IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Interaction and Verification

A. Beugnard

C1

2021

- The notion of interaction
- The notion of interface
- A bit of architecture
- Interaction model
- Contract and interaction
- Interaction diagrams

- Between Human and Machine (HMI)
- Among machines (network, distributed systems)
- Among programs
- Among agents (software)
- Among humans

How to describe/specify an interaction ?

With it, verifications can be made. . .

A problem well stated is on its way to solution

Bergson, XXth

concurrency Many activities, reliable communication, a common clock exists

distribution Slow communication, unreliable, no common clock, global state to build

point of view/aspect	optimistic	pessimistic
concurrency	speed (load-balancing)	interleaving
distribution	fault tolerance (high disponibility)	faults

The core issue : interactions !

Agenda

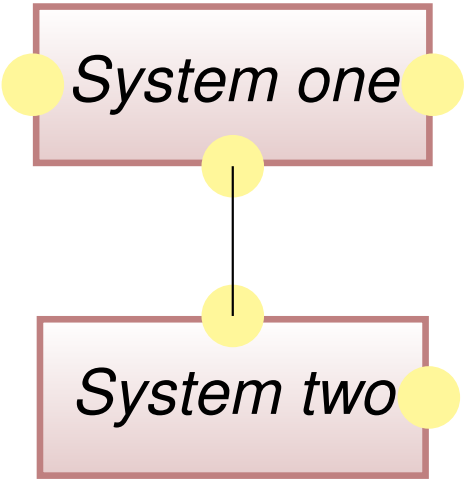
- 2h of abstractions
- 2h of models
- 2h of diagrams

- See references
- Cours de Bernard Espinasse (Univ. Aix-Marseille) : Communication et langages de communication dans les SMA
- Cours de Rachid Guerraoui (EPFL) : Distributed Algorithms

Background

The reciprocal action or influence that can be established between two or more objects (or persons). An interaction is broken down into several sequences, exchanges and turns of speech.

Wikitionnaire



- At least 2 objects/actors/systems
- They exchange (information, material, energy, . . .)

Should the whole system be described ? individual objects/actors/systems ?

- *Interaction*, first case
- *Interface*, second case

An *interaction* describes exchanges among systems.
Systems offer *interfaces*.
Interfaces describes rules, assumptions, *context* (assumed, legitimate) of interaction.

Choosing an interface (or interaction) description language, means choosing rules, constraints, properties

Interface

- What is an interface ?
- What is a system ?
- Why is it interesting
  - Frequent
  - Useful (abstract)
  - Absent or ill defined, implies big issues during integration, test or even in function

water/air ; cell ; cables ; chip ; API ; mechanic - exploded view ; HMI

Famous issues

- A380 delay
- US/EU Satellite
- Coupling of 2 TGV trains

Due to the complexity of the wiring : 530km, 100000 cables and 40300 connectors.
Source : Cadalyst magazine article, *What Grounded the Airbus A380 ?* 6 Décembre 2006 By : Kenneth Wong [1]
Personal information, unofficial sources : the German and French designers German and French designers were not using the same version of the Katia software, which caused alignment errors.

1. `http://www.cadalyst.com/management/what-grounded-airbus-a380-5955`

Point at the border between two elements, through which exchanges and interactions take place

Examples

- Human - Human
- Human - Computer/System (HMI)
- System - System

- What purpose ?
  - Control a system
  - Interact with a system (ask for a service)
- How is it described ?
  - A schema, a plan, an instruction manual, a contract, etc.
- How is it used ?

- Ensuring coupling
- Ensuring interoperability
- Get both parties to agree
  - Data type, unit, message order (protocol), intensity, etc.

- Multiplying couplings
- Reducing couplings
  - Mediator
  - Pivot

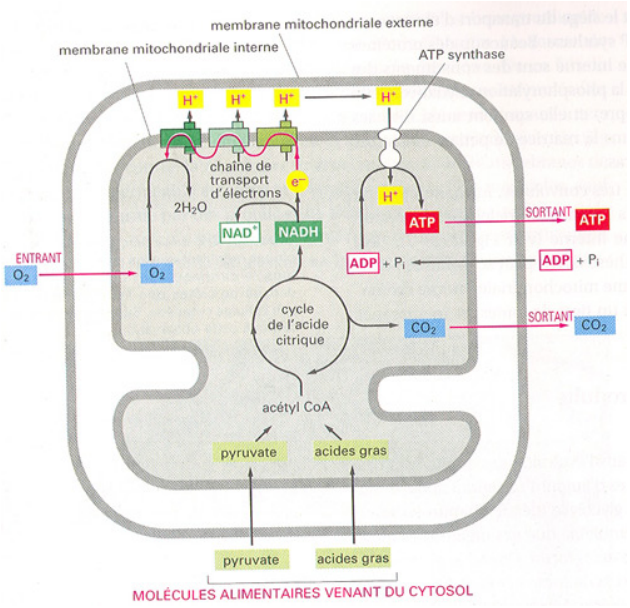
Example :
N languages, need N(N-1) translations or 2(N-1) with a pivot ?

Interface structure

Many points of view

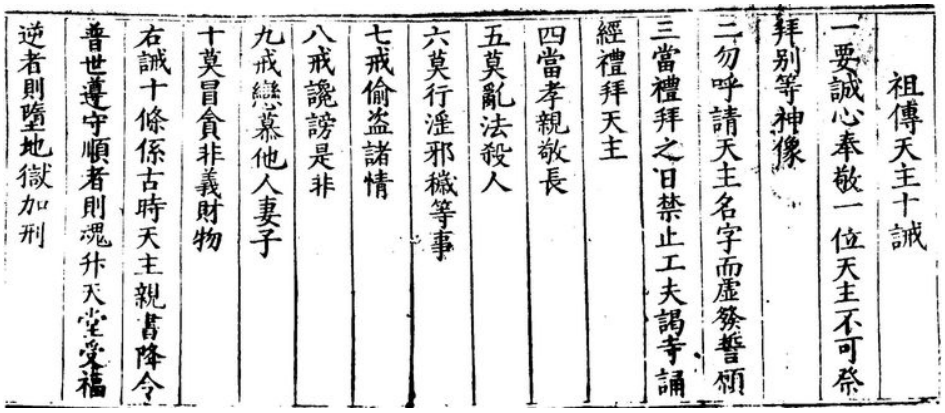
- Physics/Chemistry
- Science of communication
- Electronics
- Computer science

Various description tools. . .



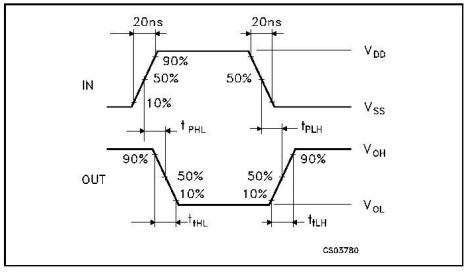
`http://www.humans.be/pages/biomitochondrie.htm`

1. Exchange of compounds


祖傳天主十誡
一要誠心奉敬一位天主不可祭拜別等神像
二勿呼請天主名字而虛發誓願
三當禮拜之日禁止工夫謁寺誦經禮拜天主
四當孝親敬長
五莫亂法殺人
六莫行淫邪穢等事
七戒偷盜諸情
八戒讒謗是非
九戒繼慕他人妻子
十莫冒貪非義財物
右誡十條係古時天主親書降令
普世遵守順者則魂升天堂受福
逆者則墮地獄加刑

http://www.matteo-ricci.org/Opera/m1_decalog.html

1. Physics (the medium : paper, air, electrons, etc.)
2. Orthographic (coding : ASCII, unicode, morse code, jpeg, etc.)
3. Lexicon (words : separators)
4. Grammar (sequencing rules)
5. Functional (meaning, usage, etc.)



