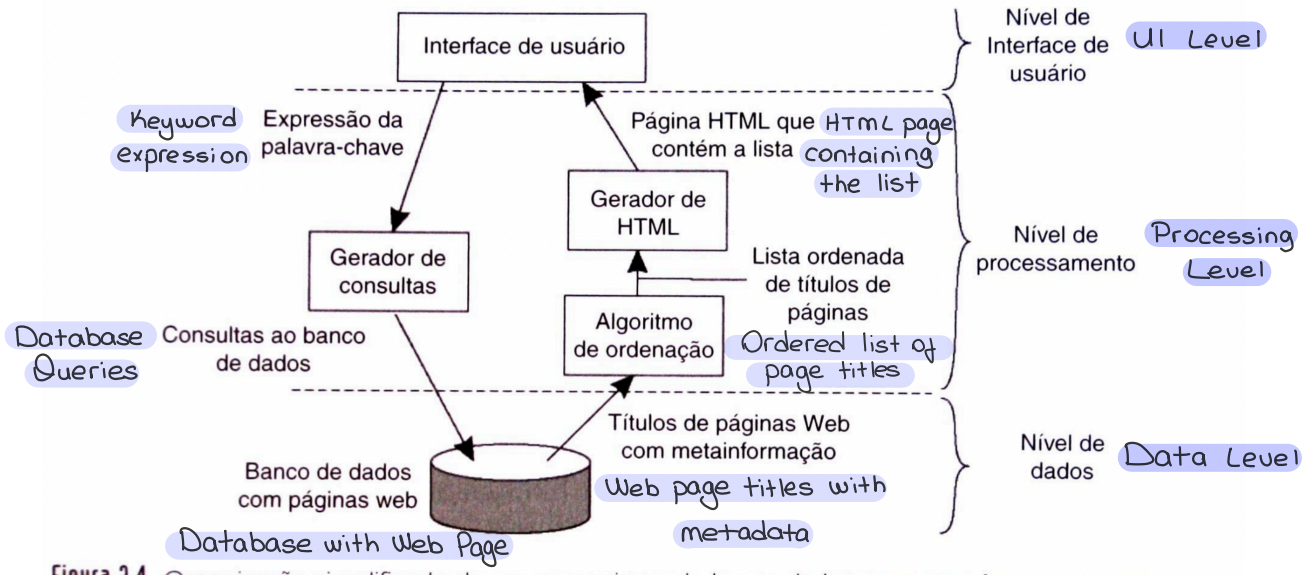


Figura 2.4 Organização simplificada de um mecanismo de busca da Internet em três camadas diferentes.

COMMENT ON WHO DOES WHAT

Typically, thin clients facilitate system management

- The functionality to be updated is on the server

Fat customers favor scalability

- Ex.: The Gmail Revolution

HOW TO DIVIDE?

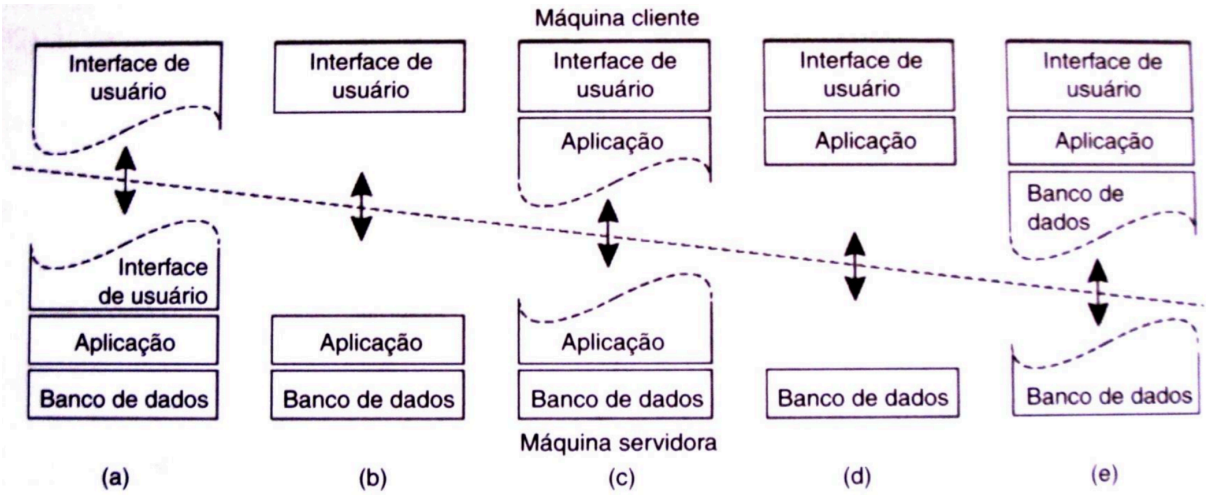
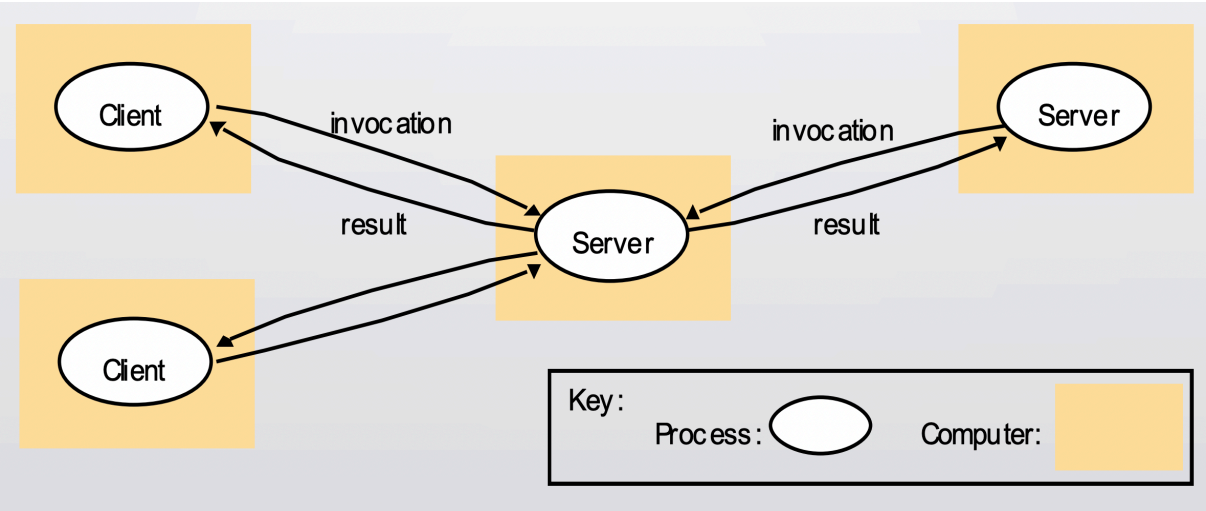


Figura 2.5 Alternativas de organizações cliente–servidor (a)—(e)

CLIENT-SERVER ARCHITECTURE



Client/Server Architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client.

=> it has one ore more client computers connected to a central server over a network or internet connection

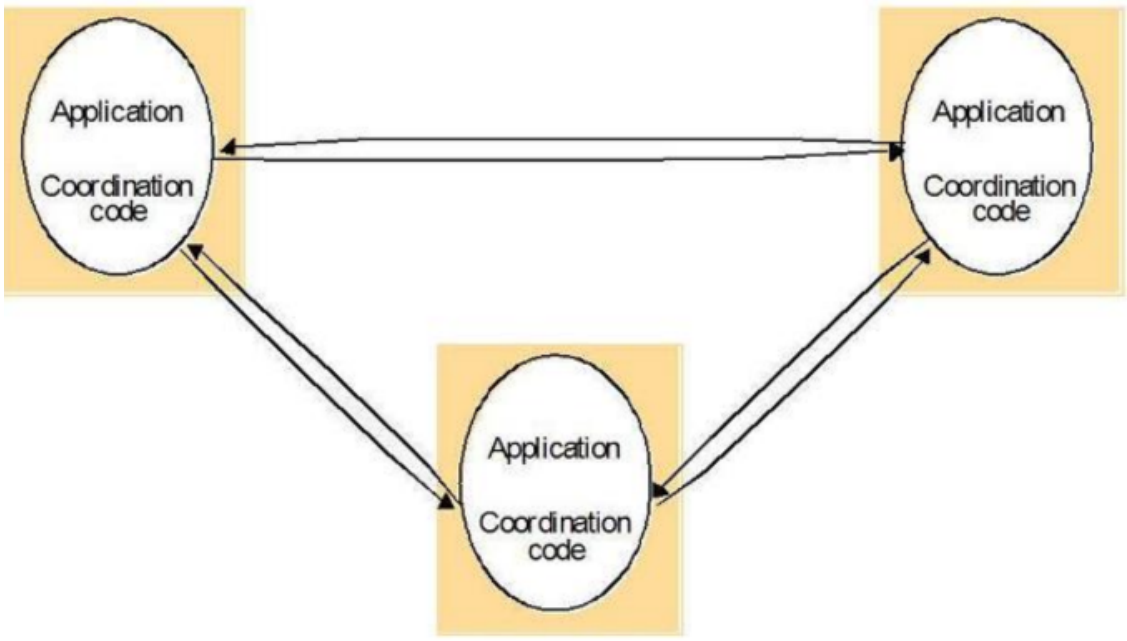
=> all the request and services are delivered over a network

2. DECENTRALIZED ARCHITECTURE (P2P) - peer-to-peer

Processes are all the same:

- All processes involved in a task perform similar functions, interacting cooperatively as peers
- No distinction between client and server.

Scalable architecture – not centralized



Formed by a set of nodes, organized in an overlay or overlay network

- Overlay: network in which nodes are processes and links represent possible communication channels

Communication cannot be done directly

- Structured or unstructured topologies

2.1 STRUCTURED TOPOLOGIES

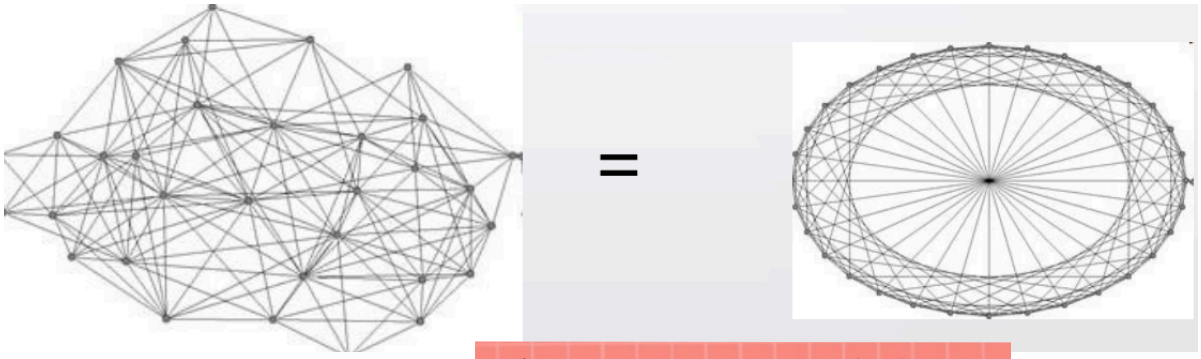
Topology reflects a data structure considering who has what resource
⇒ Topoloji, kimin hangi kaynaklara sahip olduğunu göz önünde bulundurarak bir veri yapısı yansıtır

Principle: deterministic procedure (pseudo-random)

- To maintain the topology
- To discover resources

Topology reflects a data structure considering who has what resource

- Most common structure is a distributed Hash Table: **DHT**
- Another example is structuring nodes in a distributed tree



The core of a DHT is a hash table. Key-value pairs are stored in DHT and a value can be looked up with a key. The keys are unique identifiers to values that can range from blocks in a blockchain to addresses and to documents.

UNDERSTANDING DHTS

Resources and nodes receive random identifiers

The resource identifier space is divided between the nodes according to the node identifiers

What differentiates a DHT from a normal hash table is the fact that storage and lookup on DHT are distributed across multiple (can be millions) nodes or machines

EXAMPLE: CHORD

Nodes are arranged in a ring

Item k is mapped to node with lowest id with id ≥ k

2.2 UNSTRUCTURED TOPOLOGIES

→ Based on random algorithms

Based on random algorithms:

- Each node gets a list of random neighbors
- When it needs a resource, this node asks its neighbors who has

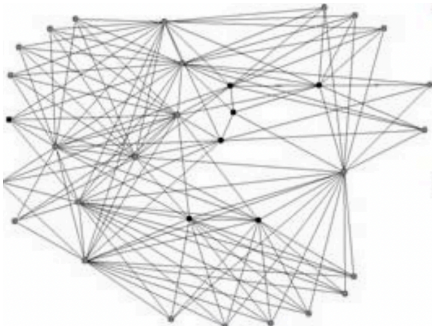
- each node gets a list of random neighbors
- when its needs a resource, this node asks its neighbors who has

How to find the data?

