2 System models

2.1 Outline

What are the three basic ways to describe Distributed systems? –

- Physical models – consider DS in terms of hardware – computers and devices that constitute a system and their interconnectivity, without details of specific technologies
- Architectural models – describe a system in terms of the computational and communication tasks performed by its computational elements. Client-server and peer-to-peer most commonly used
- Fundamental models – take an abstract perspective in order to describe solutions to individual issues faced by most distributed systems
  - interaction models
  - failure models
  - security models

Difficulties and threats for distributed systems:

- Widely varying modes of use
- Wide range of system environments
- Internal problems
- External threats

2.2 Physical models

- Baseline physical model – minimal physical model of a distributed system as an extensible set of computer nodes interconnected by a computer network for the required passing of messages.

Three generations of distributed systems

- Early distributed systems
  - 10 and 100 nodes interconnected by a local area network
  - limited Internet connectivity
  - supported a small range of services e.g.
    - shared local printers
    - file servers
    - email

    * file transfer across the Internet

- Internet-scale distributed systems
  - extensible set of nodes interconnected by a network of networks (the Internet)
- Contemporary DS with hundreds of thousands nodes + emergence of:
  - mobile computing
    * laptops or smart phones may move from location to location – need for added capabilities (service discovery; support for spontaneous interoperation)
  - ubiquitous computing
    * computers are embedded everywhere
  - cloud computing

  - pools of nodes that together provide a given service

- Distributed systems of systems (ultra-large-scale (ULS) distributed systems)

- significant challenges associated with contemporary DS:

Figure 2.1 Generations of distributed systems

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

2.3 Architectural Models

Major concerns: make the system *reliable*, *manageable*, *adaptable* and *cost-effective*

2.3.1 Architectural elements

- What are the entities that are communicating in the distributed system?
- How do they communicate, or, more specifically, what communication paradigm is used?
- What (potentially changing) roles and responsibilities do they have in the overall architecture?
- How are they mapped on to the physical distributed infrastructure (what is their placement)?

Communicating entities

- From system perspective: **processes**
  - in some cases we can say that:
    - **nodes** (sensors)
    - **threads** (endpoints of communication)
- From programming perspective
  - ***objects***
    - computation consists of a number of interacting objects representing natural units of decomposition for the given problem domain
    - Objects are accessed via interfaces, with an associated interface definition language (or IDL)

- ***components*** – emerged due to some weaknesses with distributed objects
  - offer problem-oriented abstractions for building distributed systems
  - accessed through interfaces
    - \+ assumptions to components/interfaces that must be present (i.e. making all dependencies explicit and providing a more complete contract for system construction.)
- ***web services***
  - closely related to objects and components
  - intrinsically integrated into the World Wide Web
    - using web standards to represent and discover services

The World Wide Web consortium (W3C):

> Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artefacts. A Web service supports direct interactions with other software agents using XML-based message exchanges via Internet-based protocols.

- objects and components are often used within an organization to develop tightly coupled applications
- web services are generally viewed as complete services in their own right

Communication paradigms

What is:

- interprocess communication?
- remote invocation?
- indirect communication?

Interprocess communication – low-level support for communication between processes in distributed systems, including *message-passing* primitives, direct access to the API offered by Internet protocols (socket programming) and support for *multicast communication*

Remote invocation – calling of a remote operation, procedure or method

Request-reply protocols – a pattern with message-passing service to support client-server computing

Remote procedure call (RPC)

- procedures in processes on remote computers can be called as if they are procedures in the local address space
- supports client-server computing with servers offering a set of operations through a service interface and clients calling these operations directly as if they were available locally
  - RPC systems offer (at a minimum) access and location transparency

Remote method invocation (RMI)

- strongly resemble RPC but in a world of distributed objects
- tighter integration into object-orientation framework

In RPC and RMI –

- senders-receivers of messages
  - coexist at the same time
  - are aware of each other's identities

Indirect communication

- Senders do not need to know who they are sending to (*space uncoupling*)
- Senders and receivers do not need to exist at the same time (*time uncoupling*)

Key techniques in indirect communication:

- Group communication
- Publish-subscribe systems:

  - (sometimes also called distributed event-based systems)
  - publishers distribute information items of interest (events) to a similarly large number of consumers (or subscribers)