`http://eicom.ru/pdf/datasheet/ST_Microelectronics_PDFS/HCF40107B/HCF40107B.html`

`http://www.ti.com/lit/ds/symlink/sn74ls00.pdf`

1. Physics (mechanics, pins, etc.)
2. Logic (signal names, direction)
3. Electrics (voltage/current control, levels/states, switching speeds, hold times, etc.)
4. Protocol

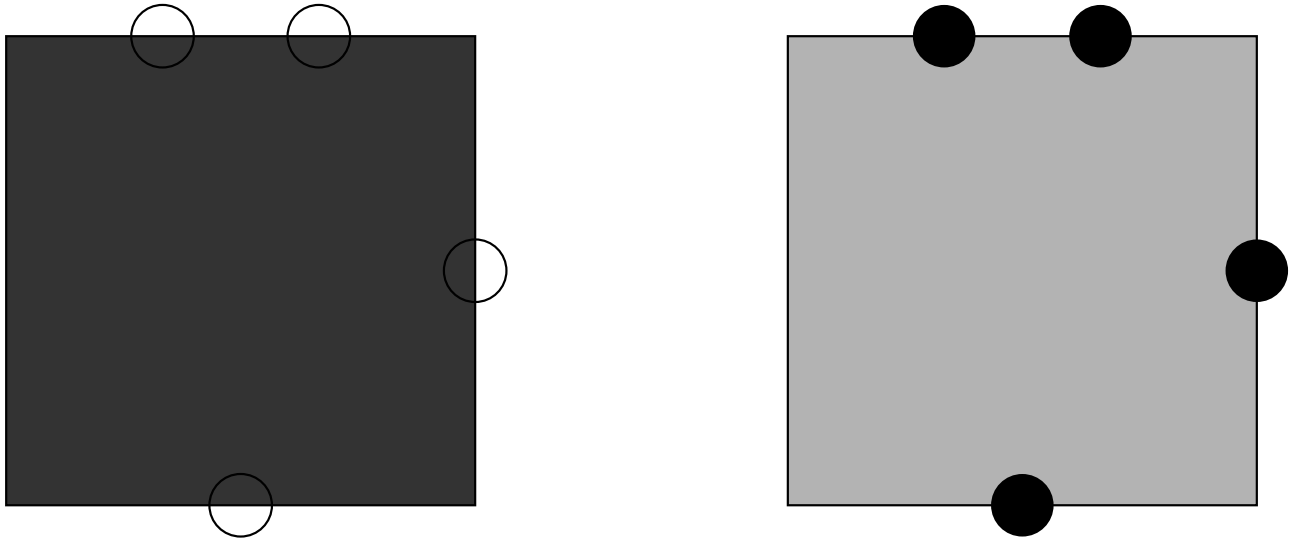
Possible Functions of the Common High-Level API

Category	Function	Description
Management	Open	Open a resource in the system, create a handle to the resource
	Close	Close the resource, free the resource handle
	Reset	Put the resource in a know state
	Execute	Load a resource or cause a process to start
	Claim	Lock access to a shared resource
	Release	Unlock access to a shared resource
Memory Access	MemRead	Read the memory resource (this may be local or remote via a bus)
	MemWrite	Write the memory resource
Link Access	LinkRead	Read the stream resource into a buffer
	LinkWrite	Write to the stream resource from a buffer
Signals	SendSignal	Send a signal to the resource
	WaitSignal	Wait for a signal from the resource
	CallbackConnect	Attach a callback to a resource's signal
	CallbackDisconnect	Detach a callback from a resource's signal

Table 1 Listed here are some of the possible functions of the high-level common layer, which would use something like COM to implement the remappable interfaces to the next layer down, the board services layer. Every device in a system would be given a resource name/handle, allowing it to be addressed and interacted with.

`http://www.rtcmagazine.com/articles/view/100056`

1. Physics (usual hidden)
2. Logic (names)
3. Semantics (meaning)
4. Synchronisation (usage, protocol)
5. Quality of service



Simple cases : a system and its interface(s)
Isolated : Encapsulation. Black box.
In fact grey box ; the interface exposes part of the content by choosing the level of exposure.

Why ?

- Decoupling, isolating, splitting
- Divide and conquer (Complexity mastering)

Consequences :

- Simplified use (for users)
- Separate undestranding (for researchers)
- Have it developed separately (for engineer)

The identification of the right interfaces is linked to the identification of of the right boundaries.

- Physical systems often have natural mechanical boundaries
- Software systems offer more flexibility. (module, class, package, component, aspect, etc.)

A guide (in software engineering) : strong consistency - weak coupling ; responsibility.

Beyond objectives of this lesson. . .But, we'll do a bit of architecture.

Interface specification

Description of all that is necessary for the proper use of the system.

- Mechanics (weight, mechanical strength, dimensions, materials, friction, etc.)
- Electric (voltage, intensity, direction, evolution of the signal, etc.)
- Information (coding, lexicon, direction, order, quality of service, etc.)

Questions : what, how, where, when, who, must find an answer. . .
What about *why* ?

The same questions have different levels of answers :

- The client, the user
- The manufacturing organiser
- The architecte
- The designer
- The maker

Organize questions (ex : Zachman, DODAF)

	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*
Objective/Scope (contextual) *Role: Planner*	List of things important in the business	List of Business Processes	List of Business Locations	List of important Organizations	List of Events	List of Business Goal & Strategies
Enterprise Model (conceptual) *Role: Owner*	Conceptual Data/ Object Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
System Model (logical) *Role:Designer*	Logical Data Model	System Architecture Model	Distributed Systems Architecture	Human Interface Architecture	Processing Structure	Business Rule Model
Technology Model (physical) *Role:Builder*	Physical Data/Class Model	Technology Design Model	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
Detailed Reprentation (out of context) *Role: Programmer*	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Speculation
Functioning Enterprise *Role: User*	Usable Data	Working Function	Usable Network	Functioning Organization	Implemented Schedule	Working Strategy

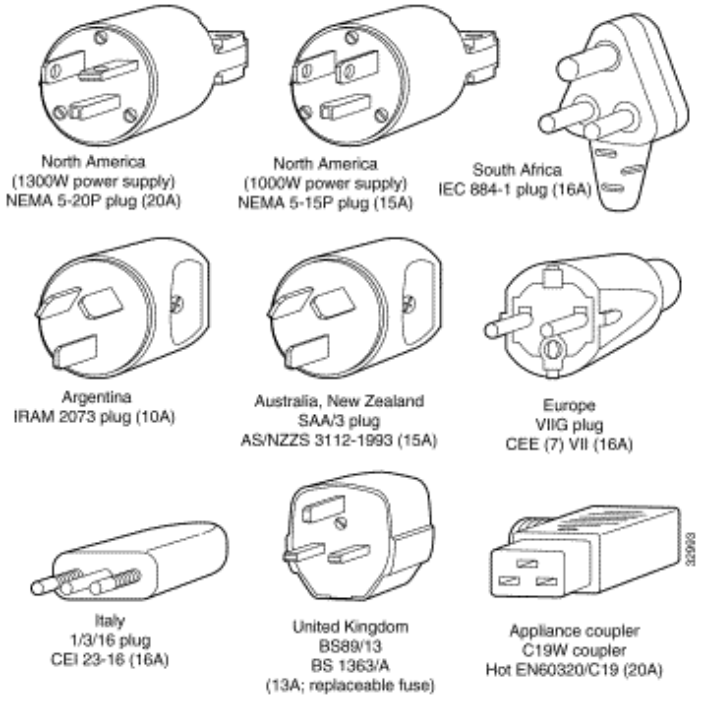
`http://en.wikipedia.org/`

Zachman's *Architecture Framework* does not consider (explicitly) :

- cost of service
- legal responsibility
- service acces rights
- etc.