- Flood the network with a search (flooding)

- flood the network with a search → flooding

Example of unstructured topologies



BitTorrent

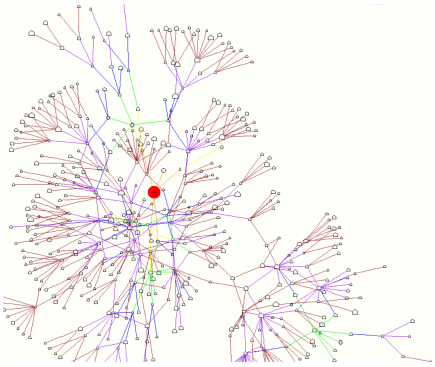
- Each node receives a random list of neighbours and connects to them
- Nodes exchange parts of the file with their neighbours

Gnutella v1.0

Nodes know one or a few nodes to join the network

We announce ourselves to everyone who wants to hear

Each node maintains a random list of acquaintances



SUPERNODES (SUPER PAIRS)

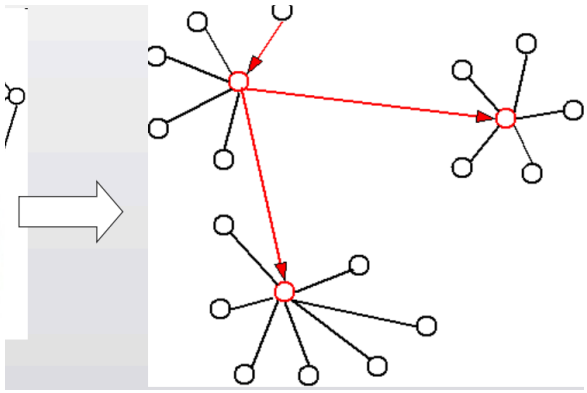
As the network grows, locating data items in unstructured P2P systems can be problematic

If resources need to be frequently located on the network, not structuring hinders **scalability**

One solution is to create indexes

- A supernode is responsible for a set of nodes or resources
- A node requests a resource from a supernode
- Resource discovery is restricted to supernodes

– supernode –
in peer-to-peer networking, a supernode is any node that also serves as one of that network's relayers and proxy servers, handling data flow and connections for other users. This semi-distributed architecture allows data to be decentralized without requiring excessive overhead at every node



CENTRALIZED VS. DECENTRALIZED

Centralization:

- *Simplicity of implementation*
- *Management simplicity*
- *Potential bottleneck*

Decentralization:

- *Scalability*
- *Robustness*
- *Complexity*

HYBRID COMMUNICATION ARCHITECTURE

- Centralized directory, distributed function
- Napster, MSN, Skype, etc.
- Directory is simple and reasonably scalable if centralized

Software Architecture: Organization of logical components

System Architecture: Organization in physical components

System architecture depends on Software architecture

Architectural Models

↳ **Interconnection:** Defines how SW components are connected and how data is exchange between them

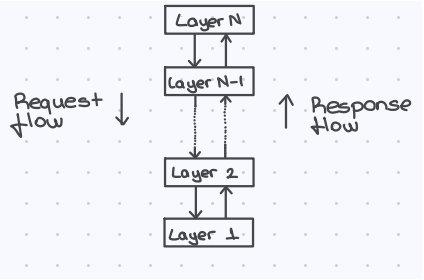
↳ **Communication:** Defines the division of responsibilities between the (general) components and the assignment of these to computers

Component: It is a modular unit with well-defined interfaces
→ it is very replaceable

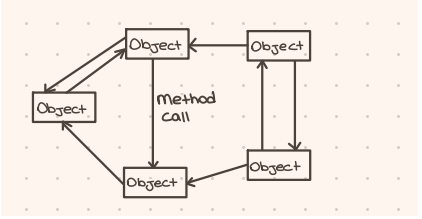
Connector: It is a mediator of communication or coorperation between components

The most important models
1. Layered Architecture
2. Object-based Architecture
3. Event-based Architecture
4. Architecture Based on Shared Data

→ **1. Layered Architecture**
- → Idea is to divide application components into several logical layers.
  - a component at layer N will make a request to layer N-1, but usually not the other way around
- → Ex: WEB application using AJAX



→ **2. Object-based Architecture**
- → Each object corresponds to a component
- → Objects are connected through procedure calls (methods)
  - call can be local or remote
- → Ex: Java RMI application

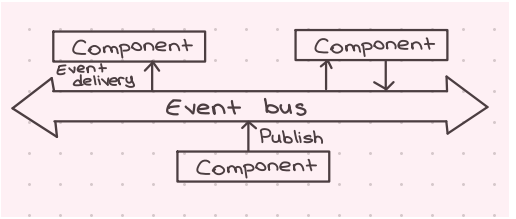


→ 3. Event-based Architecture

→ Components communicate through event propagation and can optionally carry data

→ Are based on publish/subscribe system

- components publish events and only those who subscribe to these events will receive them

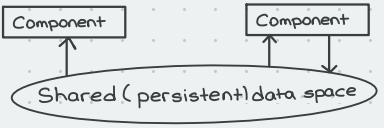


→ 4. Architecture Based on Shared Data

→ All components are loosely coupled (gevşek bağlanmış) } advantagem

- they can be easily integrated and removed from the system

→ The idea is that components communicate through a common repository



Architecture Model Regarding Communication: It defines the logical structur of process communication and its arrangament on computers on a network.

- Defines the distribution of responsabilities and the assignment of process to computers

→ client-server (Centralized)

→ Peer-to-peer (Decentralized)

→ Hybrid (Centralized + Decentralized)

1. Centralized Architecture

Server: → A process waits for requests from other processes
→ When accepting an order, perform a service and respond appropriately

Client: → Order Requester. Requires an operation on the server.

→ Client/Server Architecture: It is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client.

- it has one or more client computers connected to a central server over a network connection
- all the request and services are delivered over a network

2. Decentralized Architecture

→ Processes are all the same

- all processes involved in a task perform similar functions, interacting cooperatively as peers
- no distinction between client and server

→ Formed by a set of nodes, organized in an overlay network

→ Communication can not be done directly

- structured or unstructured

2.1 Structured Topologies

→ Deterministic procedure (pseudo-random)
- to maintain the topology
- to discover resources

→ Topology reflects a data source considering who has what resource
- Distributed Hash Table (DHT) → most common structure

DHT: → key-value pairs are stored in DHT
→ value can be looked up with a key → unique identifiers

2.2 Unstructed Topologies

→ Based on random algorithms
- each node gets a list of random neighbors
- when it needs a resource, this node asks its neighbors who has

→ flooding

Supernodes (Super Pairs) → allows data to be decentralized without requiring excessive overhead at every node

Centralized	vs	Decentralized
• simplicity of implementation		• ecability
• management simplicity		• robustness
• potential bottleneck		• complexity

Hybrid Communication Architecture

→ centralized directory, distributed function
- napster, msn, skype

→ Directory is simple and reasonably → if centralized

Protocols

Protocols - rules that the processes that are communicating must follow

- Level protocols
- The client-server model
- Remote Procedure Call (RPC)
- Group communication

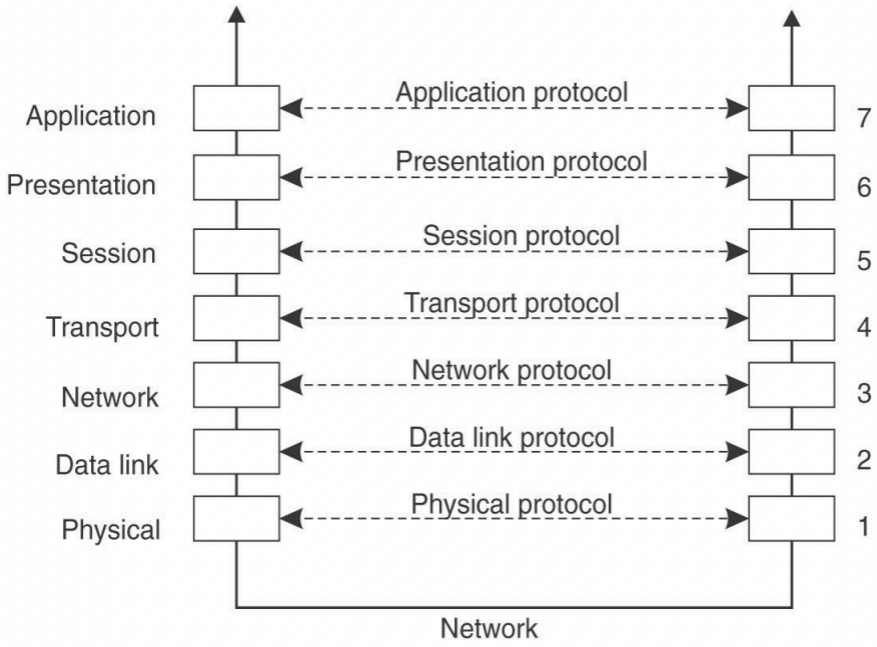
LAYER PROTOCOLS

- As there is no shared memory, all communication on SDs takes place through message exchange
- What is the meaning of the bits sent? What voltage is used to signal 0 and 1? How do you detect the final bit of the message, or that a message has been damaged or lost?
- The International Standards Organization (ISO) has developed a reference model for the interconnection of open systems (OSI)

- An open system can communicate with any other open system using the 1983 OSI model protocols
- Abstract model of networks
- In order for a group of computers to communicate on a network, they must all use the same communication protocols

- Connection-oriented protocols (phone call)
- Connectionless protocols (letter)
- Process creates message that when going through the various layers of protocols is broken up and has headers added to it

- Communication division into 7 layers
- Layering makes your implementation more flexible, facilitating updates and fixes
- Networks do not need to implement all layers
- Each layer performs well-defined function, and they are defined to minimize communication between them



Connection-Oriented Protocols: TCP is an example of of a connection-oriented protocol.
→ It requires a logical connection to be establish between two processes before data is exchanged.
→ The connection must be maintained during the entire time.
⇒ this process is much like a phone call
- the caller must knows the person's telephone number and the phone must be answered

Connectionless Protocol: Allows data to be exchanged without setting up link between processes
→ Each unique of data can travel over different paths to reach the final destination.
→ Some data packets might be lost or might arrive out of sequence to other data packets
→ UDP is a connectionless protocol
- it is similar to sending a letter where you don't acknowledge receipt

STRUCTURE

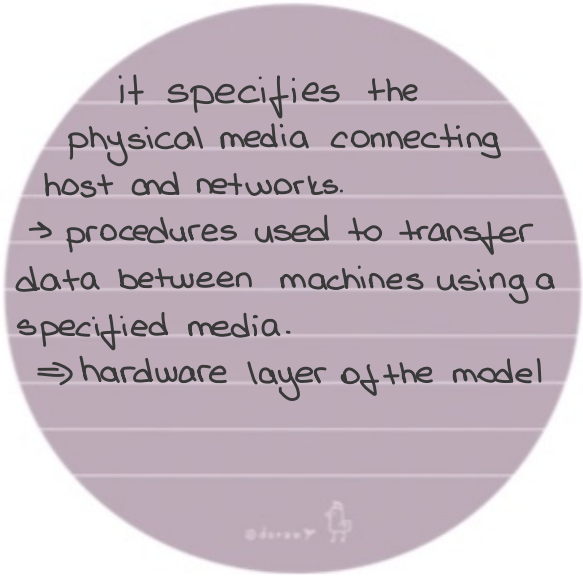
- Layer provides services to upper layer
- Layer uses lower layer services
- Layers at the same level “communicate”
- A layer only takes cognizance of the lower layer
- Interaction between layers made by services
- Division of tasks
- Facilitates abstraction

LAYERS

1. Physical Layer
2. Data Link Layer
3. Network Layer
4. Transport Layer
5. Session Layer
6. Presentation
7. Application

1. PHYSICAL LAYER

- Transmission of bit sequences over the physical medium
- Specific
  - Voltage and currents
  - Timing
  - Connectors and Pins
  - Physical means used
  - Electronic and mechanical aspects
- Domain of electronic engineering
- Does not address transmission error correction



2. LINK LAYER

- Organize bit sequences into sets of bits called frames
- Recognizes start and end of frames
- Detects lost frames and requests retransmission

it manages the reliable delivery of data across the physical network
→ it provides the abstraction of a reliable connection over the potentially unreliable physicall layer

3. NETWORK LAYER

- Forward information from source to destination (routing)
- Controls transmission flow between subnets (congestion control)
- Accounting functions
- Establish a single addressing scheme independent of the subnet used
- Allows connection of heterogeneous subnets

it is responsible for routing machine
→ it determines the path a transmition must take, based upon the destination machine's address.
→ it must respond to network congestion problems

4. TRANSPORT LAYER

- Divides and reassembles binary information into packets
- Ensures the sequence of packages
- Ensures reliable connection between source and destination of communication
- First layer that establishes source-destination communication

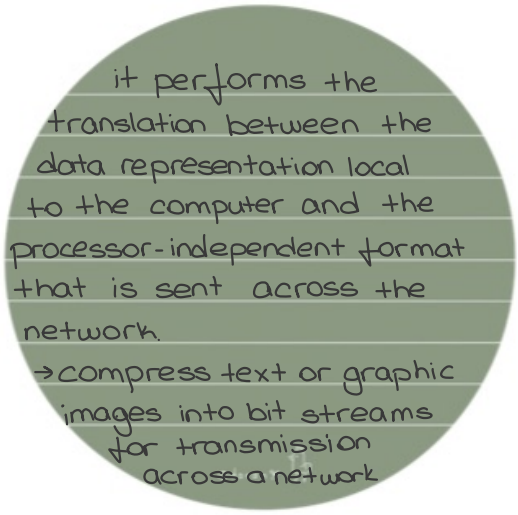
it provides end-to-end sequenced delivery of data.
→ it is the lowest layer that provides applications and higher layers with end-to-end service.
→ it hides the topology and characteristics of the underlying network from users

5. SESSION LAYER

- Manages communication sessions
- Session is a communication that needs to store states
- States are stored to allow re-establishment of communication in case of communication failure
    - Ex: Resume file transfers

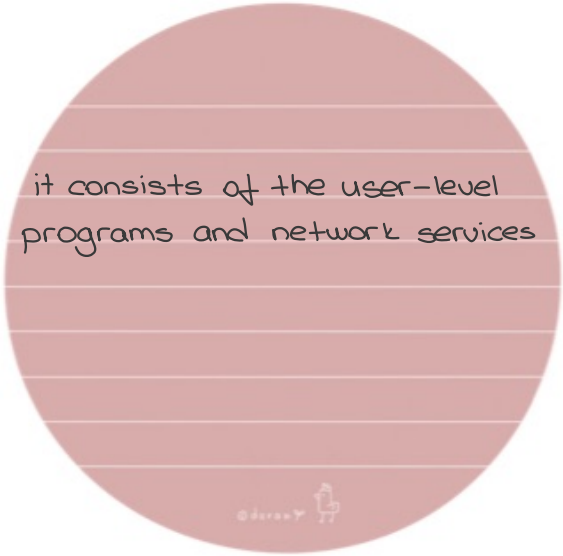
6. PRESENTATION LAYER

- Handles high-level data representation
- Adoption of standardized character representation system
- Adoption of standard numeric representation codes
- Data compression
- Data encoding