- Message queues:
  - (publish-subscribe systems offer a one-to-many style of communication), message queues offer a point-to-point service
  - producer processes can send messages to a specified queue
  - consumer processes can
    * receive messages from the queue or
    * be notified

- Tuple spaces (also known as generative communication):
  - processes can place arbitrary items of structured data, called tuples, in a persistent tuple space

- other processes can either read or remove such tuples from the tuple space by specifying patterns of interest
- readers and writers do not need to exist at the same time (Since the tuple space is persistent)

- Distributed shared memory (DSM):
  - abstraction for sharing data between processes that do not share physical memory

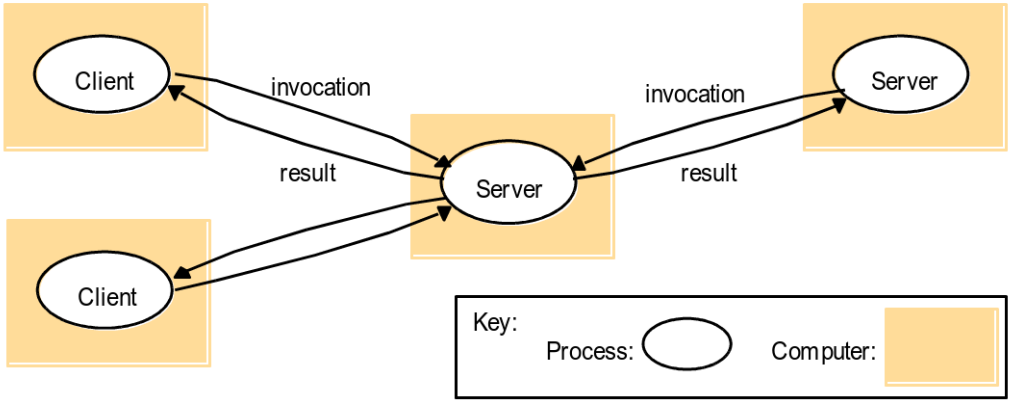
Figure 2.2 Communication entities and communication paradigms

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Roles and responibilities

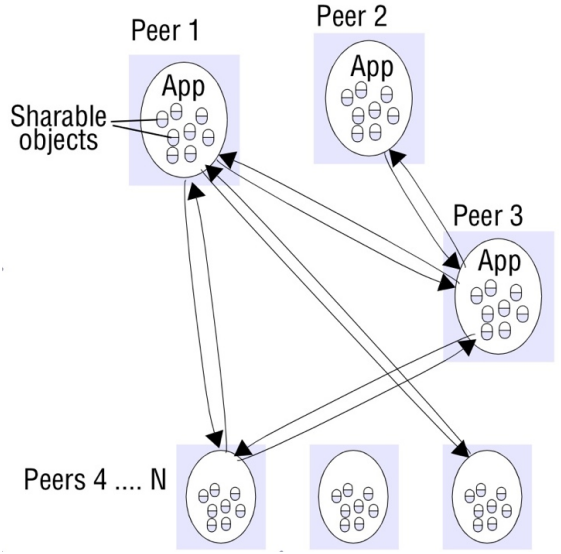
- Client-server

Figure 2.3 Clients invoke individual servers



- Peer-to-peer

Figure 2.4a Peer-to-peer architecture



  - same set of interfaces to each other

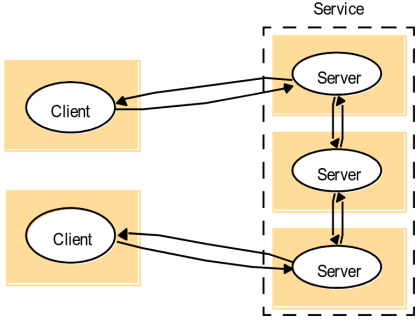
Placement

- crucial in terms of determining the DS properties:
  - performance
  - reliability
  - security

Possible placement strategies:

- *mapping of services to multiple servers*
  - mapping distributed objects between servers, or
  - replicating copies on several hosts
  - more closely coupled multiple-servers – cluster