- Natural language
- Programming language (API)
- Modeling languages (UML, SysML)

Goal : *describe to check connections compatibility*



C code library `sqrt(double):double` What is ensured ? Link edition only. Not semantics.

How ensuring that `sqrt(double):double` computes the square root ?

```
sqrt(x:double):double
 pre: x >=0
 post : result >=0 && x = result * result
```

Precondition and postcondition define the *meaning* of the operation.

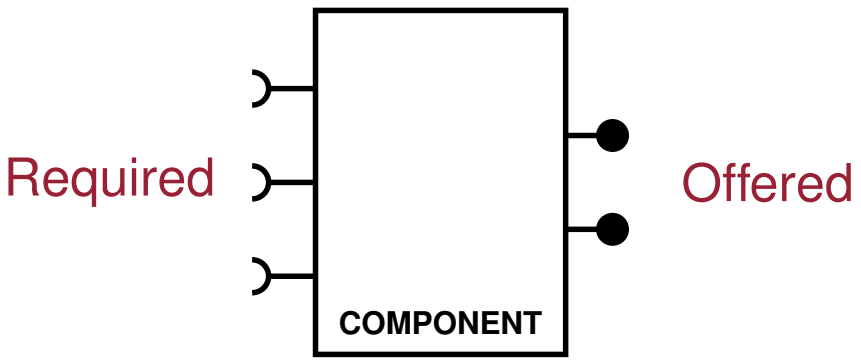
Offered service

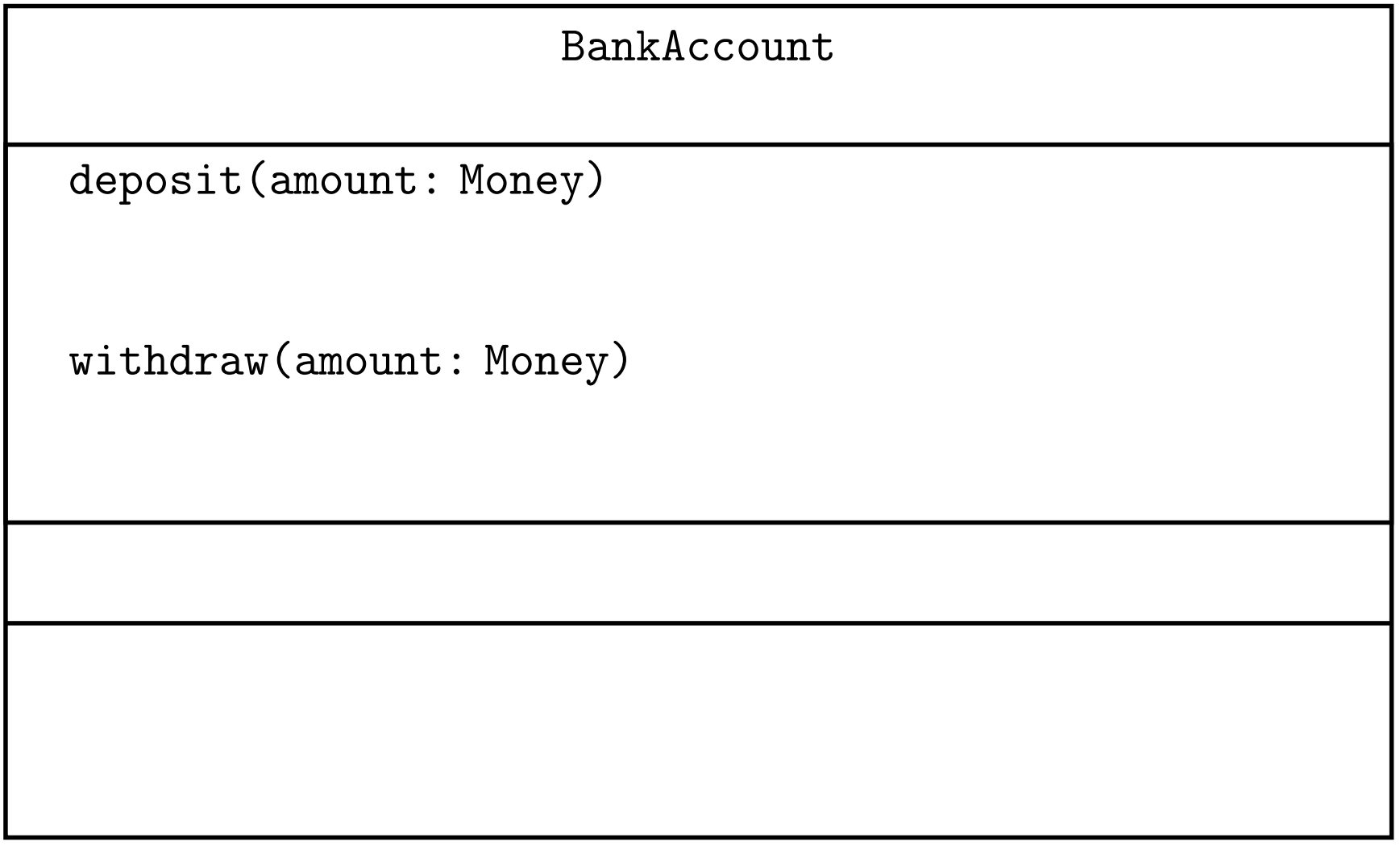
1. Syntactic level : `sqrt(double):double`
2. Semantics level : `pre: post:`
3. Synchronisation level : expected rules (concurrency, allowed sequences, protocol)
4. QoS level : quantitative properties (efficiency, reliability, availability, etc.)