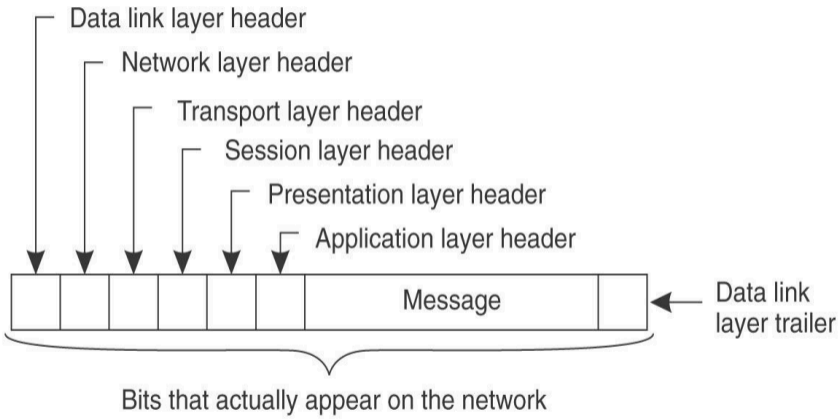


7. APPLICATION LAYER

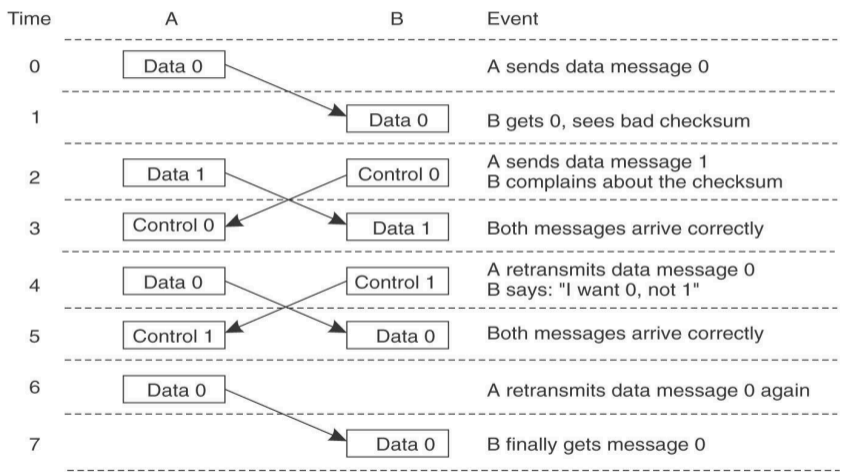
- Applications that offer services to the end user
- Unification of file systems and directories
- Examples
    - E-mail
    - Remote login
    - File transfer
    - Remote execution



NETWORK MESSAGE



LINK LEVEL COMMUNICATION



Layer Protocol

→ As there is no shared memory, all communication on DS done through message exchange
→ An open system can communicate with any other open system using OSI model protocols
→ To communicate on a network, they must all use the same protocols
→ Layers makes implementation more flexible, facilitating updates and fixes → advantage
→ Network do not need to implement all layers

Connection-oriented Protocol

→ Phone call
→ TCP is an example of of a connection-oriented protocol
→ It requires a logical connection to be establish between two processes before data is exchanged.
→ The connection must be maintained during the entire time.
→ The caller must knows the person's telephone number and the phone must be answered

Connectionless Protocol

Allows data to be exchanged without setting up link between processes
→ Each unique of data can travel over different paths to reach the final destination.
→ Some data packets might be lost or might arrive out of sequence to other data packets
→ UDP is a connectionless protocol
  - it is similar to sending a letter where you don't acknowledge receipt

1. Physical Layer

→ It specifies the physical media connecting host and networks
→ procedures used to transfer data between machines using a specified media
  ⇒ hardware layer of the model

2. Link Layer

→ It manages the reliable delivery of data across the physical network
→ It provides the abstraction of a reliable connection over the potentially unreliable physicall layer

3 Network Layer

→ It is responsible for routing machine
→ It determines the path a transmition must take, based upon the destination machine's address.
→ It must respond to network congestion problems

4. Transport Layer

→ It provides end-to-end sequenced delivery of data.
→ It is the lowest layer that provides applications and higher layers with end-to-end service.
→ It hides the topology and characteristics of the underlying network from users

5. Session Layer

→ It manages sessions between cooperating applications

6. Presentation Layer

→ It performs the translation between the data representation local to the computer and the processor-independent format that is sent across the network.
→ Compress text or graphic images into bit streams for transmission across a network

7. Application Layer

→ It consists of the user-level programs and network services

Client-Server Model

THE CLIENT-SERVER MODEL

- The model says how machines can communicate, but what is the structure of the SD?
- Classify machines into servers, which provide services to client machines
- Machine can run multiple client and/or server processes
- Eliminates overhead by implementing a request/reply protocol
- Simplicity and efficiency

ADDRESSING

- How to send the message to the server machine and then to the server process?
- Address must also include the case number (encoded in the customer program)
- Eliminates ambiguity, does not need a global coordinator generating process numbers, but it is not transparent
- What to do when the server goes down? Machine replacement does not work

- Each process can have its own address. Requires a centralized process address allocator. It doesn't have good scalability
- Processes choose their own addresses randomly from a large number space.
- Good scalability. Processes locate others through broadcast messages. address cache
- Extra messages are posted on the network

- Another option is to use name servers. References to server processes are made using ASCII names and a name server performs the translation.
- Address caches can be used
- Requires a centralized component

- Name server can be replicated, but care must be taken to maintain consistency across servers

COMMUNICATION PRIMITIVES

- Locked vs. Unlocked Primitives
- Buffered vs. Unbuffered Primitives
- Trusted vs. Unreliable Primitives

Syncronous

LOCKED VS. UNLOCKED PRIMITIVES

- In blocked (synchronous) primitives, the process is blocked while a message is sent or received
- In unblocked (asynchronous) primitives, the process is unblocked before sending the message
- Process continues its execution in parallel with the message sending
- Process that sends the message cannot use the buffer until the message is sent. How do I know the message was sent?

- Copying the message to a buffer (appears to be a blocked primitive for the sending process)
- Extra copy is “wasted”, and can reduce system performance
- Another solution is to interrupt the sender when sending the message is completed, releasing the buffer for the sending process
- Efficient method that allows the greatest degree of parallelism between the options presented.

- Complicates programming and debugging programs
- Similar analysis with processor receivers
- Use of timeouts (timers) to prevent processes from being blocked forever in case of transmission failure

BUFFERED VS. UNBUFFERED PRIMITIVES

- receive(addr,&m) – process tells the core of your machine that it is expecting a message addressed to addr and that it should put it in m
- receive has to be executed before receiving the message (unbuffered), otherwise the kernel doesn't know who to deliver the message to
- The core may discard the message, which will likely be sent again and perhaps receive has been executed

- After several attempts client may think that server has crashed or address is invalid. This problem is aggravated in case the server has many requests
- Nucleus can temporarily store messages and start timers. Use of (buffered) mailboxes
- The problem repeats when mailboxes fill up
- Sender blocked until receiving ACK from server. If the mailbox is full, the sender is suspended by the scheduler

↳ acknowledgement

TRUSTED VS. UNTRUSTED PRIMITIVES

- Define whether or not the system guarantees message delivery
- Define send semantics as unreliable, implementing reliable communication is the task of users
- Core of receiving machine sends ACK back to core of sending machine. Transparent to processes
- Server response serves as an ACK. You can implement sending an ACK from the core of the sending machine to the core of the receiving machine

IMPLEMENTING THE CLIENT-SERVER MODEL

- Choice of types of communication primitives
- Message numbering to help correct transmission errors
- ACKs for individual packets (more messages, less retransmission) or entire messages (less messages, more complex retrieval)
- Choice will depend on network error rate
- Examples of packet types: REQ, REP, ACK, AYA, IAA, TA, AU

- AYA package serves to differentiate between a request that is taking a long time to be served and a server that crashed
- TA can be used to indicate the server mailbox is full but the address is correct while AU indicates the address is wrong

Code	Packet type	From	To	Description
REQ	Request	Client	Server	The client wants service
REP	Reply	Server	Client	Reply from the server to the client
ACK	Ack	Either	Other	The previous packet arrived
AYA	Are you alive?	Client	Server	Probe to see if the server has crashed
IAA	I am alive	Server	Client	The server has not crashed
TA	Try again	Server	Client	The server has no room
AU	Address unknown	Server	Client	No process is using this address

Client/Server Model

→ How machines can communicate
→ machine can run multiple client and/or server processes

Addressing

→ Each process has address ⇒ requires a centralized process address allocator
→ Process locate others through broadcast messages
→ Name servers (ASCII) → can be replicated
→ Requires a centralized component

Communication Primitives

1. Blocked vs Unblocked Primitives
2. Buffered vs Unbuffered Primitives
3. Trusted vs Untrusted Primitives

1. Blocked vs Unblock Primitives

→ Blocked (synchronous) primitives ⇒ the process is block while sent or receive
→ Unblocked (asynchronous) primitives ⇒ process is unblocked before sending
→ Process that sends the message can't use the buffer until message is sent
  ↳ copying message to buffer (blocked - for sending)
  ↳ interrup the sender when sending is completed, releasing the buffer for sending

To know the message was sent

2. Buffered vs Unbuffered Primitives

→ Receive (addr, &m)
  address ↓ ↳ to put the address
→ Receive has to be executed before receiving it (unbuffered)
  ↳ otherwise kernel doesn't know who to deliver
→ The core may discard the message
  ↳ can send again
→ Client may think server has crashed
→ Nucleous can temporarily store message (mailboxes)
  ↳ mailboxes can be full
→ Sender blocked until receiving ACK from server
  ↳ If mailbox is full ⇒ sender is suspended

3. Trusted vs Untrusted Primitives

→ To guarantee message delivery
→ Core of receiving machine sends ACK to core of sending machine

II. Remote Invocation of Procedures

1. Remote Procedure Calls (RPC)
2. Communication between Processes
3. Remote Method Invocation Concept (RMI)

Model of RPCs and

Remote Object Invocation

REMOTE PROCEDURE CALLS (RPC)

- It is a protocol for remote procedure calls, composed of layers supported on a distributed computing environment (DCEDistributed Computing Environment)
- It follows a set of rules that specify ways to encode and decode information transmitted between two applications.
- Helps programmers design and understand distributed programs more easily because it links client-server communication to conventional procedure calls.

- The RPC paradigm is application oriented, not communication protocol oriented
- Allows the programmer to design a conventional program that solves the problem, and then break the program down into procedures that can be run on multiple computers.
- A message sent by a client to a server corresponds to a "call" from a remote procedure, and a response from the server to the client corresponds to a "return" from a procedure call.

- An operation has a signature (the name of the operation), its input parameters, results, and exceptions that can happen
- The signature of an operation is encapsulated in a structure called IDL (Interface Definition Language), responsible for specifying the characteristics of the procedure provided by the server to its client
- Calling a procedure that is located on a remote system requires specifying which system to contact, how to encode the parameters, how to receive the response, and how to decode the response for use on a specific system

- RPC enables communication between machines with different OSs and/or hardware configurations, as the transferred message is written in a standardized data structure
- Machine with less processing capacity may request services for a faster one
- Example - file servers
- Allows any programmer with knowledge of structured programming to be able to develop distributed applications without major difficulties

- RPC enables communication between machines with different OSs and/or hardware configurations, as the transferred message is written in a standardized data structure
- Machine with less processing capacity may request services for a faster one
- Example - file servers
- Allows any programmer with knowledge of structured programming to be able to develop distributed applications without major difficulties

- The stub makes a remote client call that runs on the server machine look like a local call made to a procedure inside the same client machine.
- This function handles converting the data to an external representation of the data, sends a request message, receives the response, and converts the return from the external representation to the specified data type.

Stub steps when invoked:

- Initiate connection to the server unit containing the remote object.
- Write and transmit parameters to the remote server unit.
- Wait for method invocation results.
- Read the returned results.
- Return values to the calling object.

- Skeleton makes a service request from a client, which is received from another machine, look like a local request call.
- Server registers itself in a binder (program that knows the location of the server, and connects the client and server with the name service)
- To invoke a procedure, the client makes a request to the 'client stub', which transports the message to the server, which in turn delivers the message to the 'server stub'

Skeleton steps when receiving a call:

- Read the parameters sent by the "stub".
- Invoke the method on the remote object.
- Write and transmit the result to the object that performed the call.

DYNAMIC BINDING

- How does the client locate the server?
- The formal specification of the server (types of parameters passed, service names, server version) is used by the stub generator
- When starting its execution, the server exports its interface sending a message to the binder (server log) containing its name, version, an ID and a handle
- Client sends a message to the binder requesting a version service, and receives a handle and ID if such server currently exists
- Flexible, but generates extra messages and can become a bottleneck

RPC FAILURES

Types of failure:

1. Client cannot locate server
2. Request message from client to server is lost
3. Reply message from server to client is lost
4. Server fails after receiving request
5. Customer fails after submitting request

Type 1 - Client cannot find server

- ▫ Client may have wrong server address. Global error variable or exceptions (not all languages have exceptions and it does away with system transparency)

Type 2 - Request message from client to server is lost

- ▫ Use of timers

Type 3 – Reply message from server to client is lost

- ▫ Timers could cause the request to be sent again, but some operations cannot be performed more than once (funds transfer)
- ▫ Sequenced requests and retransmission bits

Type 4 - Server fails after receiving request

- Server may fail before or after processing request.
- How to differentiate the two cases?
- Options are to ensure:
  - at least once
  - At most once
  - nothing
- The ideal of running exactly once is not possible in some cases

Type 5 - Client fails after submitting request

- Client fails after sending request, creating orphans, which waste CPU cycles, can block resources, etc.
- Elimination of orphans through Extermination, Reincarnation, Mild Reincarnation or Expiration
- Problems if an orphan is eliminated while maintaining some resource lock or has made unfulfilled requests to start other processes

IMPLEMENTATION ASPECTS

- **RPC protocols**: connection-oriented vs. not connection-oriented (simpler communication vs. more efficient communication)
- Protocol specific to RPC (may be more efficient) or general, such as IP (implementations already exist on several systems)
- **ACKs** – Stop-and-wait or burst protocols send one ACK per packet or one per group of packets, respectively
- Selective repetition can be implemented with burst confirmation protocol

Timers

- Rarely the maximum value of a timer is reached, but it still has to be added and removed from some list.
- Maximum timer values affect the performance of algorithms and not their correctness
- Can be stored in a linked list or in the process table

AN EXAMPLE OF RPC

Sun-RPC: System originally created for Sun machines but currently used by other OSs

The defined architecture includes:

- A language for defining interfaces (procedure headers, etc.)
- The RPCGEN tool, which automatically generates server and client stubs
- An RPC library, which can be used directly when building programs that do not use RPCGEN
- The communication protocol between the stubs