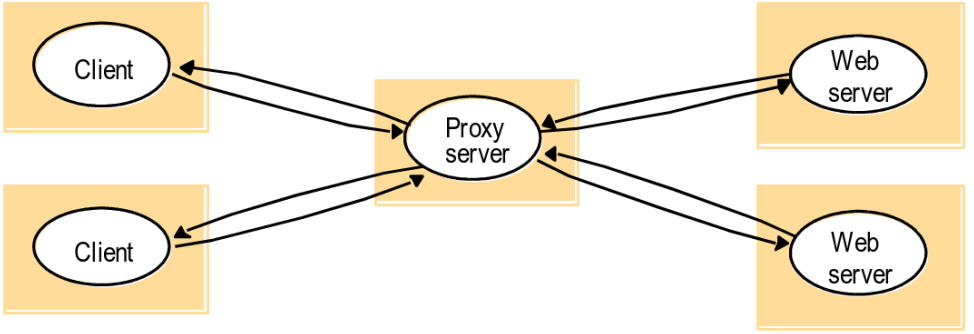
Figure 2.4b A service provided by multiple servers



- *caching*

  – A cache is a store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves

Figure 2.5 Web proxy server

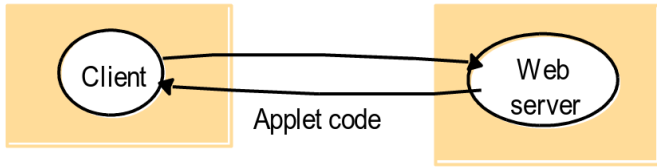


- *mobile code*

  – Applets are an example of mobile code

  Figure 2.6 Web Applets

  a) client request results in the downloading of applet code



  b) client interacts with the applet

  – yet another possibility – *push* model: server initiates interaction (e.g. on information updates on it)

- *mobile agents*

  - Mobile agent – running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone's behalf (e.g. collecting information), and eventually returning with the results.

  - could be used for
    - software maintenance
    - collecting information from different vendors' databases of prices

Possible security threats with mobile code and mobile agents...

2.3.2 Architectural patterns

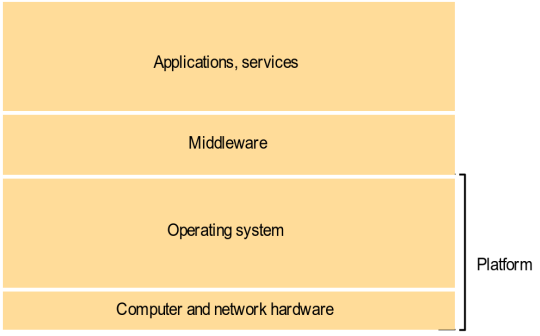
Layering

Layered approach – complex system partitioned into a number of layers:

- vertical organisation of services
- given layer making use of the services offered by the layer below
- software abstraction
- higher layers unaware of implementation details, or any other layers beneath them

Platform and Middleware

Figure 2.7 Software and hardware service layers in distributed systems



- A platform for distributed systems and applications consists of the lowest-level hardware and software layers.

- Middleware – a layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers.

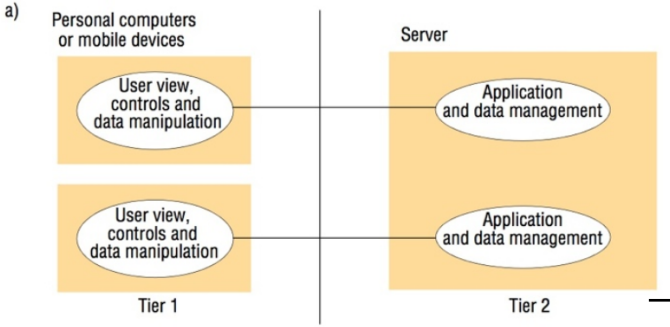
Tiered architecture

Tiering is a technique to organize functionality of a given layer and place this functionality into appropriate servers and, as a secondary consideration, on to physical nodes

Example: two-tier and three-tier architecture
functional decomposition of a given application, as follows:

- presentation logic
- application logic
- data logic

Figure 2.8 Two-tier and three-tier architectures



- three aspects partitioned into two processes
- (+) low latency
- (-) splitting application logic

- (+) one-to-one mapping from logical elements to physical servers
- (-) added complexity, network traffic and latency

AJAX (Asynchronous Javascript And XML) – a way to create interactive, partially/selectively-updatable webpages

- extension to the standard client-server style of interaction in WWW
  - Javascript forntend and server-based backend