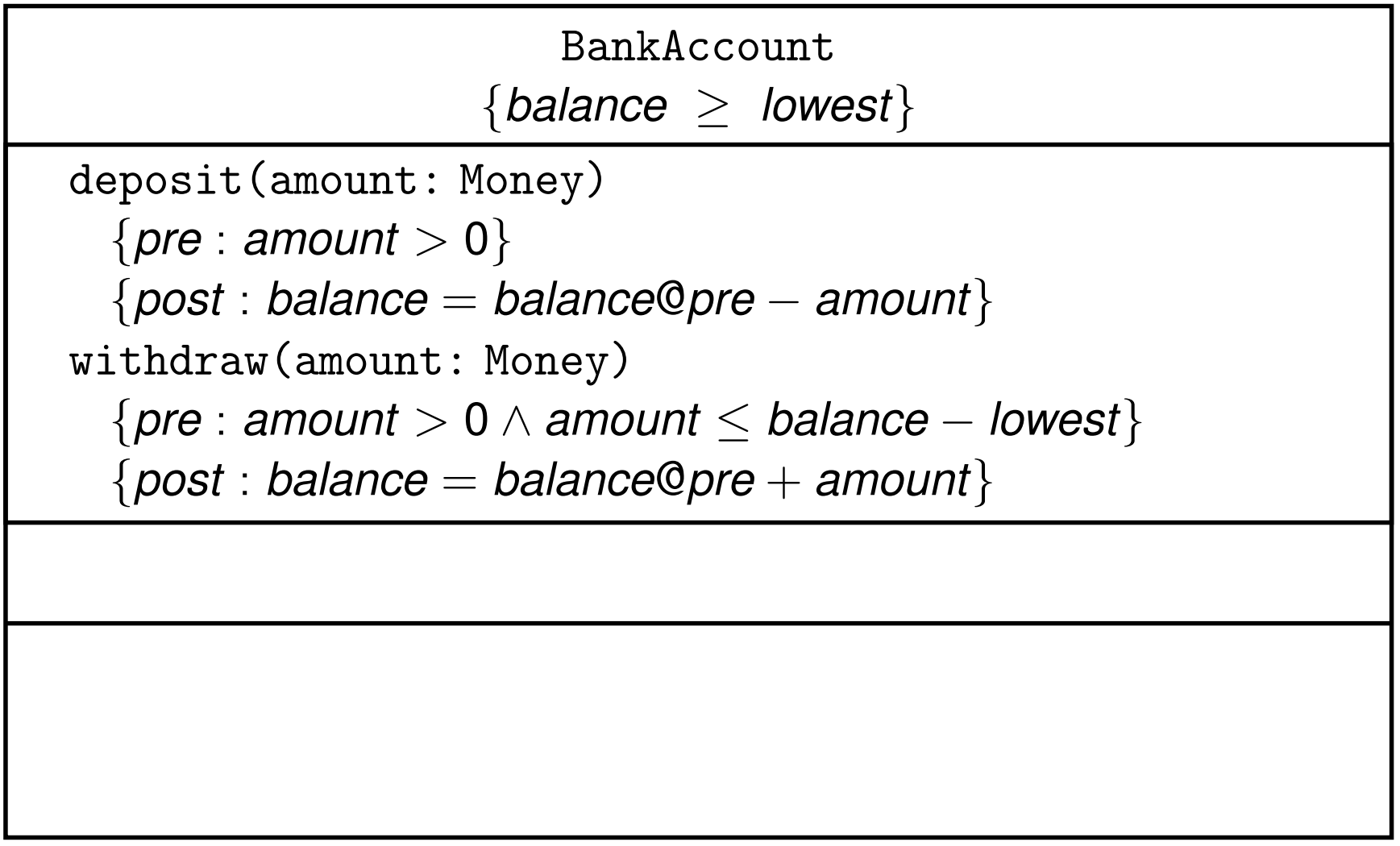
Required service

1. Syntactic level : `sqrt(double):double` – same signature
2. Semantics level : `pre: post:` – verification (static or dynamic)
3. Synchronisation level : usage rules (concurrency, allowed sequences, protocol)
4. QoS level : Niveau QoS : client expectations

- Logic unit (not necessarily unit of deployment)
- With an explicit description of :
  - What it does (offered)
  - What it needs (required)
- more or less formal (documentation, contrast, . . .)
- That can be assembled




BankAccount
deposit(amount: Money)
withdraw(amount: Money)


BankAccount
{balance ≥ lowest}
deposit(amount: Money)
{pre : amount > 0}
{post : balance = balance@pre − amount}
withdraw(amount: Money)
{pre : amount > 0 ∧ amount ≤ balance − lowest}
{post : balance = balance@pre + amount}

BankAccount $\{balance \geq lowest\}$

`deposit(amount: Money)`
$\{pre : amount > 0\}$
$\{post : balance = balance@pre - amount\}$
`withdraw(amount: Money)`
$\{pre : amount > 0 \wedge amount \leq balance - lowest\}$
$\{post : balance = balance@pre + amount\}$

Lifecycle = init.(`deposit + withdraw`)*.close

`BankAccount` $\{balance \geq lowest\}$

`deposit(amount: Money)`
$\{pre : amount > 0\}$
$\{post : balance = balance@pre - amount\}$
`withdraw(amount: Money)`
$\{pre : amount > 0 \wedge amount \leq balance - lowest\}$
$\{post : balance = balance@pre + amount\}$

Lifecycle = init.(`deposit + withdraw`)*.close
ResponseTime(`deposit`) $< 1s$ when *Users* < 1000 *ResponseTime*(`withdraw`) $< 1s$ when *Users* < 1000 Availability(`BankAccount`) all days from 1:00 to 0:00

- Syntactic contrat : languages - types (IDL, . . .)
- Semantics contract : OCL, assertions, Eiffel, JML
- Synchronisation contract : Automats, Protocol State Machine, temporal logics, . . .
- QoS contract : QML

Issues

- Specify with precision
  - . . .just seen
  - Tools ?
- Verify assemblies
  - statically (closed systems)
  - dynamically (open systems)
  - Tools ?
- Ensure evolution
  - An interface change impacts linked systems
  - decoupling specification - implementation
  - Tools ?

Conclusion on interfaces

An interface is a contract : static and dynamic.

- Essential for specifying systems – intermediation
- Ease usage – instructions for use
- Ease assembly
- Guide design/development

The notion of interface is at the heart of systems engineering.
The notion of interface is at the heart of the study of interactions.

Historical example

We identify :

- functions, modules
- interactions
- data (table, tree)
- inputs, outputs

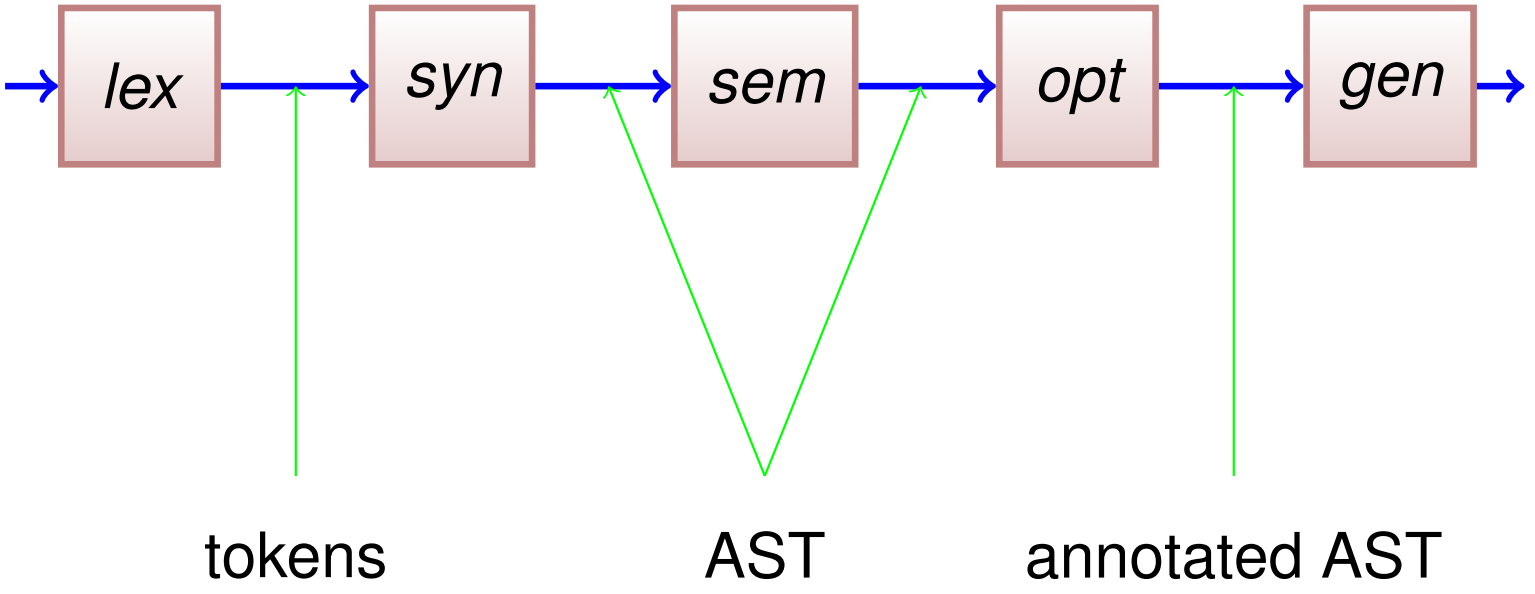
What abstractions ?