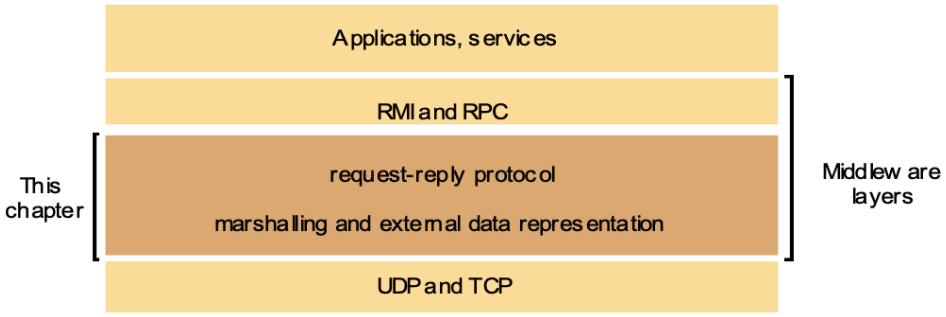
- You can use TCP or UDP
- Translation between data formats is done through the use of a standard representation, the XDR (eXternal Data Representation Standard)
- Conversion is specified for a predefined set of data types

Communication
between
Processes

GOALS

- Know the primitives used for communication between processes
- Overview of inter-process communication paradigms

LAYERS OF A MIDDLEWARE



REPRESENTATION AND DATA ORGANIZATION

- All information stored in a program has the following characteristics:
- Are stored in some kind of structure
- Are formed by a sequence of bytes
- Are transformed into bytes before transmission
- Are reassembled to the original format after receipt
- Have different types

Characters can be:

- ASCII
- Unicode

Data representations can be:

- Big-endian
- Little-endian

There are two methods to allow two different computers to exchange data:

- Values are converted before transmission into a common external format, agreed in advance between the two machines
- Send the data in the format used by the transmitter and together also send necessary information about the format used

To support RPC you need:

- That arguments are transparently passed from one machine to another
- Before being transmitted, the data must be transformed into primitive data formats according to an external format
- During reception, the primitive data is reassembled to a format accepted by the machine

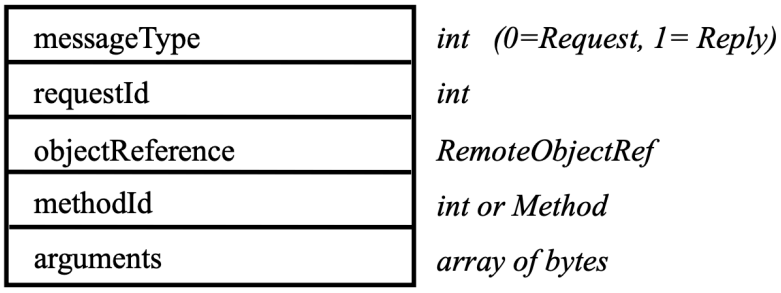
- This common format standard is called External Data Representation
- The process of assembling data to an externally accepted format is called Marshalling
- UnMarshalling is the inverse process, where the external format is disassembled to a format known locally by the machine
- Due to the complexity involved in the Marshalling and UnMarshalling process, this task is commonly performed by Middleware

COMMUNICATION BETWEEN PROCESSES

- How does communication between processes take place?
- Message exchange
- What is a message?
- It is an object of communication prepared in a possible form of transmission

MESSAGES

Structure of a message in the Client Server model



- MessageType – indicates whether it is a request or a response
- RequestID – Message identifier
- ObjectReference - Object identifier
- MethodId – Method identifier
- Arguments - Arguments to be passed/returned

COMMUNICATION BETWEEN PROCESSES

Characteristics of interprocess communication:

- Message passing between two pairs of processes can be implemented by two basic operations: send and receive.
- Communication activity always involves a synchronization mechanism.
- Processes must somehow have synchronization to send and receive data.

Synchronous and Asynchronous Communication:

- A queue is associated with each message. The process that sends the message does nothing more than insert the message into a remote queue. The receiving process removes the message from the local queue.
- In the synchronous form of communication, the transmitter and receiver synchronize the passage of each message.

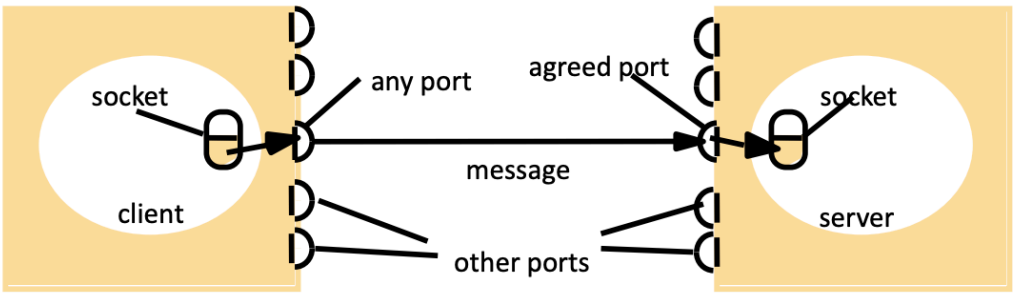
Synchronous and Asynchronous Communication:

- In the asynchronous form of communication, the sending operation is non-blocking. In this way, data transmission can take place in parallel with the transmitting process.
- Operations using blocking mode have simpler implementation, but non-blocking mode offers better performance in operation.

Sockets

- Both forms of communications (UDP and TCP) use socket abstraction to provide the means of end-to-end communication between processes.
- The term socket originally originated from BSD UNIX, but is also used in many versions of UNIX, Linux, Windows, and Mac OS.
- An inter-process communication is based on the use of a socket in the transmitting process and another socket in the receiving process.

- The most basic primitives of interprocess communication are send and receive.
- If they are blocking, the communication is synchronous
- Communication using Socket



COMMUNICATION LAYERS

Network layer:

- Responsible for best-effort package communication between us
- Packages may follow different routes or not arrive

Transport layer:

- Communication between processes
- Multiplexing of communication across ports

TRANSPORT

Typically, a programmer only handles the transport layer

- TCP
- UDP

Lower layers are responsible for sending raw data without guarantees of:

- Receipt
- Order
- Integrity

The transport layer makes it tractable for the developer

UDP

Communication by UDP datagram

- A packet is transmitted without confirmation or message resending
- Errors are not returned or handled

Information from a datagram:

- Message size
  - Total = 65536 bytes including header
- Block
  - waiting for message arrival
- Timeout
  - maximum waiting time for a reception

Failure Model for UDP:

- Failure by default
  - message may be lost occasionally
- Ordering
  - information may be delivered out of order

Applications:

- DNS
- Multimedia

UDP is the transport-level network protocol

- Transmission of datagrams converted into packets
- Optional Checksum
- No guarantee of arrival or ordering
- There is no retransmission

UDP PRIMITIVES

send() can be non-blocking

receive() blocking a datagram

- Receive from any source
- May have a wait timeout

Messages are sent when given to the IP and are put in a receive buffer at the destination until the *receive()* call

UDP MODEL

Messages may be lost

- Communication is subject to failures of omission

Messages may go out of order

- Following different paths in routing

Several approaches can use UDP as a solution

- DNS
- NTP
- Video streaming
- Audio broadcast
- Between others...

TCP

Provides an abstraction for a *stream* of bytes, which can be used to write or read data

Consideration should be given when using TCP:

- Message size
- Loss of message
- Flux control
- Sorting and duplicating packages
- Connection-oriented message

Usage characteristics:

- Data compatibility
- Communication locks
- Cannot distinguish between a network or process failure

Use:

- HTTP
- FTP
- Telnet
- SMTP

Characteristics

- Reliability
- Data streams (in connections)
- Automatic segmentation of flows into packages
- Guarantees with retransmissions, ordering and *checksums*
- Flux control
- Buffering

TCP PRIMITIVES

accept() and *connect()* establish a connection

send() and *receive()* of bytes, not datagrams

Abstract:

- Message sizes
- Message losses
- Differences in sending and receiving speed (flow)
- Duplication and ordering of messages

TCP MODEL

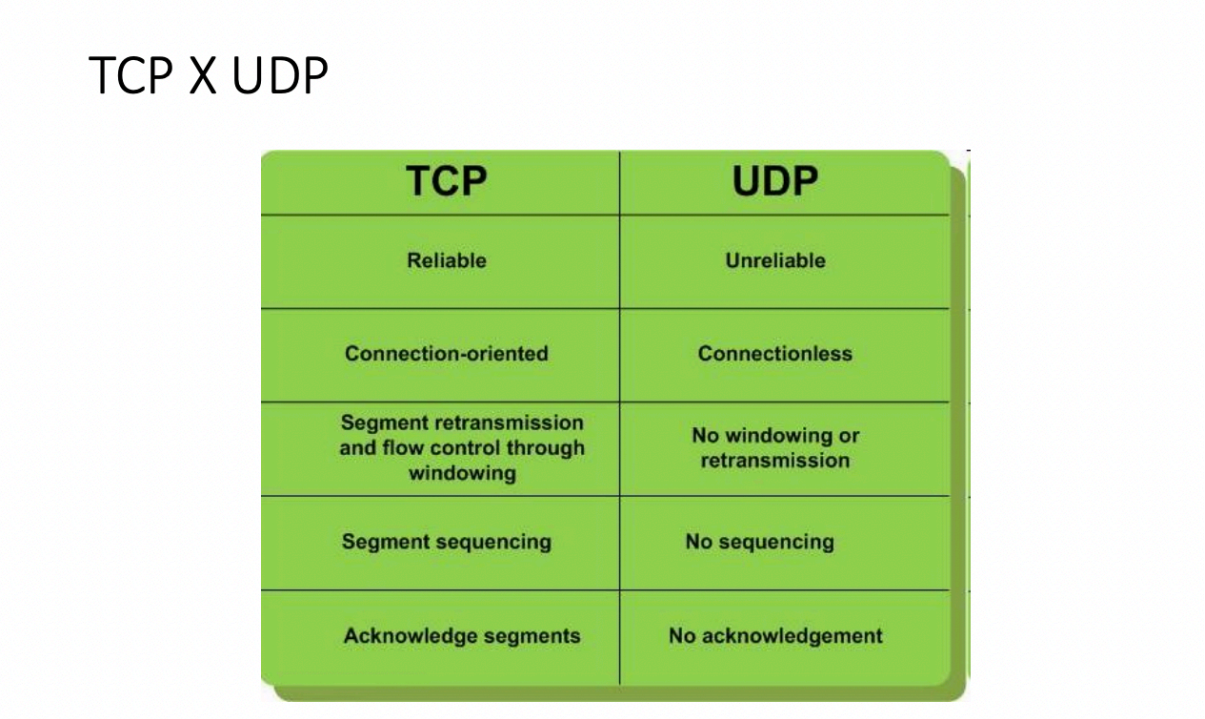
There are no omission failures or cluttering messages

Timeouts hide from the user that the world is asynchronous

Processes do not immediately detect which message was not received

- Wait for ACKs

TCP X UDP



TCP	UDP
Reliable	Unreliable
Connection-oriented	Connectionless
Segment retransmission and flow control through windowing	No windowing or retransmission
Segment sequencing	No sequencing
Acknowledge segments	No acknowledgement

In short, reliability vs. performance

- TCP properties are not always important

TCP AND UDP

Who use?

- UDP:
  - NTP, DHCP
  - video and audio
- TCP:
  - HTTP, FTP, SMTP

What strategies to adopt in a system that requires the security of TCP and the agility of UDP?

ABSTRACTIONS

send() and r*eceive()* are the basic primitives

It is commonly more convenient to use higher-level abstractions such as:

- Communication 1 – N → Multicast
- Loosely coupled communication → Queuing
- Efficient group communication → MPI

MULTICAST

Communication 1 → several

Exists at the network level (IP Multicasting)

- It is usually done over UDP
- Everything that is sent to the group is received by everyone

Broadcast is a particular case of Multicast

Reliable and orderly Broad/Multicast is a powerful primitive

QUEUING

Used in communication where origin and destination do not need to be online at the same time

Systems based on publish/subscribe use this approach

The principle is that clients allocate messages to queues

- Queues are persistent, implemented by servers
- Destinations read messages from their queues

MESSAGE PASSING INTERFACE

MPI - Message Passing Interface

- It is a standard for data communication in parallel computing.

In the MPI standard, an application is made up of one or more processes that communicate, triggering functions to send and receive messages between processes.

Objective

The goal of MPI is to provide a broad standard for writing message passing programs in a practical, portable, efficient and flexible way.

MPI is not an IEEE or an ISO standard, but rather an industry standard for message exchange program development.

Before MPI, companies provided their own libraries

There are several implementations of the pattern

- Open MPI
- LAM/MPI
- Los Alamos MPI (LA-MPI)
- FT-MPI

Meets the needs of a parallel communication system

- Transparent peer-to-peer communication
- Group communication
- Abstractions for writing portable applications
- Between others...

MPI primitives

The basics for each process:

- MPI_INIT
- MPI_FINALIZE
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_SEND
- MPI_RECV
- MPI_BCAST
- MPI_REDUCE

COMMUNICATION BETWEEN PROCESSES

Example of Client-Server Communication

- HTTP
- The HTTP (HyperText Transfer Protocol) protocol is used by the client web browser to make requests to Web Servers and receive response information

HTTP

Servers manage two distinct types of resources:

- Data
- Software

Data are, for example, pages stored on servers that are sent to clients after a request.

Programs are CGIs or scripts that can be run for specific tasks.

HTTP is a protocol that basically specifies rules for exchanging messages between the client and the web server.

It presents specific methods to control server resources, such as: GET, PUT, HEAD, POST, DELETE, etc.

Also define a method for user access authentication.

Content negotiation

- The client can include information in the request to inform the server what type of data it is supporting. This allows the server to better tailor the data to be presented on the client side.

Authentication

- Challenges for access through logins and passwords are possible for the server to verify the client's right to data access.

The original HTTP protocol specifies that a request always then closes the communication after the server sends the response.

In HTTP 1.1 version it is possible to define a persistent connection. This approach makes it possible to use more efficiently the transmission of several documents to the same client and can be closed at any time by both parties (server or client).

The protocol allows any type of data to be transmitted using HTTP.

The MIME – Multipurpose Internet Mail Extensions are defined to define the types and subtypes of data and inform browsers of the type definition they are receiving.

Examples: *text/plain, text/html, image/gif*, etc.