Figure 2.9 AJAX example: soccer score updates

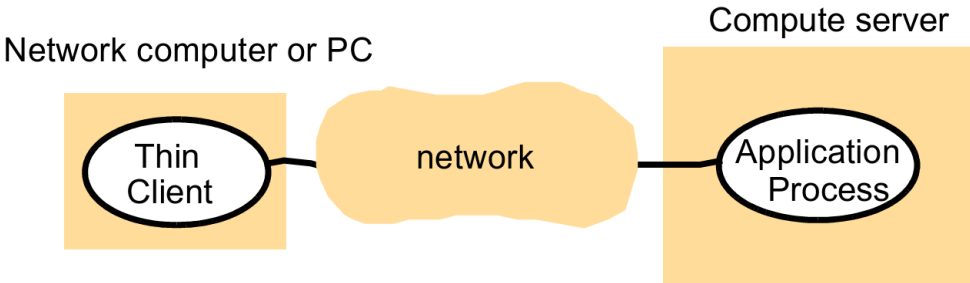
```
new Ajax.Request('scores.php?game=Arsenal:Liverpool',
  {onSuccess: updateScore});
  function updateScore(request) {
  .....
    ( request contains the state of the Ajax request including the returned result.
      The result is parsed to obtain some text giving the score, which is used
      to update the relevant portion of the current page.)
  .....
  }
```

(two-tier architecture)

Thin clients

- enabling access to sophisticated networked services (e.g. cloud services) with few assumptions to client device
- software layer that supports a window-based user interface (local) for executing remote application programs or accessing services on remote computer

Figure 2.10 Thin clients and computer servers

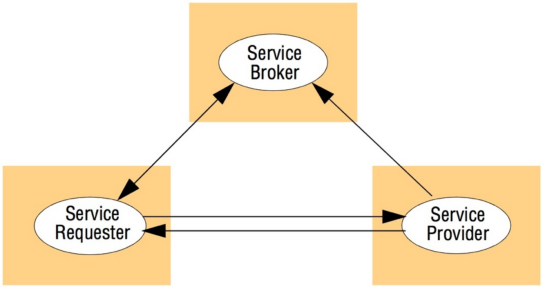


Concept led to Virtual Network Computing (VNC) – VNC clients accessing VNC servers using VNC protocol

Other commonly occurring patterns

- ***proxy* pattern**
  - designed to support location transparency in RPC or RMI
  - proxy created in local address space, with same interface as the remote object
- ***brokerage* in web services**
  - supporting interoperability in potentially complex distributed infrastructures
  - service provider, service requestor and service broker
  - brokerage reflected e.g. in registry in Java RMI and naming service in CORBA

Figure 2.11 The web service architectural pattern



- ***Reflection* pattern**
  - a means of supporting both:
    - introspection (the dynamic discovery of properties of the system)
    - intercession (the ability to dynamically modify structure or behaviour)
  - used e.g. in Java RMI for generic dispatching
  - ability to intercept incoming messages or invocations

- dynamically discover interface offered by a given object
- discover and adapt the underlying architecture of the system

2.3.3 Associated middleware solutions

The task of middleware is to provide a higher-level programming abstraction for the development of distributed systems and, through layering, to abstract over heterogeneity in the underlying infrastructure to promote interoperability and portability.

Categories of middleware

Figure 2.12 Categories of middleware

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Limitations of middleware

Some communication-related functions can be completely and reliably implemented only with the knowledge and help of the application standing at the end points of the communication system.

Example: e-mail transfer need another layer of fault-tolerance that even TCP cannot offer

2.4 Fundamental models

What is:

- Interaction model?
- Failure model?
- Security model?

2.4.1 Interaction model

- processes interact by passing messages –
  - communication (information flow) and
  - coordination (synchronization and ordering of activities) between processes

- communication takes place with delays of considerable duration
  - accuracy with which independent processes can be coordinated is limited by these delays
  - and by difficulty of maintaining the same notion of time across all the computers in a distributed system

Behaviour and state of DS can be described by a *distributed algorithm:*

- steps to be taken by each interacting process
- + transmission of messages between them

State belonging to each process is completely private

Performance of communication channels

- *latency* – delay between the start of message's transmission from one process and the beginning of receipt by another
- *bandwidth* of a computer network – the total amount of information that can be transmitted over it in a given time
- *Jitter* – the variation in the time taken to deliver a series of messages

Computer clocks and timing events

- *clock drift rate* – rate at which a computer clock deviates from a perfect reference clock

Two variants of the interaction model

Synchronous distributed systems:

- The time to execute each step of a process has known lower and upper bounds
- Each message transmitted over a channel is received within a known bounded time
- Each process has a local clock whose drift rate from real time has a known bound

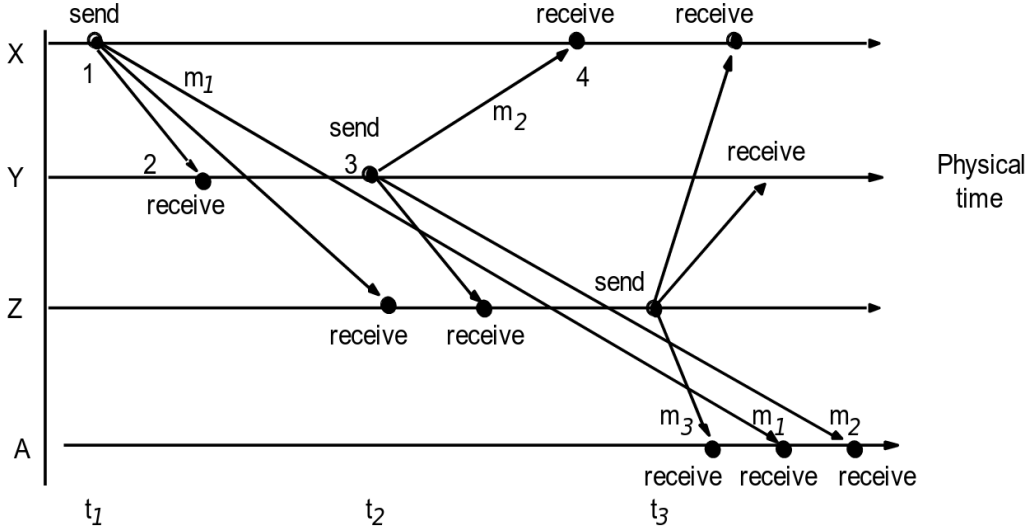
Asynchronous distributed systems:

No bounds on:

- Process execution speeds
- Message transmission delays
- Clock drift rates

Event ordering

Figure 2.13 Real-time ordering of events



- *Logical time* – based on event ordering

2.4.2 Failure model

- faults occur in:
  - any of the computers (including software faults)
  - or in the network
- Failure model defines and classifies the faults

Omission failures

- process or communication channel fails to perform actions it is supposed to do

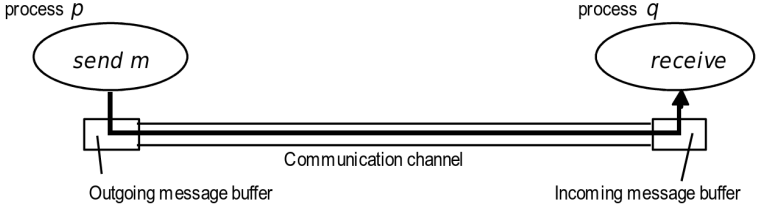
Process omission failures

- cheaf omission failure of a process is to crash
  - crash is called *fail-stop* if other processes can detect certainly that the process has crashed

Communication omission failures

- communication channel does not transport a message from p's outgoing message buffer to q's incoming message buffer
  - known as dropping messages
    - send-omission failures
    - receive-omission failures
    - channel-omission failures

Figure 2.14 Processes and channels



All failures so far: ***benigh failures***

Arbitrary failures

arbitrary or ***Byzantine failure*** is used to describe the worst possible failure semantics, in which any type of error may occur

Figure 2.15 Omission and arbitrary failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing failures

- applicable in synchronous distributed systems

Figure 2.16 Timing failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Masking failures

- knowledge of the failure can enable a new service to be designed to mask the failure of the components on which it depends

Reliability of one-to-one communication

- reliable communication:
  - *Validity:* Any message in the outgoing message buffer is eventually delivered to the incoming message buffer
  - *Integrity:* The message received is identical to one sent, and no messages are delivered twice