- boxes
- lines

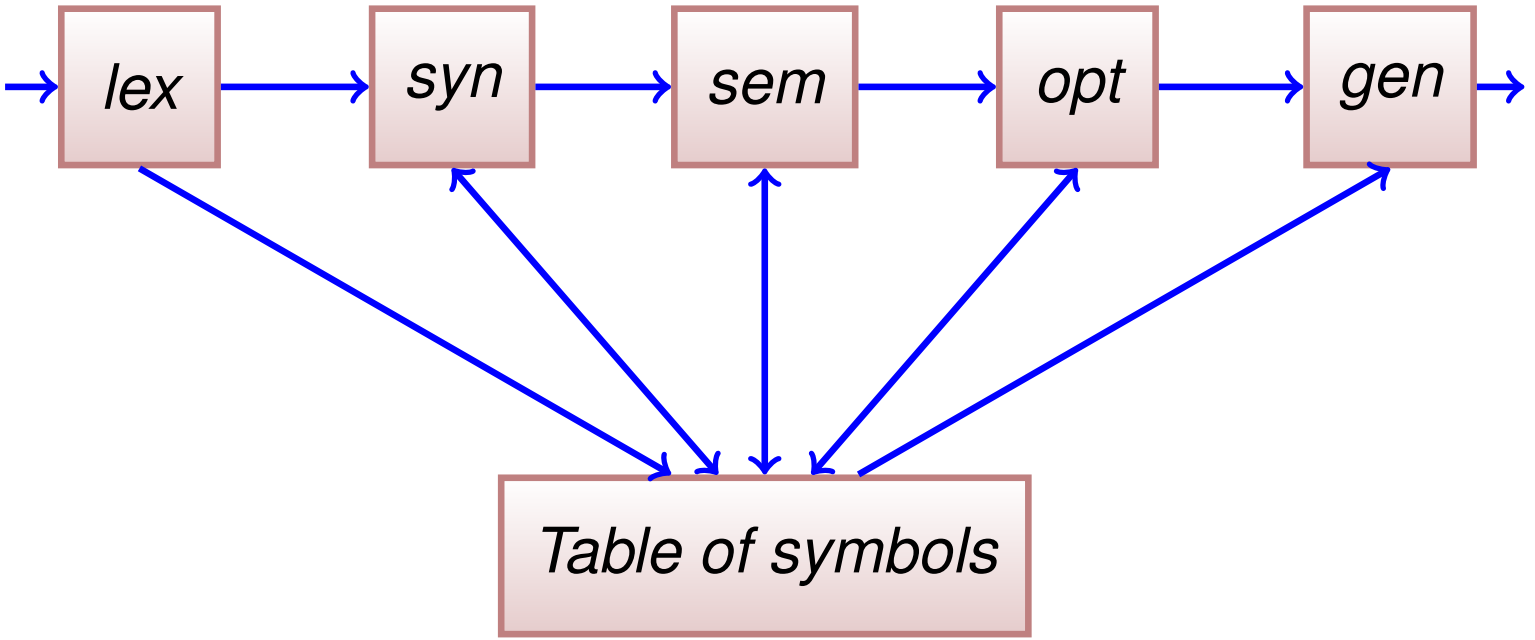
. . .and their properties . . .

Example from « An Introduction to Software Architecture », Garlan et Shaw, 1993 [GS93]

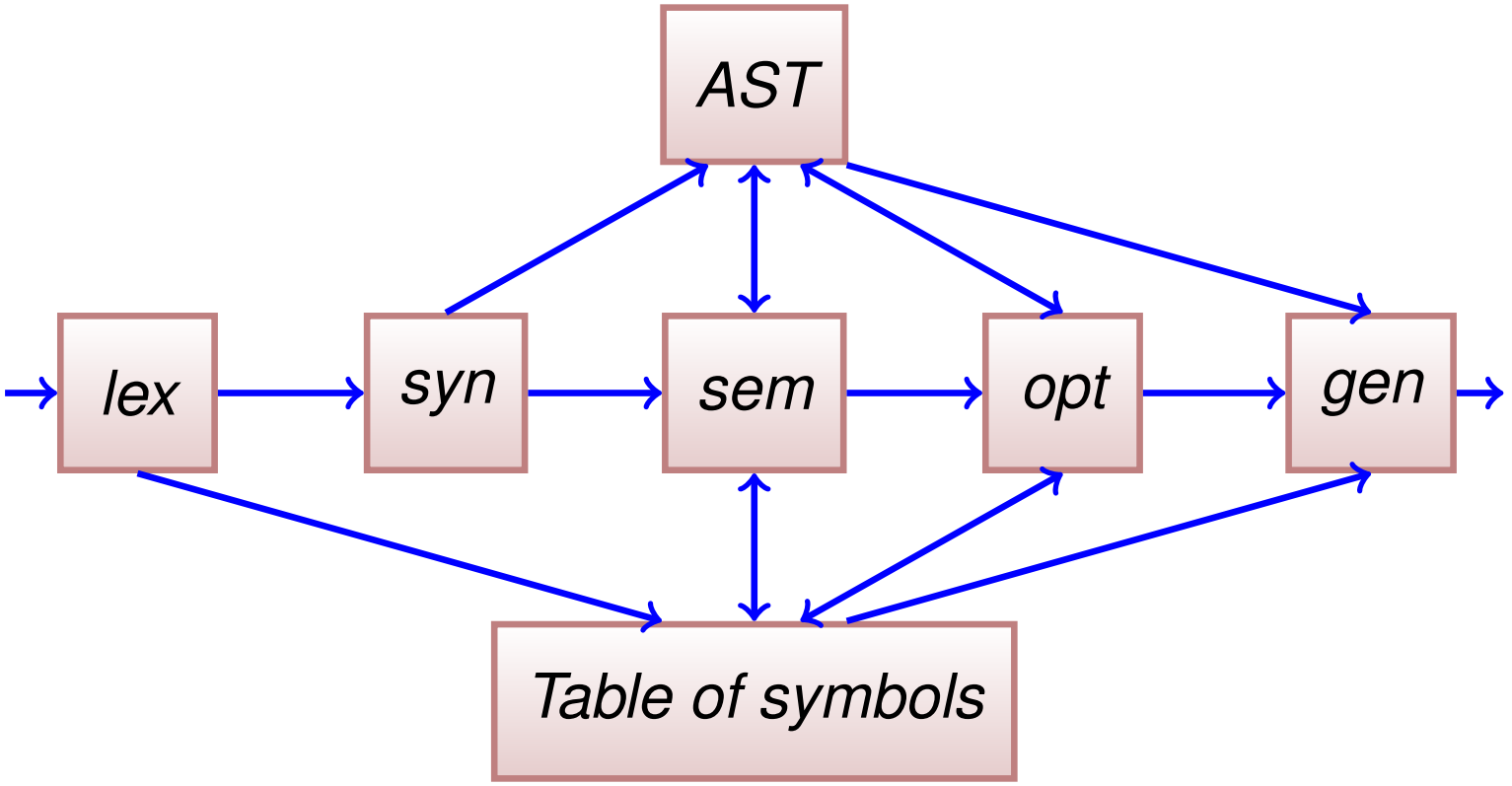
A sequence of functions. . .