Methods implemented in HTTP:

- GET
- HEAD
- POST
- PUT
- DELETE
- OPTIONS
- TRACE

Method Invocation Example

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	https://www.estgp.pt/pt/	HTTP/ 1.1		

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

Remote Method Invocation

Concepts

RMI

SUMMARY

- Introduction
- RMI Applications
- Server Side
- Client Side
- Compilation and Execution
- Conclusions

INTRODUCTION

Distributed Objects Paradigm

- be able to interact with objects located on remote machines as if they were local
- be able to build an object on one machine and transmit it to another

Distributed Object Support Systems (SSOD)

- Java RMI
- CORBA
- DCOM

RMI - Remote Method Invocation

- Java language
- Since JDK version 1.1
- API specified through the java.rmi package and its subpackages

Set of collaborating objects that communicate over a network

- Objects on different virtual machines can interact

Integer objects can be passed or returned as parameters.

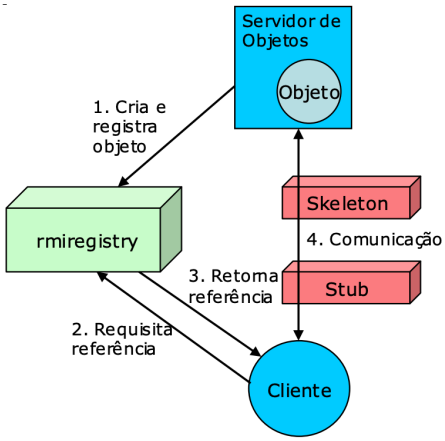
- New code (object) is dynamically loaded
- Distributed systems developers have greater flexibility

RMI APPLICATIONS

Composed of:

- a server
  - create remote objects
  - Associate them with a name
- a customer >client
  - references one or more objects on the server
  - invokes the methods of these objects.
- remote object references are obtained through the RMI registry (rmiregistry)

RMI architecture



Remote Interfaces, Objects and Methods

- Distributed application = classes + interfaces
- Interfaces define methods and classes implement those methods
- Objects whose methods can be called through remote virtual machines are **remote objects**
- An object becomes remote through the implementation of a **remote interface**
  - Derived from the java.rmi.Remote class
- Remote objects must implement one or more remote interfaces.

INTERFACE AND IMPLEMENTATION

What is an interface?

How to implement the interface?

- java.rmi.UnicastRemoteObject
  - Exceptions
  - Resources on the server
  - Synchronization

INTERFACE EXAMPLE

```
import java.rmi.*;
public interface Produto extends Remote{
   /* Informa a descrição do produto */
   String leDescricao() throws RemoteException;

   /* Informa o valor do produto */
   float leValor() throws RemoteException;

   /* Altera a descrição do produto */
   void setaDescricao(String s) throws RemoteException;

   /* Altera o valor do produto */
   void setaValor(float v) throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.server.*;

public class ProdutoImplementacao extends
                UnicastRemoteObject implements Produto{
  private String nome;
  private float valor;

  public ProdutoImplementacao(String nome, float valor)
                                        throws RemoteException{
    this.nome = nome;
    this.valor = valor;
  }
  public String leDescricao() throws RemoteException{
    if(nome.equals(LeitorPropriedades.lePropriedade(
              ... //LENDO PROPRIEDADES
      else
        return nome + " e' um produto novo.";
  }
  public float leValor() throws RemoteException{
    return valor;
  }
  public void setaValor(float valor) throws RemoteException{
    this.valor = valor;
  }
  public void setaDescricao(String nome)
                                    throws RemoteException{
    this.nome = nome;
  }
}
```

STUBS AND SKELETONS

What is a stub?

- Localization
- Occupation
- Marshalling

What is a skeleton?

- Localization
- Occupation
- Unmarshalling

WHAT IS A STUB?

A stub or method stub, in Portuguese method outline, in software development, is a **piece of code** used to replace some other programming functionality. A stub can either simulate the behavior of existing code (such as a procedure on a remote machine) or be a temporary replacement for code yet to be developed. They are therefore most useful in **portability, distributed computing** as well as general software development and testing.

They are parts (fragments) of algorithms that provide the abstraction of a (local) procedure (method) call, linking it with the communication mechanism. **Stubs are like a proxy for remote objects**, they are pieces of code that make the remote call, they are used on the client and on the server.

```
BEGIN
    Temperature = ThermometerRead(Outside)
    IF Temperature > 40 THEN
         PRINT "It's HOT!"
    END IF
END
```

```
BEGIN ThermometerRead(Source insideOrOutside)
     RETURN 28
END ThermometerRead
```

WHAT IS A SKELETON?

The "skeleton" is responsible for sending the call to the remote object.

Steps of the "skeleton" when receiving a call:

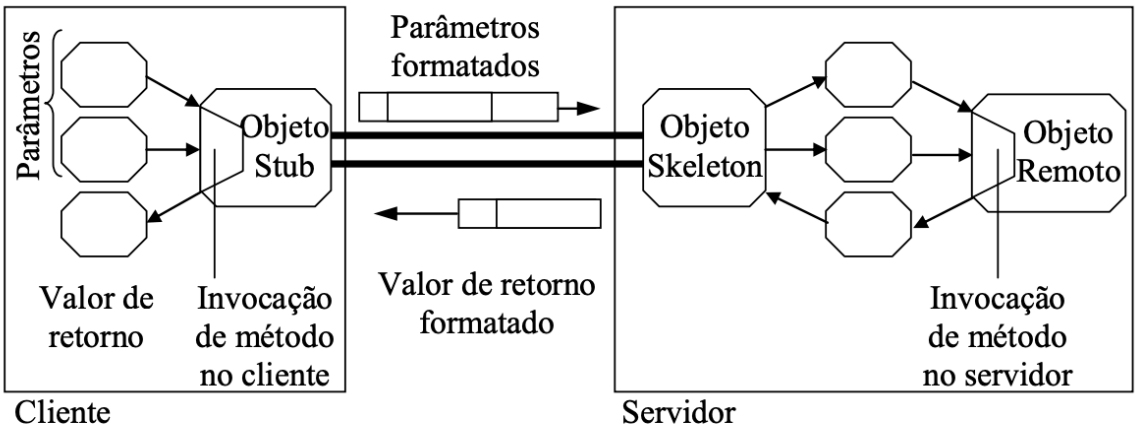
- Read the parameters sent by the "stub"
- Invoke the method on the remote object
- Write and transmit the result to the object that performed the call

MARSHALLING AND UNMARSHALLING

In computer science, marshalling is the process of transforming an object's memory representation into a compatible data format for storage or transmission, and is typically used when data they need to be moved between different parts of a computer application or from one application to another.

Marshalling is similar to serialization and is used for communicating remote entities using an object, in this case a serialized object. Simplifies complex communication steps, using custom objects instead of primitives. The inverse of marshalling is called **unmarshalling** (or demarshalling, similar to deserialization).

STUBS AND SKELETONS



REGISTERING OBJECTS

What must be provided to register?

Safety

```
/* Cria instância do Produto */
ProdutoImplementacao p1 =
        new ProdutoImplementacao("radio",20.3f);

/* Registra os produtos acima no serviço de registros */
java.rmi.Naming.rebind("MeuRadio", p1);
```

CLIENT SIDE

The client needs to manipulate objects, but it doesn't have copies of them (**the objects reside on the server**).

The client must also know what he can do with these objects (**interface**).

The functions of the client program are:

- Install an RMI security manager;
- Find a remote object;
- Invoke remote object methods

INSTALL AN RMI SECURITY MANAGER

What is an RMI security manager?

- Required when classes are offloaded from remote locations.
- It does not apply to Applets, which run under the protection of your browser security manager.

Java provides a security manager, the RMISecurityManager, which is installed with the statement:

```
System.SetSecurityManager( new RMISecurityManager() );
```

FIND A REMOTE OBJECT

To obtain a remote reference, the **lookup** method of the **Naming** class that interacts with rmiregistry is used.

```
String rmiUrl = “rmi://200.17.94.117:3333/”;
Produto p1 = (Produto)Naming.lookup(rmiUrl + "MeuRadio");
```

- The string **rmiUrl** must have the format **rmi://host:port/**
- the **name of the desired remote object** must be concatenated with the string rmiUrl to perform the remote object search.
- If any part of the URL is missing, then the Naming service automatically completes (rmi://localhost:1099/MyRadio).

INVOKING REMOTE OBJECT METHODS

The lookup method returns the **remote reference** to an object that implements the remote Product interface and this is assigned to p1.

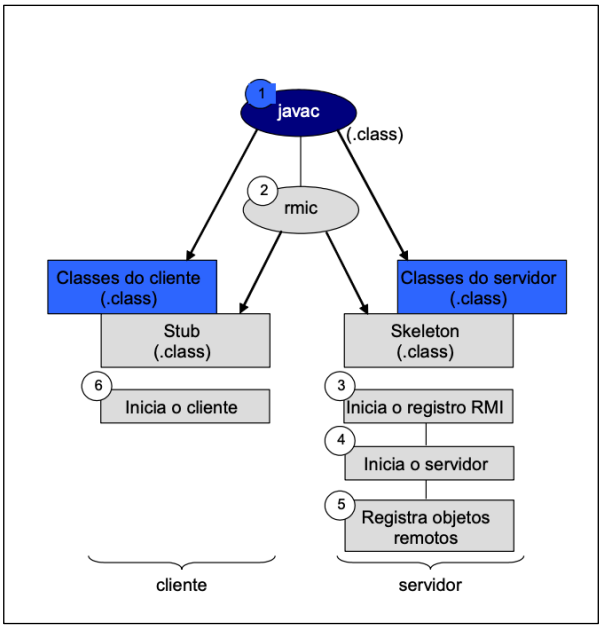
The client uses p1 to invoke remote object methods as if it were a method of an object defined **locally** on the same Java virtual machine as the client application.

```
System.out.println(p1.leDescricao());
```

COMPILATION AND EXECUTION

Step 1. Compile the classes

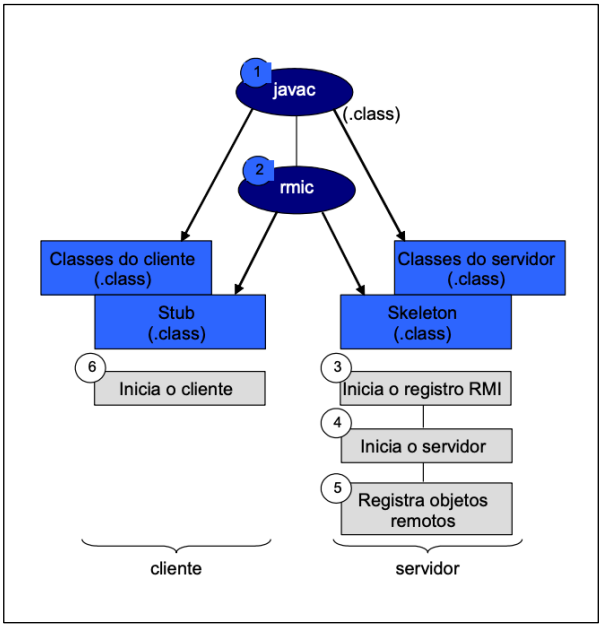
```
javac *.java
```



Step 2. Compile the class that implements the remote object with the **rmic** compiler.

```
rmic ProdutoImplementacao
```

- Produces the Stub and Skeleton classes.
- The Stub class must be available to the customer



Step 3. Start RMI logging.

Porta default 1099

No Windows : `rmiregistry`

No Unix: `rmiregistry &`

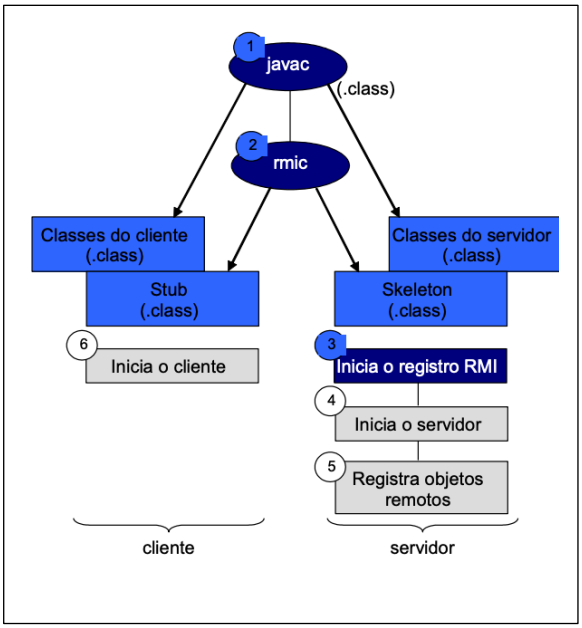
Outra Porta

No Windows : `rmiregistry 3333`

No Unix: `rmiregistry 3333 &`

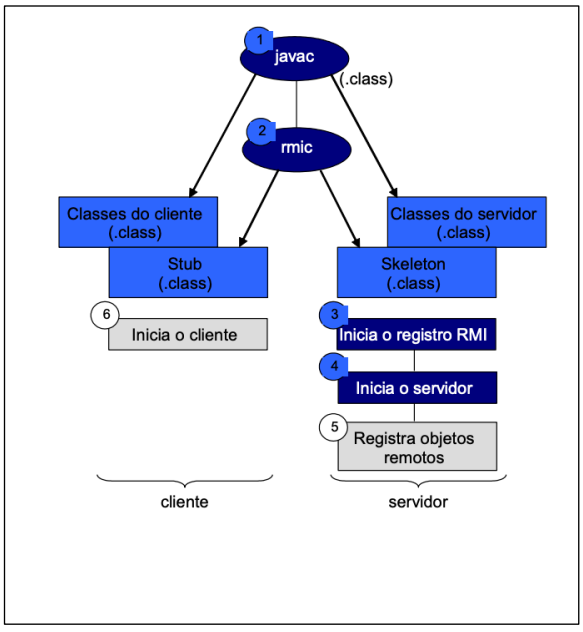
corresponds to a record of the type:

```
Naming.rebind("rmi://200.17.94.117:3333/MeuRadio",obj)
```



Step 4. Start the server.