2.4.3 Security model

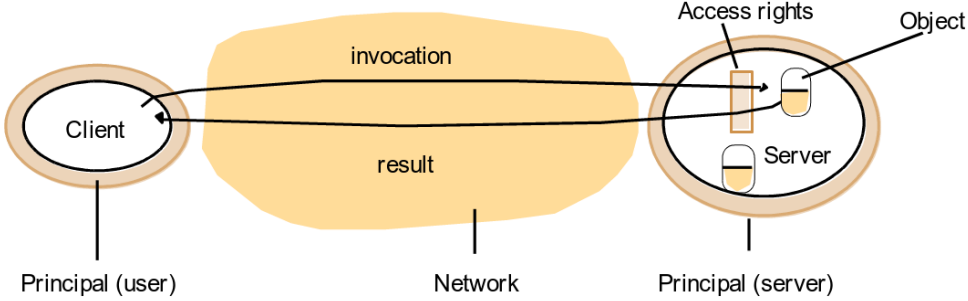
- modular nature of distributed systems and their openness exposes them to attack by
  - both external and internal agents
- Security model defines and classifies attack forms,
  - providing a basis for the analysis of threats
  - basis for design of systems that are able to resist them

the security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access.

Protecting objects

- Users with access rights
- association of each invocation and each result with the authority on which it is issued
  - such an authority is called *a principal*
    - principal may be a user or a process

Figure 2.17 Objects and principals



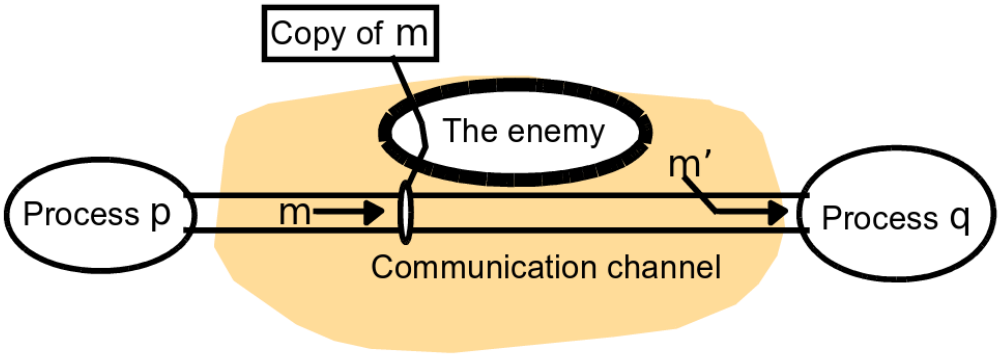
Securing processes and their interactions

- securing communications over open cahnnels
- open service interfaces

The enemy

or also: *adversary*

Figure 2.18 The enemy



Threats to processes

- lack of knowledge of true source of a message
  - problem both to server and client side
  - example: spoofing a mail server

Threats to communication channels

- threat to the privacy and integrity of messages
- can be defeated using *secure channels*

Defeating security threats

Cryptography and shared secrets

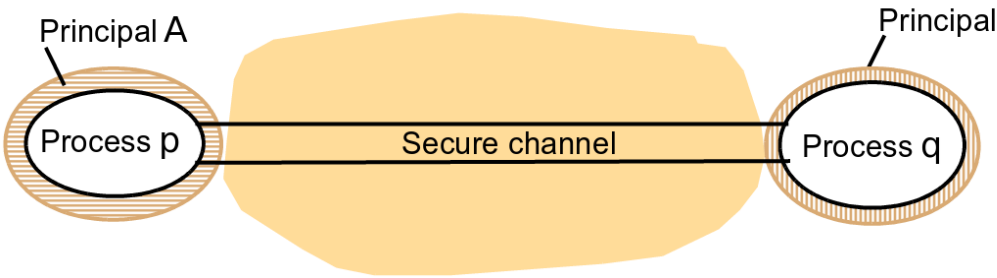
- Cryptography is the science of keeping messages secure
- Encryption is the process of scrambling a message in such a way as to hide its contents

Authentication

- based on shared secrets authentication of messages – proving the identities supplied by their senders

Secure channels

Figure 2.19 Secure channels



Properties of a secure channel:

- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing
- A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it

- Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered

Other possible threats from an enemy

- Denial of service:
  - the enemy interferes with the activities of authorized users by making excessive and pointless invocations on services or message transmissions in a network, resulting in overloading of physical resources (network bandwidth, server processing capacity)
- Mobile code:
  - execution of program code from elsewhere, such as the email attachment etc.

The uses of security models

Security analysis involves

- the construction of a threat model:
  - listing all the forms of attack to which the system is exposed
  - an evaluation of the risks and consequences of each

End of week 2