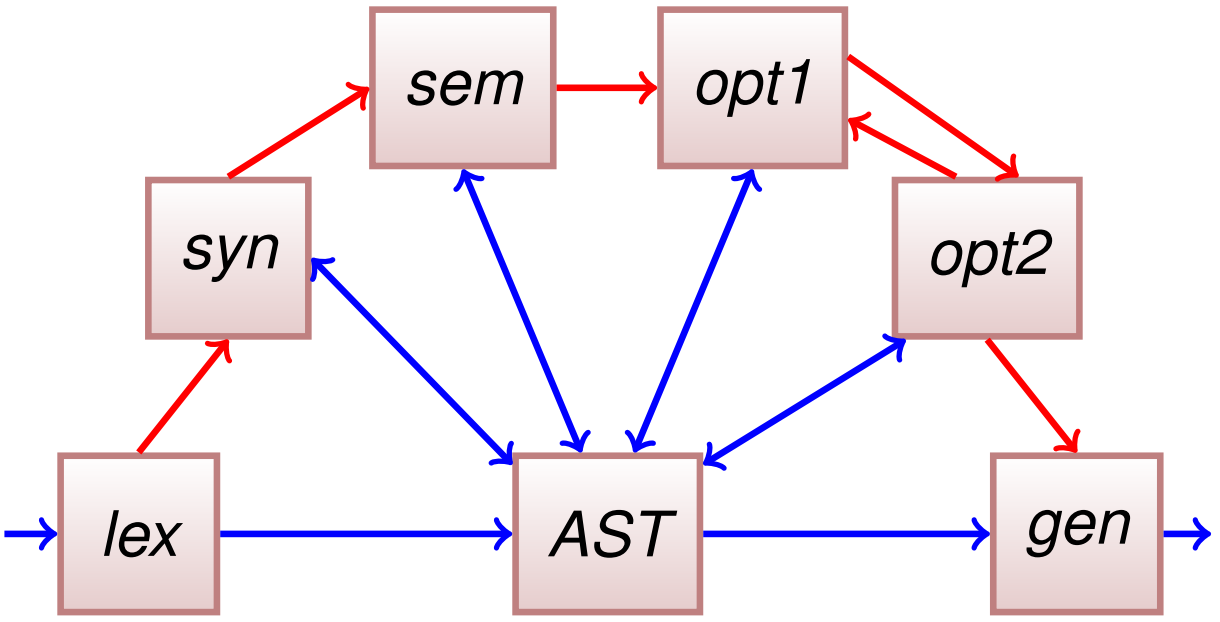
A sequence of functions sharing a table of symbols . . .



A sequence of functions sharing a table of symbols and an Abstract Syntax Tree (AST)...



A sequence of functions sharing informations and triggering optional computations. . .



You can already tell a lot from the topology : sharing, bottleneck, point of failure, etc.

Several styles

- batch
- blackboard

Different properties : efficiency, evolutivity, . . .

Architecture

An architecture is composed of :

- components (boxes)
- connectors (lines)

to build a configuration.

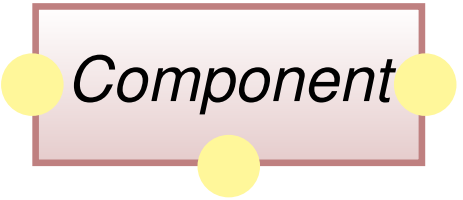
Architecture = structure with entities and relationships

No standard definition

- `http://www.sei.cmu.edu/architecture/definitions.html` or
- `http://en.wikipedia.org/wiki/Software_architecture`

No standard definition

- a unit (the box) bounded by
- a boundary (the border) which has
- access points (ports)



What a component encapsulates ?
Various “units” . . .

- a function,a procedure, a computation
- an objet
- a service
- a storing unit
- an interface
- . . .

A good practice is to associate a responsibility

Fewer definitions . . .

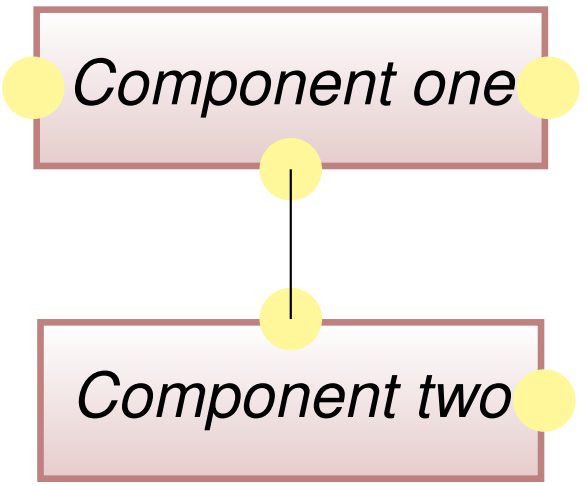
- a connection that links
- two (or more) components which
- play a role (in the connection)

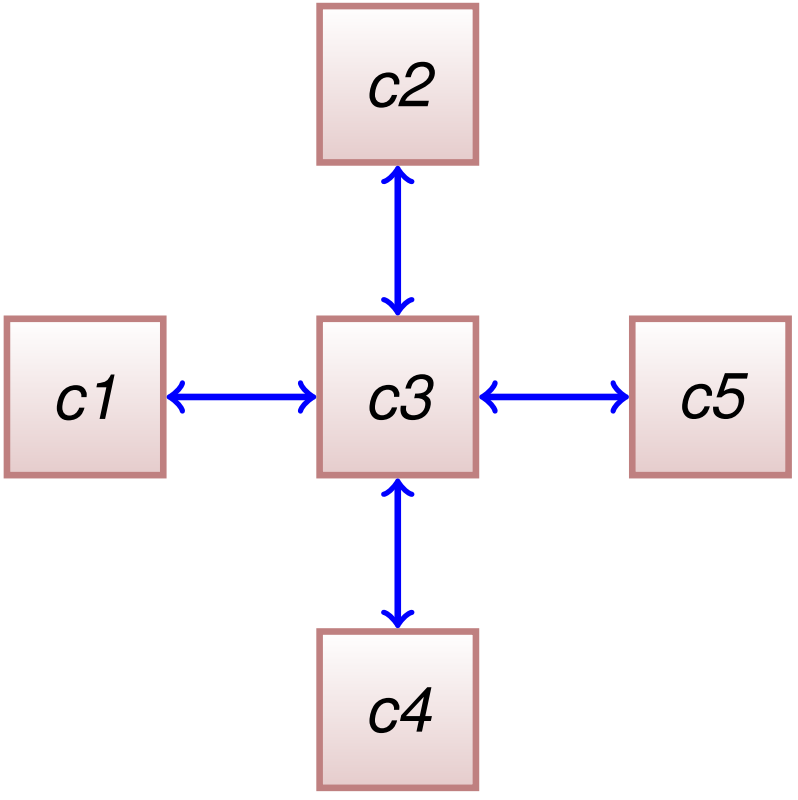
What a connecteur represents ?
Various “connection means” . . .

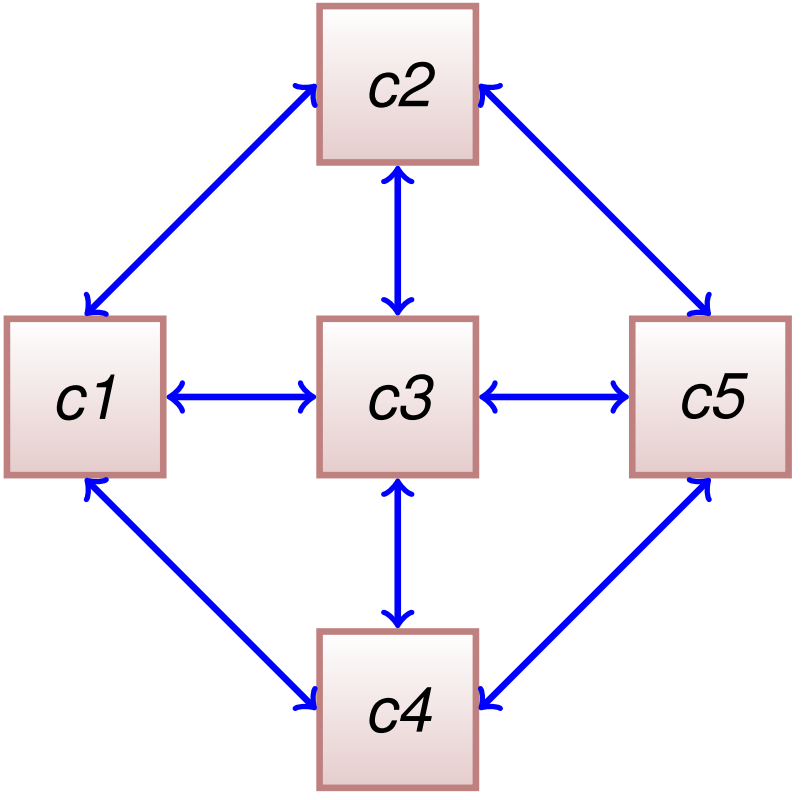
- a bus
- a protocol
- a (remote) procedure call
- . . .