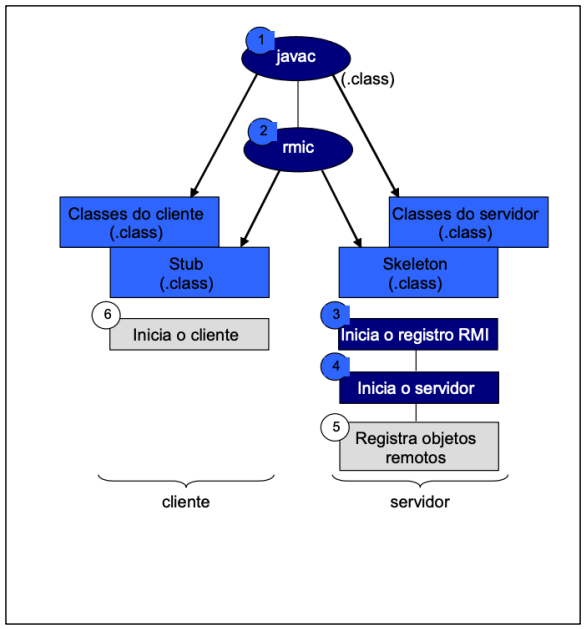
```
java ProdutoServidor
```



Step 5. Register the remote object.

Occurs during server code execution by calling the rebind method of the Naming class.

```
Naming.rebind("rmi://200.17.94.117:3333/MeuRadio",obj)
```

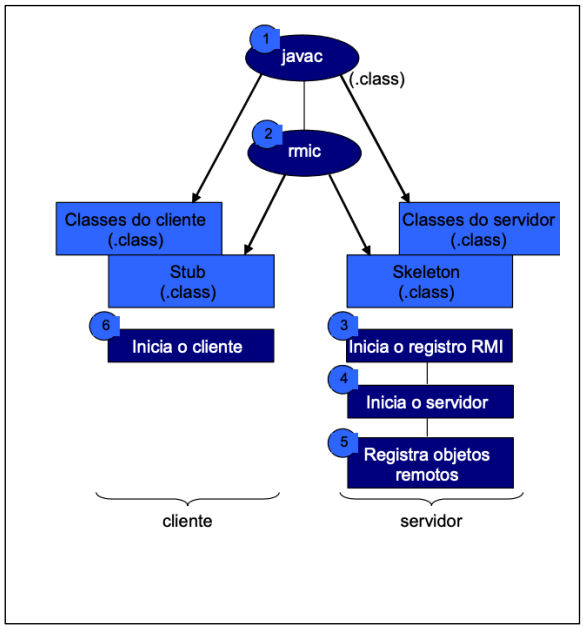


Step 6. Start the client.

```
java ProdutoCliente
```

java.rmi.ConnectException

This exception indicates that the server application has not started or that the remote object has not been linked to *rmiregistry*.



SAFETY

You can specify a policy file via the –Djava.security.policy command line argument

```
java -Djava.security.policy=java.policy ProdutoCliente
```

where java.policy is a policy file

```
grant {
    permission java.net.SocketPermission "*:1024-65535","connect,accept";
    permission java.net.SocketPermission "*:80", "connect";
};
```

This java.policy file allows you to do two actions:

- Connect to or accept connections on ports greater than 1024 on any host;
- Connect port 80 (HTTP port).

NOTE: when the user who runs the server application is not the machine's administrator, it is necessary to use the policy file.

CONCLUSIONS

Flexibility

- Application behavior is extended

Performance

- Remote calls use the network
- Cost of (un)packaging parameters

Solutions

- Structure the application
- Move data with the minimum number of calls
- Caching data

Distributed Systems

Names

GOALS

- Understand the use of names in distributed systems
- Understand the resource discovery process
- Analyze the solution space for a distributed name system

ADRESSES

Why don't we always use the address as a name?

- Device, code, service mobility
- Multiple replicated servers

ATTRIBUTES

Names are not always enough

Attributes reference entities by characteristics

- Each entity has sets (attribute, value)

Sometimes there is a distinction between services that use names and attributes

BASIC PRINCIPLES

Object-oriented middleware platforms use object *references* to address server objects

A way to obtain such object references without the need for assumptions about physical location is needed.

A name is a sequence of strings that can be "*linked*" to an object reference.

A name can be resolved to get the corresponding object reference

There can be many server objects in a distributed object system

Server objects can have multiple names

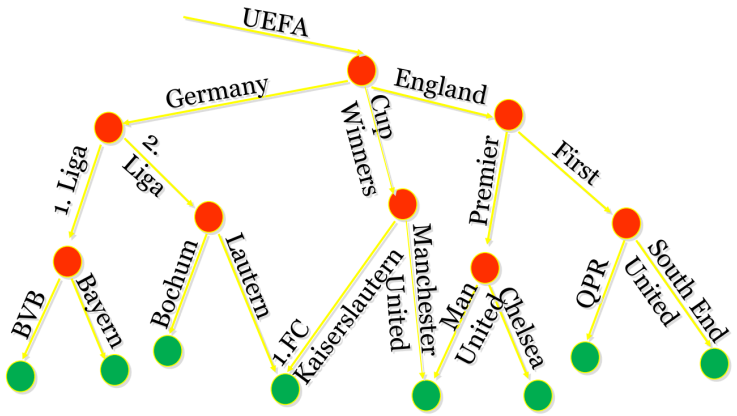
This leads to a large number of "name bindings"

The namespace must be organized in a hierarchy to avoid

- name conflicts
- poor name resolution performance

This hierarchy can be obtained through **name contexts**

NAME CONTEXTS



COMPOUND NAMES

Names are composed of possibly more than one component

The sequence of components of a name describes a path through the name context graph

Example:

- (“UEFA”, “England”, “Premier”, “Chelsea”)

NAME SERVER

Name "*bindings*" (to object references) are managed by name servers

Not every server object needs a name

Some server objects can have multiple names

Name servers must store name strings persistently

Name servers must be “*calibrated*” for efficiency in name resolution

The name server itself can be distributed

THE CORBA NAME SERVICE

Supports binding of names to CORBA object references

Scope of a name: context of names

Multiple names can be defined for the same object reference

Not all object references need names

NAMES IN CORBA

Names are made up of *simple names*

Simple names: *type-value pairs*

The *value* attribute is used for name resolution

The *type* attribute is used to provide information about the object's role

NAMES IN RMI: RMI REGISTRY

Simplified version of the CORBA Name Service

No support for composite names

Security restriction: name bindings cannot be created from remote machines

There must be a registry on each machine

Different *registries* must be integrated within a *federated* namespace

NAME SERVICE LIMITATIONS

In all of the approaches mentioned:

- Clients must always identify the specific server through a name

Inappropriate if the customer just wants to use a service with certain characteristics and quality, but doesn't know on which server:

- E-commerce
- Video on demand
- Automatic cinema ticket sales

TRADING

Locating objects in a transparent way of physical location

Using a naming service is simple, but may not be appropriate when:

- customers do not know the servers
- there are multiple servers for the same service

Trading supports server discovery based on functionality and quality of service

- Name Service ↔ White Pages
- Trading ↔ Yellow Pages

CHARACTERISTICS OF A TRADING SERVICE

Trader operates as an intermediary between clients and servers

Analogous to the idea of a *realtor*

Common language between client and server

- Types of service
- Qualities of service

Server registers with the trader

Server defines a guaranteed quality of service (QoS)

- Static or dynamic definition of QoS

Customers consult the trader to obtain

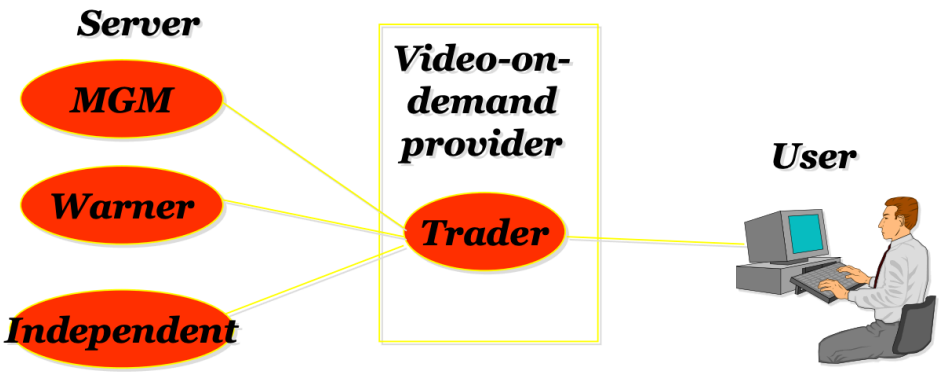
- It's a service of a certain kind
- a certain level of quality

The trader supports

- comparison of service types with specified properties
- search for services

Example

• Video-on-demand service:



DEFINITION OF SERVICE TYPES

A service type defines:

- The functionality provided by a service
- The quality of the service provided

Functionality is defined through an object type

QoS is defined based on properties

- property name
- property type
- property value
- property mode
  - mandatory / optional
  - read only / modifiable

SERVICE TYPE HIERARCHY

An object type can have multiple implementations with different QoS

The same object type can be used in different service types

Sub-typing relationship can be explored by the trader when searching for services of a certain type

TRADING POLICIES

Depending on restrictions and available services, a large set of offers may be returned by a query.

Trading policies are used to restrict the size of the list of service offerings returned

- Specifying a maximum limit
- Restriction on Service Replacement
- Restriction on modifiable properties

TRADERS FEDERATION

A trader participating in a federation:

- offers other traders the services he is aware of
- forwards queries to other traders for services they are not aware of

Problems

- service imports may not end
- duplication of offers

KEY POINTS

Distributed objects can be located through a naming service or *trading*

A name service links (externally known) names to object references

Trading supports finding objects based on the functionality they offer, as well as the quality they guarantee

EXAMPLE: DNS

DNS - Domain Name System

Mechanism that converts network addresses into ASCII string.

Motivation: At the beginning of ARPANET, there was only one text file, host.txt to list all hosts and their IP addresses.

DNS - MOTIVATION

People consulted the txt file in order to access the desired machines.

With the growth and transformation of ARPANET in the Internet network, the need arose to create a more efficient and intelligent mechanism. It had to be distributed in order not to be dependent on a single IP address supply center.

DNS is a hierarchical, domain-based naming scheme.

It is used to assign names to hosts (IPs) and destination name.

DNS - FUNCTIONING

Working example

- An application enters a name, automatically a procedure is triggered to search the corresponding IP address of that name. An API called a resolver then sends a UDP packet to a nearby DNS server to query what the desired IP is. With IP ownership, an application can communicate directly with the other machine if it so desires.

DNS - FEATURES

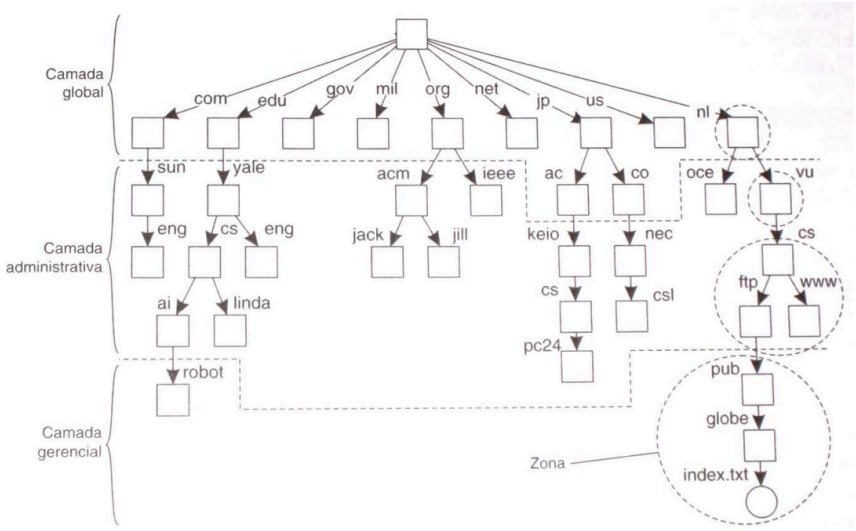
Characteristics

- The Internet is divided into domains
- Each domain can be divided into sub domains and so on.
- Can be represented by a large hierarchical tree.
- Leaves can be hosts that can represent machines or a host of a large company having more child hosts. Ex: intranet.univali.br or webmail.univali.br.

Top level domains

- Generic domains (com,edu, gov, int, net, mil, etc)
- Country domains (br,ch, jp, al, etc).

DNS NAMESPACE



Each domain needs the permission of the superior domain to be created. On the other hand, it doesn't need anyone's permission to create a subdomain under his.

DNS - RESOURCE RECORD

All DNS servers contain a configuration file called the Resource Record. When someone looks up a domain name, they're actually looking at those records.

An Resource Record is made up of five fields. They are usually encoded in ASCII text, occupying one line for each record.

Format:

- Domain_name - informs the domain that this record applies to.
- Time_to_live - Stability of information in seconds
- Type - Informs the record type
- Class - Indicates the class of information, for internet this field contains IN
- Value - Defines the content that depends on context (type)

DNS - TYPES OF RECORDS

Types of Records

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

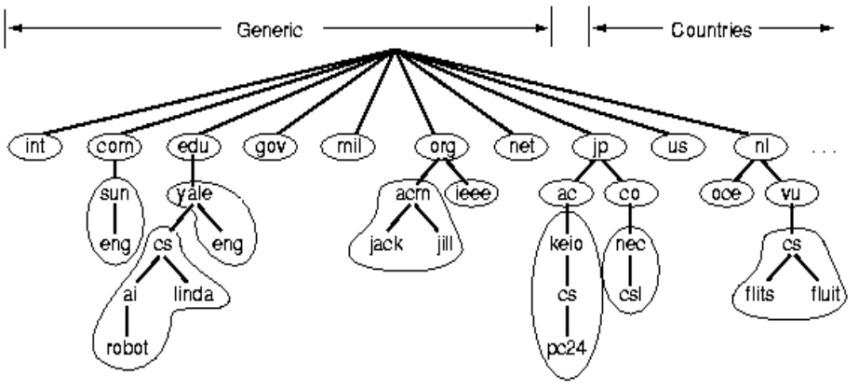
NAME SERVERS

In theory, a single nameserver with all the DNS BD could be used to do the mapping

The DNS namespace is zoned in such a way that there is no overlap.

DNS NAMESPACE

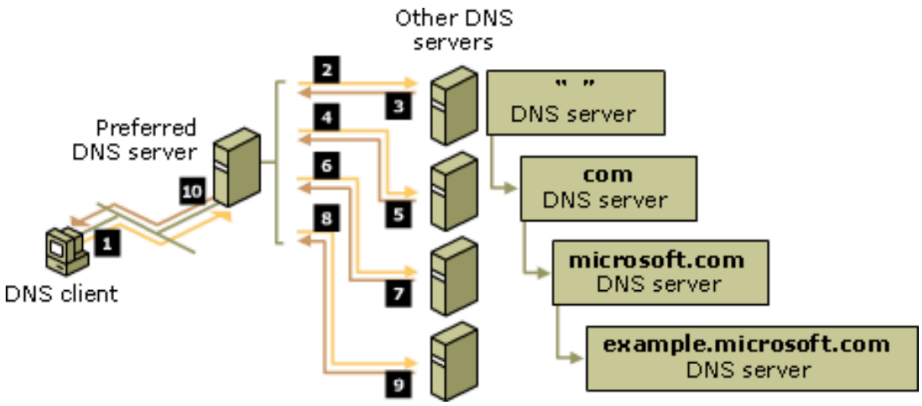
Name Servers (zones)



DNS - QUERY

Recursive Query:

- example.microsoft.com



DNS - OFFICIAL RECORDS

Official record - are returned by resource records that are managed by the servers that control that domain, so it is always correct.

Records obtained through caches are not as reliable as they may be out of date.

III. Distributed Applications for the Internet

1. Web Services Concepts

Web Services

Concepts

SCHEDULE

Introduction

Web Service

- Architecture

Solutions

- Basic steps
- Deploying with Apache Axis

New protocols

Conclusion

THE IT LANDSCAPE IN CORPORATIONS

New trends knocked on corporations' doors

- Migrated the focus from "**data management**" to "**management of processes and customers**"
- **Redesign of processes** and implementation of large business management systems (ERP)

Internet Success

- **Make** part of corporate **information** available to users or systems that cross corporate boundaries
- E-commerce

ENTERPRISE APPLICATION INTEGRATION

The **diversity** of systems coexisting in companies is **enormous**

- Large commercial packages to custom developed applications
- Different “software houses”
- Different technologies (host-centric, client-server, n-tier, etc),
- Different platforms (mainframes, Unix, Windows, etc).

SERVICE ORIENTED ARCHITECTURE (SOA)

SOA is an architecture that represents software functionality as services

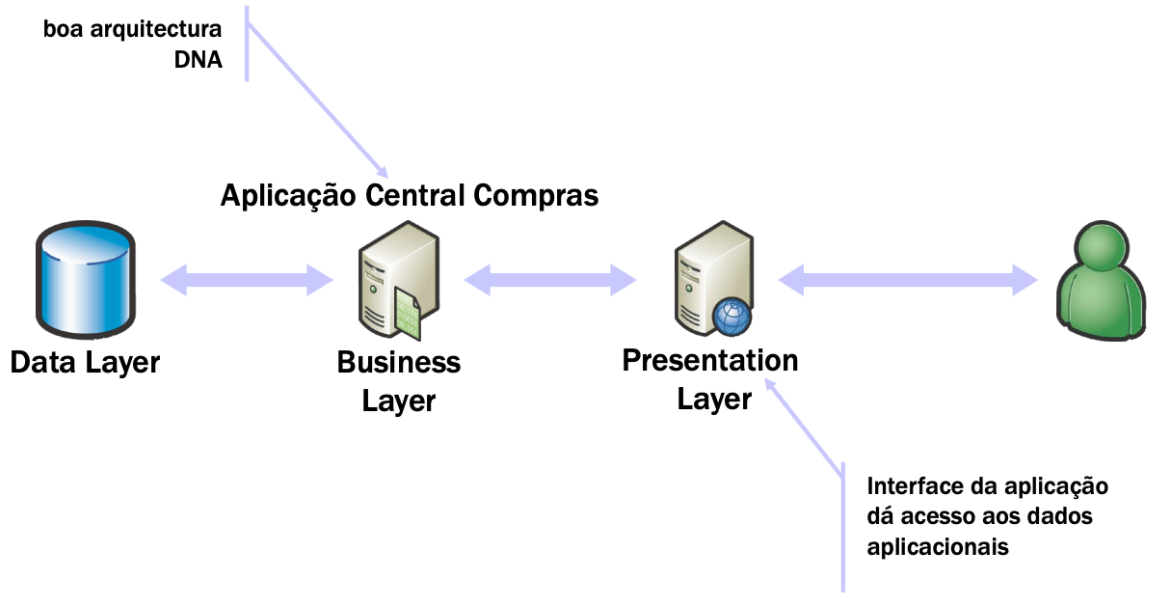
Technologies for SOA already existed

- Ex.: CORBA, RMI, etc...

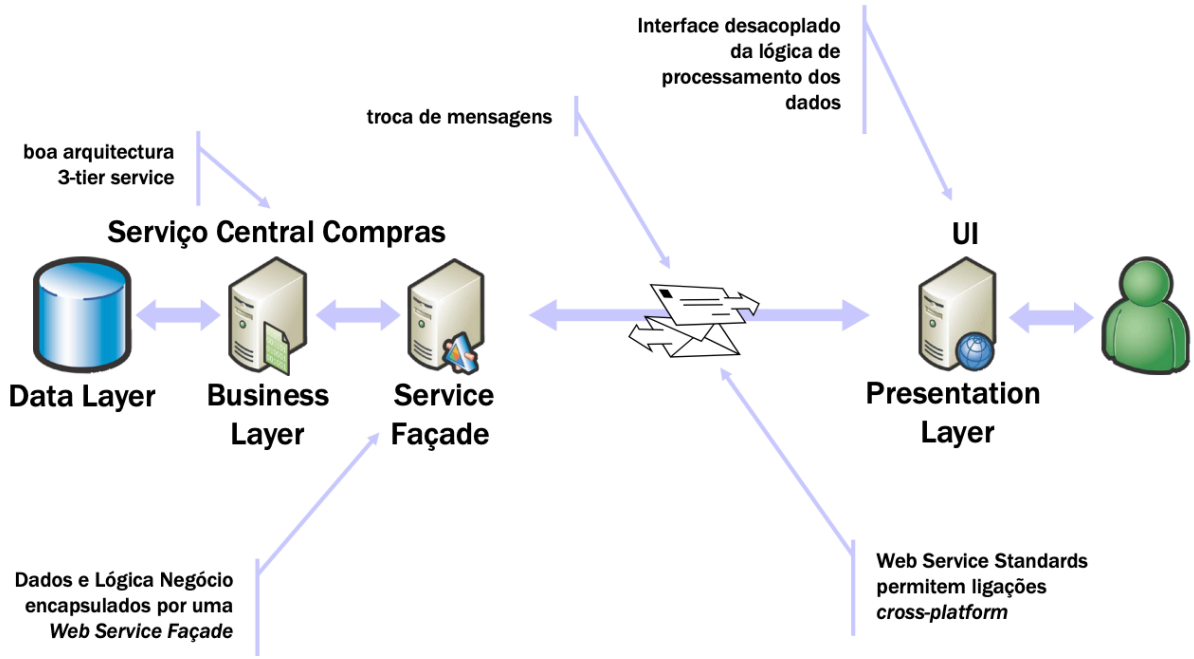
Interoperability is very important

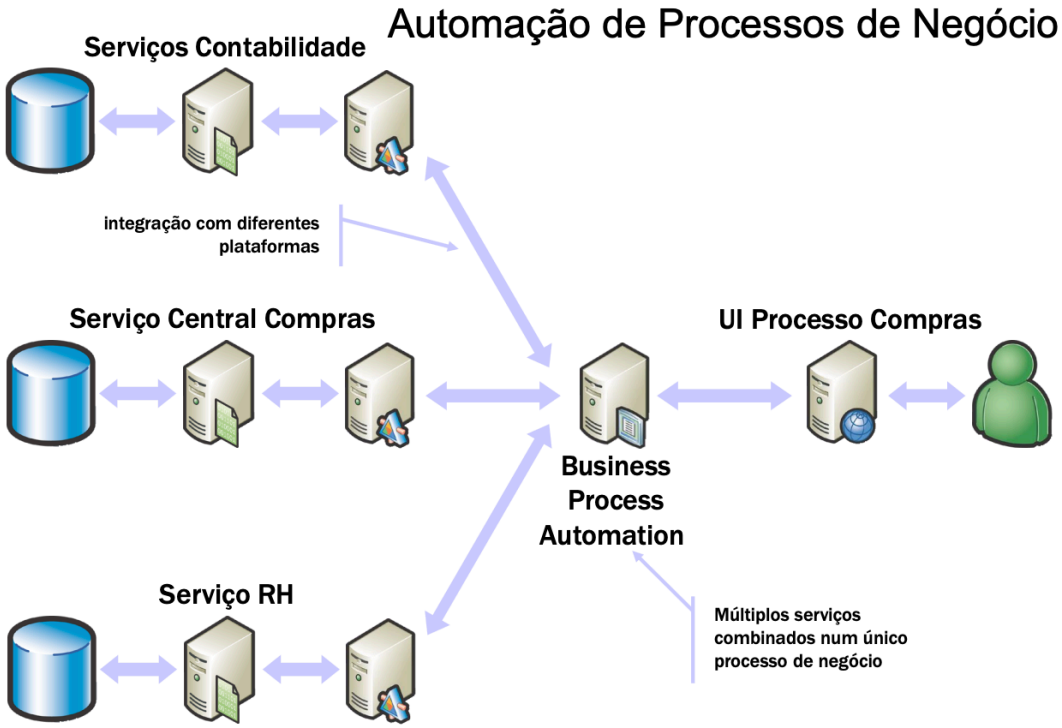
- Standardization
- Weak coupling

EXAMPLE: 3-LAYER APPLICATION



SERVICE ORIENTED ARCHITECTURE

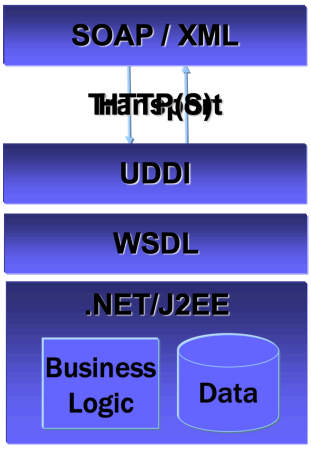




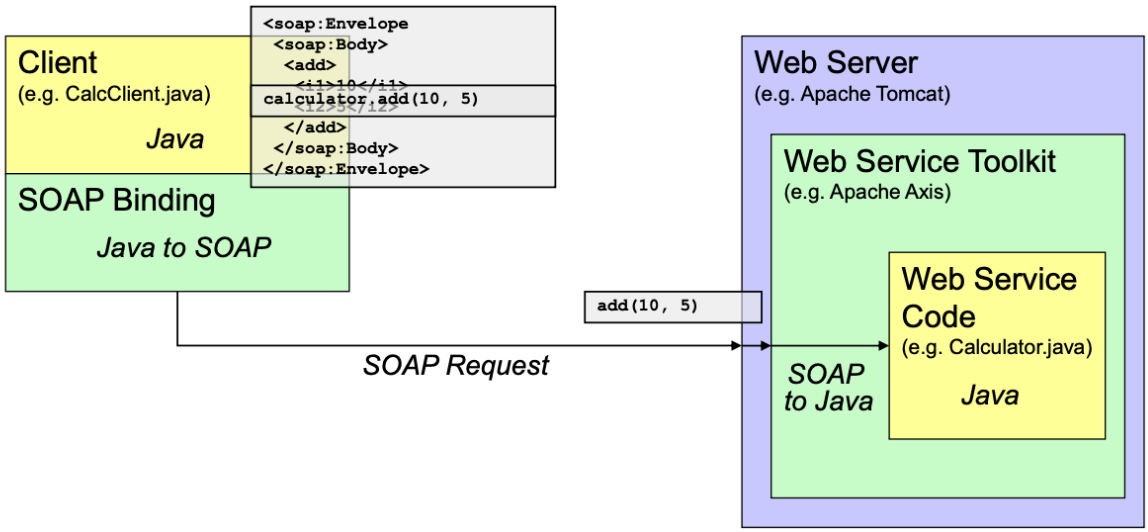
WEB SERVICES

- Web Services is a **remote object calling** technology
- Allows the **infrastructure for creating** distributed applications (web or otherwise)
- Allow the creation of small reusable code modules and available for building **"LEGO" applications**
- Uses **Web protocols as a means of transport** and communication
- High degree of **abstraction** regarding programming **languages** and hardware **platforms** /software

WEB SERVICE ARCHITECTURE



WEB SERVICES INVOCATION



BENEFITS

- Standard adopted by **W3C** and industry.
- Traverses **firewalls**, **routers** and proxy servers.
- Simple and **easy** approach to deployment and interoperability.
- S-I Consortium- **www.ws-i.org**
- Ensure interoperability
- **Open standards**
- **Platform independent**

- 170+ software manufacturers involved
- Interoperability between platforms, applications and languages
- Greater evidence that the industry is aligned with web services

SOLUTIONS

Decisive factors for the choice

- **Solution maturity**
- Integration with IDEs
- Security reuse
- Scalability (*serverside and stateless* implementation)
- **Documentation**

BEST KNOWN TECHNOLOGIES

NET Framework

Java

- Apache Axis
- JaxRPC
- IBM SOAP Toolkit

Any platform that has **HTTP** communication routines and knows how to handle **XML** data can implement Web Services.

Step 1: Service definition

SOA - Service Oriented Architecture

- A **service is a well-defined**, self-contained function that does not depend on the state or context of other services;

Step 2: Implementation

Service implementation:

- The service to be provided via the web service must be identified and "isolated" in a typically stateless API;
- Usually this **service is already implemented in the "legacy**", just defining and implementing an API for its access on the original platform.

Step 3: Value Objects

Input and output parameters:

- The API of a service should only contain "**compatible" types with web services** (basic types, strings, typed arrays or structures – VOs – of these types).

Step 4: Creating the WS

Public interest definitions:

- API definition (and value objects);
- Generation of the formal definition of the web service (WSDL);
- If VOs are compatible, the WSDL will contain a full description of the data structures (even complex ones);
- WS is, in practice, implemented by a common web application (HTTP/HTTPS), without the use of any proprietary extension.

Step 5: WS Publishing

Publishing the web service is physically like publishing a common web application

The public definition of the web service (WSDL) will be available for download by the web application that implements it

Web services API will be stable - changes and updates will be released in advance

Step 6: WS consumption

Customer generation:

- Access to the web service definition (WSDL) is all that is needed;
- IDEs have the ability to generate client stubs from WSDL;

APACHE AXIS

The SOAP Processing Engine

- JAX-RPC Client System
- JAX-RPC Server System (Servlet based)
- SAAJ
- Extensible and flexible architecture
- Tools, Examples, Documentation, …
- A good one to start with Web Services.