A good practice is to associate a transfert of data or control

A set of assembled components and connectors.
Ports are bound to ports. (What about compatibility ?)

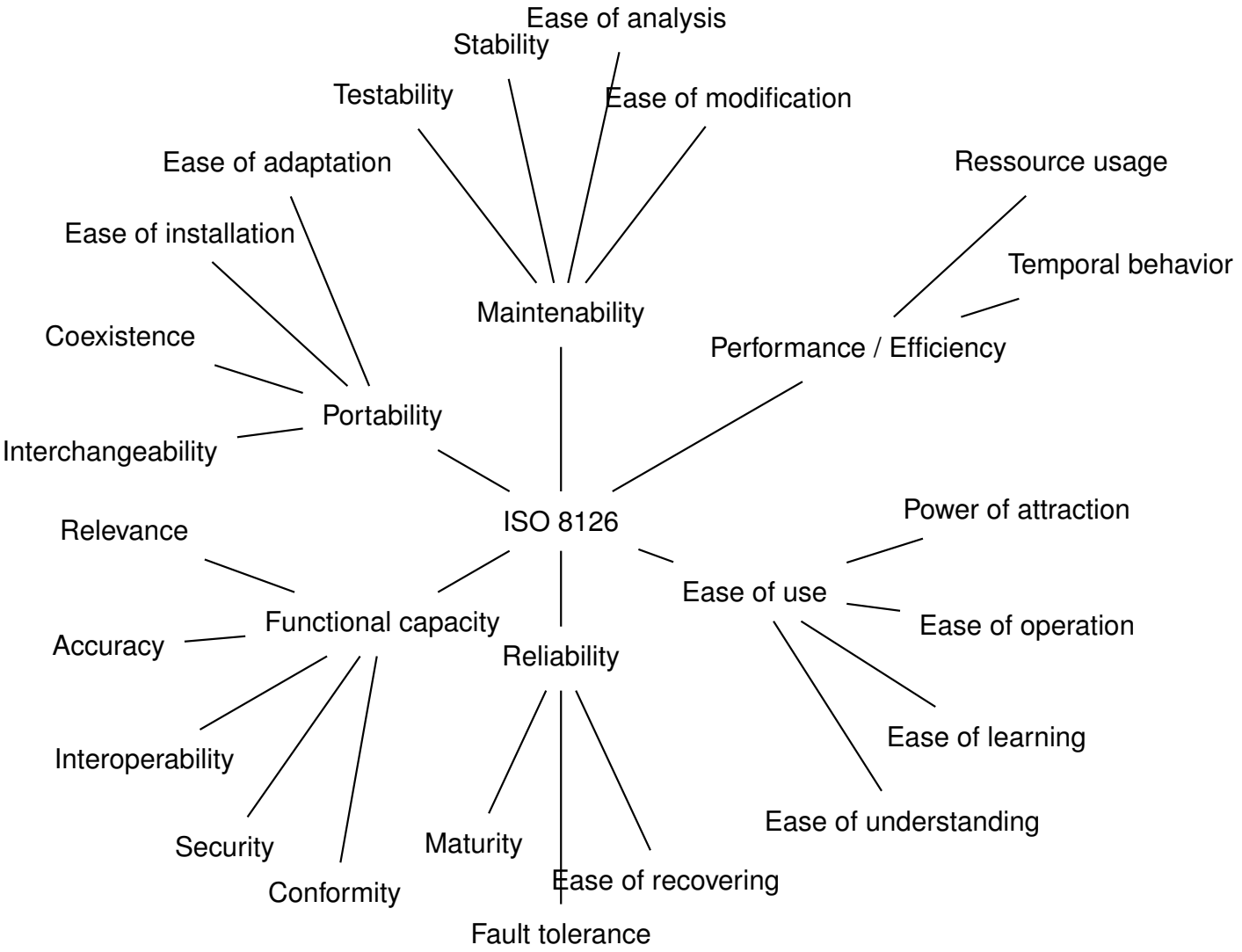



c2
c1
c3
c5
c4


c2
c1
c3
c5
c4

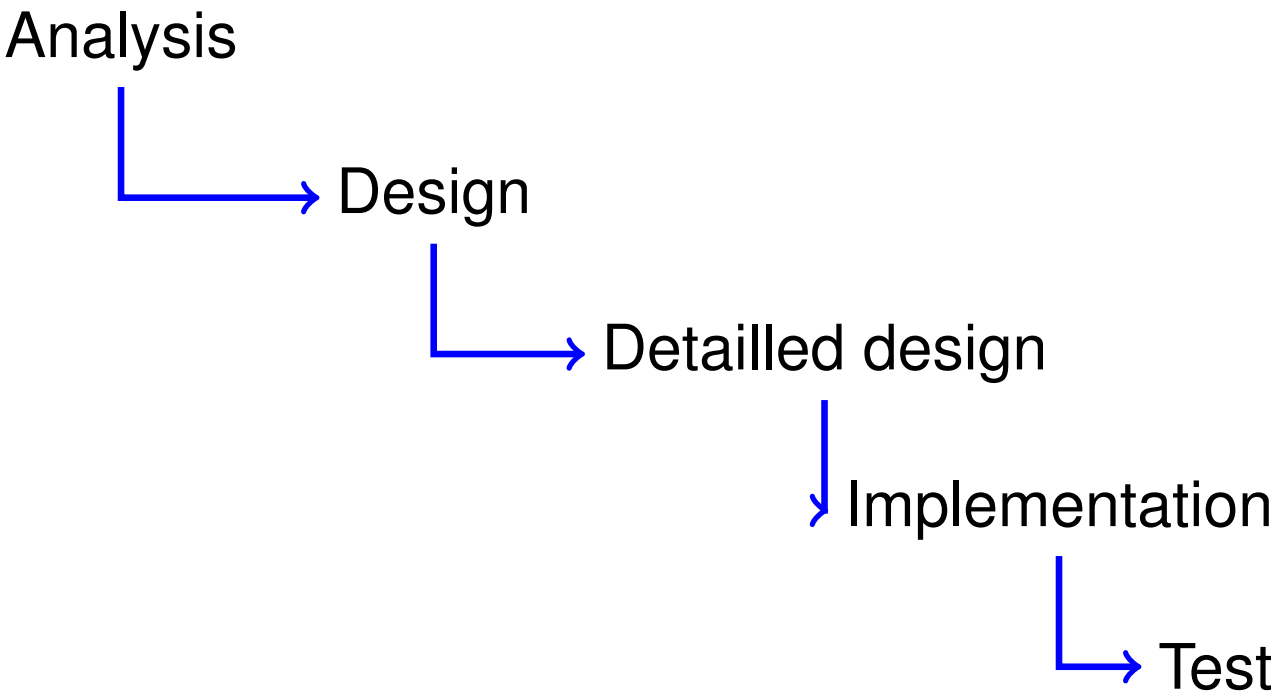
- Coupling (goal : low coupling)
- Cohérence (goal : strong consistency)
- Robustness
- ...