Open-source, Apache Software Foundation

RUNNING APACHE AXIS

Installation

Apache Tomcat (version 4.1.x)

- http://jakarta.apache.org/tomcat/

Java

- http://java.sun.com/

Apache Axis

- http://ws.apache.org/axis/

SIMPLE EXAMPLE

```
public class Calculator{
  public int somar(int numA, int numB){
        return numA + numB;
  }
  public int subtrair(int numA, int numB){
        return numA - numB;
  }

  public int multiplicar(int numA, int numB){
        return numA * numB;
  }
}
```

PUBLISHING WS WITH AXIS

Depends on the use of complex types.

- Primitive Types
  - **Copy** the ".java" file to TomCat's **webapps** folder and change its extension from ".java" to ".jws"
- Complex Types
  - Should provide the server with the **ability to deserialize / serialize** complex objects

DEPLOYMENT DESCRIPTORS

JWS is quite simple, but it has limitations:

- Source code
- Cannot specify handlers, mappings, dispatchers

WSDD (Web Services Deployment Descriptors) is a mechanism that facilitates the installation and configuration of a WS allowing:

- Mapping Type
- Different types of transport – HTTP/S and CP/IP
- Binary Attachments

PUBLISHING A WS USING WSDD

- We've **written** a **WSDD** with the configuration data we want for our Web Service.
- We copy the ".class" files that we use in the Web Service to the /WEB-INF/classes folder of Axis.
- We performed a WSDD deploy using the **Axis AdminClient** tool with the following command:

Java org.apache.axis.client AdminClient deploy.wsdd

CONSUMING THE WEB SERVICE (CLIENT)

Dynamic Invocation Interface (DII)

Generation Stubs from Service WSDL description

Packages

- axis.jar
- jaxrpc.jar
- commons-logging.jar
- commons-discovery.jar
- saaj.jar
- wsdl4j.jar

CONSUMING A WS USING DII

```
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.namespace.QName;

public class CalculadoraClient {
  public static void main(String [] args) {
    try {
      String endpoint = "http://localhost:8080/axis/calculadora.jws";
      Service  service = new Service();
      Call   call = (Call) service.createCall();
      call.setOperationName(new QName(endpoint, "somar"));
      call.setTargetEndpointAddress( new java.net.URL(endpoint) );
      Integer ret = (Integer)call.invoke(new Object[]{new Integer(5), new Integer(6)});
      System.out.println("somar(5, 6) = " + ret);
    } catch (Exception e) {
         System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```

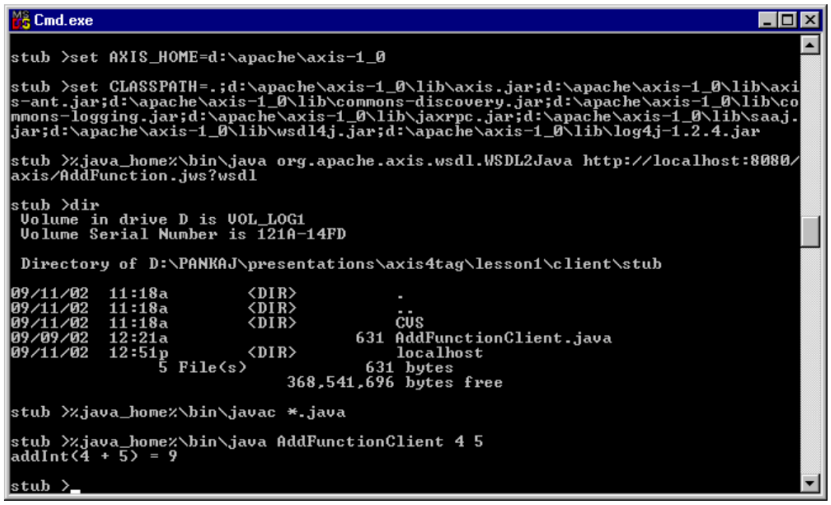
CONSUMING A WS USING STUBS

Generate the stubs:

```
java org.apache.axis.wsdl.WSDL2Java \
  http://localhost:8080/axis/Calculadora.jws?wsdl
```

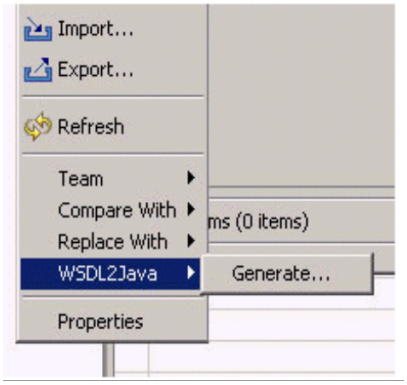
```
Import localhost.*;
public class CalculadoraClient{
  public static void main(String [] args) {
    try {
      CalculadoraService calcSF = new CalculadoraServiceLocator();
      Calculadora calc = calcSF.getCalculadora();
      System.out.println("somar(5, 3) = " + calc.somar(5, 3));
    } catch (Exception e) {
      System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```

GENERATING STUBS, COMPILING AND RUNNING THE STUB CLIENT



CONSUMING WS USING WSDL2JAVA

Right-click on the file, select the WSDL2Java option and then click Generate. Classes will be created in a standard package.



ALPINE: THE ALTERNATIVE PROPOSED

Embrace XML & XPath

Use the latest XML tools

Forget rpc/encoded SOAP

Queued/Asynchronous API

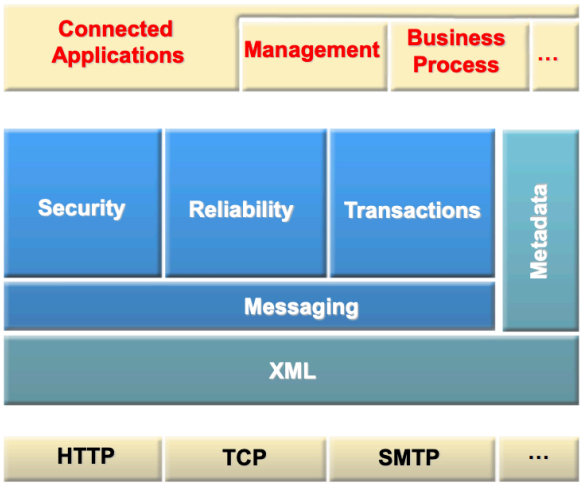
WSDL-first

NEW PROTOCOLS

Current limitations:

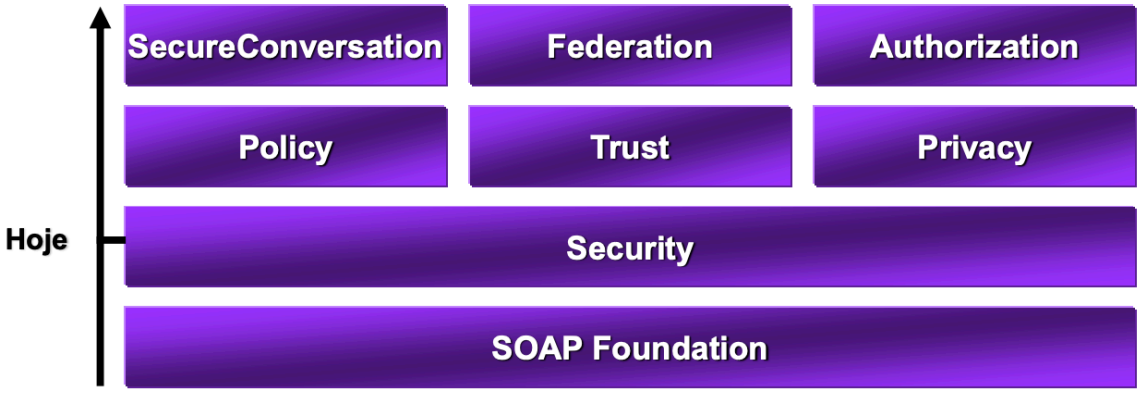
- Security?
- Delivery Guarantees?
- Transactions?
- Other transports?
- Asynchronous Messages (One-Way)?
- Routing/Addressing?
- Other standards (Ex: Pub/Sub)?

WEB SERVICES PROTOCOLS (WS-*)



- **Messaging Specifications**
  SOAP
  WS-Addressing
  MTOM (Attachments)
  WS-Eventing
- **Security Specifications**
  WS-Security
  WS-SecureConversation
  WS-Trust
  WS-Federation
  WS-Federation Active Requestor Profile
  WS-Federation Passive Requestor Profile
  Web Services Security Kerberos Binding
- **Reliable Messaging Specifications**
  WS-ReliableMessaging
- **Transaction Specifications**
  WS-Coordination
  WS-AtomicTransaction
  WS-BusinessActivity
- **Metadata Specifications**
  WSDL

WEB SERVICES SECURITY ROADMAP



CONCLUSION

There is a lot of advertising on the Web Services

- Text format greatly increases overhead
- There is no "**neutrality**" of transport
- **Different versions** of patterns
- There are many patterns that “interact”, but don't define any restrictions to interact
- UDDI **does not meet** the needs
- Automatic **generation tools get in the way**
- There is **no 100%** interoperability

IV. Maintenance and Security

1. Information and Software Security

Information and Software Security

CONCEPTS

For Stallings and Brown (2015, p. 1.1) the main objective of information security is, therefore, to answer three basic questions:

1. What are the assets we need to protect?

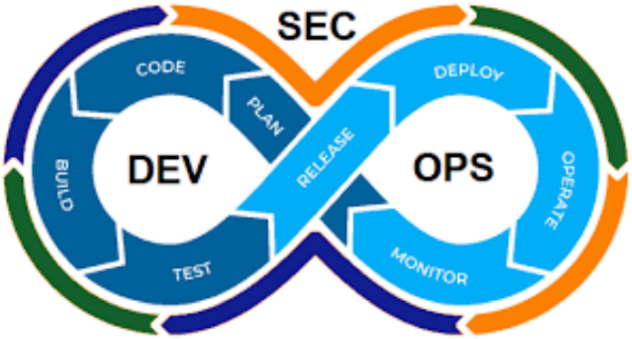
2. What threats exist for these assets?

3. What can we do to combat these threats?

DEVOPS

What is secure DevOps?

Secure DevOps, also called DevSecOps or Rugged DevOps by professionals, is a term often used to describe DevOps practices that include security checks and analysis across the entire software production pipeline. It's no coincidence that "Sec" is in the middle



Scenario 1: The application uses unverified data in an SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not verified correctly, an attacker can access any user's account.

```
https://example.com/app/accountInfo?acct=notmyacct
```

Scenario 2: An attacker simply forces navigation to destination URLs. Administrator rights are required to access the administrator page.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

If an unauthenticated user can access any page, it's a failure. If a non-admin can access the admin page, this is a failure.

OWASP

OWASP (Open Application Security Project) is a non-profit charitable organization and was founded in the USA in 2004. OWASP, as an open online community, creates and makes freely available articles, methodologies, documentation, tools and technologies in the field of web application security.

OWASP is an umbrella of themes, best practices and methodologies on software development.

FUNDAMENTALS OF SECURE SOFTWARE

1. “Weak link protection

– The defensive security of the system must be seen as an interconnected chain, attackers will look for breaches in the weakest link in this chain.

2. Defense in depth

– Manage risk defensively across all layers and components so that if one fails, the other has a fair chance of being able to function.

3. Fail safely

– Handle faults and errors correctly and safely. Mishandling failures often causes security breaches in terms of reliability.

4. Minimum privilege

– Run the software with the least privileges necessary to reduce the potential impact of an attack.

5. Compartmentalize accesses and areas

– Trying to limit the damage of attacks by compartmentalizing areas using VPNs, containers, firewalls…

6. Keep it simple Keep It Simple (KISS)

– The more complex a system, the more difficult it will be to maintain and the greater the security issues.

7. Promote Privacy

– Protecting personal and system data is extremely important. Exposure of sensitive information causes many attacks and breaches of trust.

8. Hiding secrets is hard

– Even the best security algorithms are broken and reliance on secrecy is the source of many security flaws.

9. Trust by distrusting

– Security is based on trust from a source, but even if the source is trusted, you should never fully trust anyone or anything, and you should always take countermeasures to keep the software more secure.

10. Use community resources

– If something is well-designed and has already been scrutinized by the community, it's probably a good source of security.

The table shows, in a simplified way, the classification of the frequency of attacks (OWASP, 2018), which were identified in the previous year and focused on web applications and the level of risk they could cause to the IS in the case of vulnerabilities be explored.

Pos	Vulnerabilities	Risk	Countermeasures
1.	Injection (SQL, PHP, LDAP, OS)	Medium/High	Input Sanitization, Server Whitelisting, Secure API's and Parameterized Query
2.	Authentication break	Medium/High	Multi-Factor Authentication, Session Isolation, Timeout for Inactive Sessions and Safe Cookies
3.	Exposure of sensitive data	Medium	Encrypt all data, Use secure protocols and algorithms, and Disable sensitive data caching
4.	XML Esternal Entities (XXE)	Medium	Avoid sensitive data serialization, Server whitelisting for XML loading, WAF for XXE blocking and code review
5.	Access control break	High	Invalidation of Cookies and Tokens after logout, Force Login/Logout after credential changes, server restrictions and profile-based restrictions and resources
6.	security errors	High	Ensure that no default values are used in hardware and software, Install only necessary components and Review security settings periodically
7.	Cross Site Scripting (XSS)	Medium/High	Output encoding and escaping unreliable characters and using CSP
8.	Unsecured Sealization	Medium/High	Encrypt serialized data and run with least privileges for deserializers
9.	Components with known vulnerabilities	Medium/High	Periodic updates and latest CVE and mitigations
10	Insufficient monitoring and logging	Medium	Monitor and analyze traffic and events 24x7 and practice effective incident response

Cap 1 → Introduction
Cap 2 → Architectures
Cap 3 → Processes
Cap 4 → Communication
Cap 5 → Naming
Cap 6 → Synchronization
Cap 7 → Consistency and Replication
Cap 8 → Fault Tolerance
Cap 9 → Security
Cap 10 → Distributed Object-based Systems
Cap 11 → Distributed File Systems
Cap 12 → Distributed Web-based Systems
Cap 13 → Distributed Coordination-based Systems