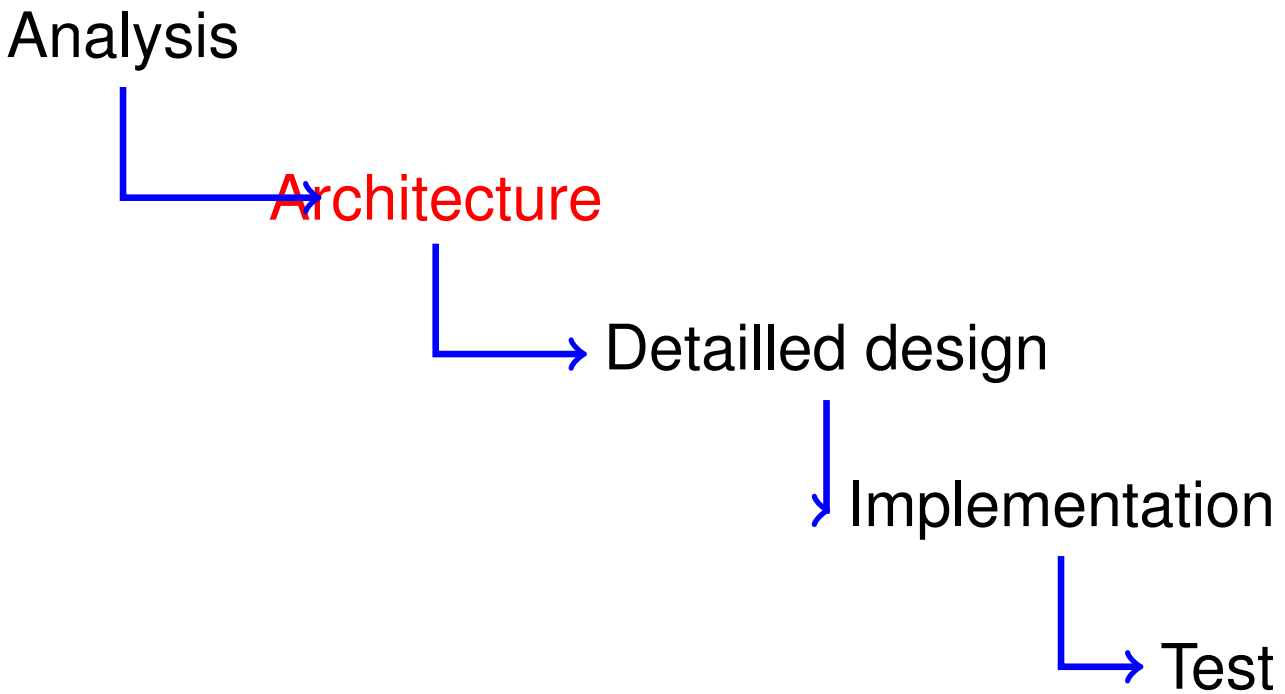
Non functional properties ISO 9126



- ACME
- Aesop
- Darwin
- MetaH
- Rapide
- Wrigth
- AADL

Very good overview in [MT00].


Analysis
Design
Detailled design
Implementation
Test


Analysis
Architecture
Detailled design
Implementation
Test

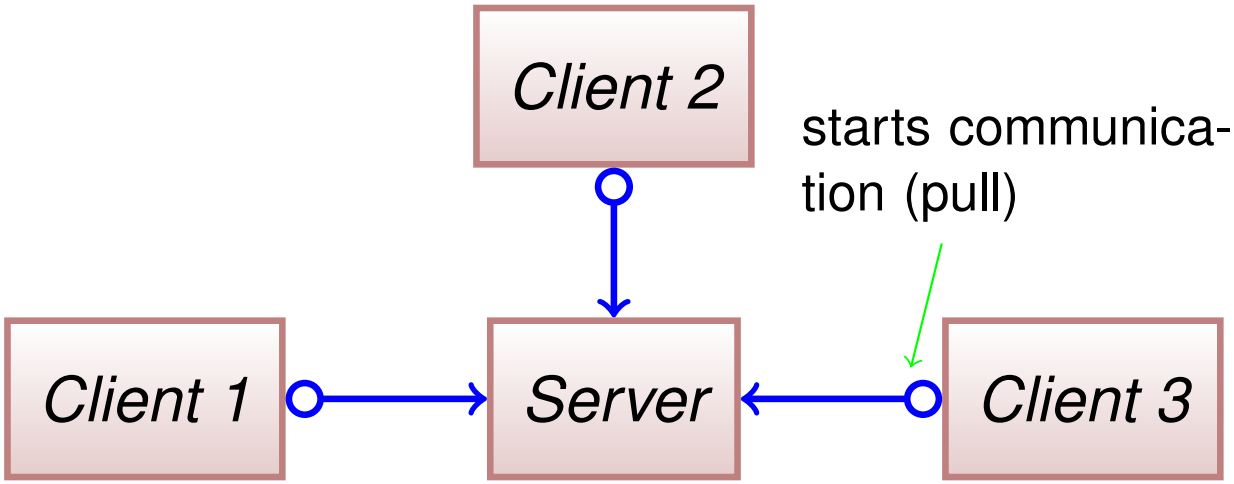
Architecture styles

Set of components types and of connectors types.

Mandatory assembly rules.

Set of components types and of connectors types. ex. Client, Server

Mandatory assembly rules.



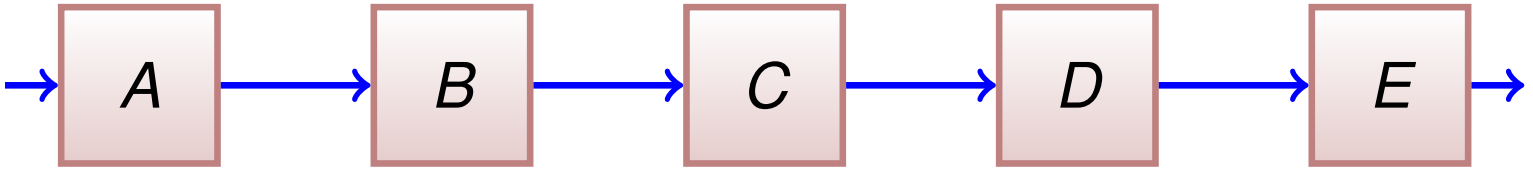
- Dataflow
- Pipe & Filter
- Blackboard
- Publish-subscribe
- Layers
- Client-Server
- Peer-to-peer
- N-tiers

- Capitalising on know-how and knowledge
- Define vocabulary
- Improve communication skills (between people)
- Associate properties (advantage/disadvantage) with styles

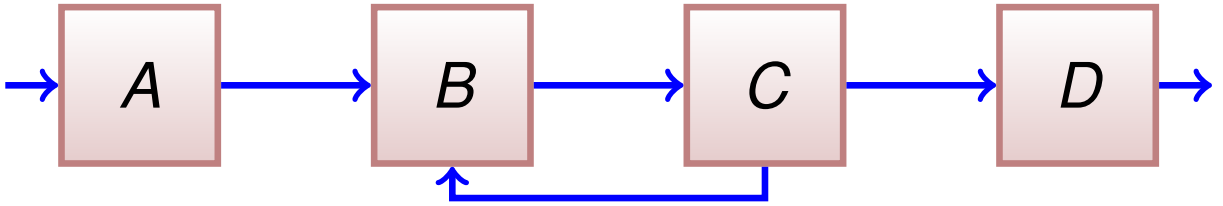
Do you know of any other comparable situations ?

- Capitalising on know-how and knowledge
- Define vocabulary
- Improve communication skills (between people)
- Associate properties (advantage/disadvantage) with styles

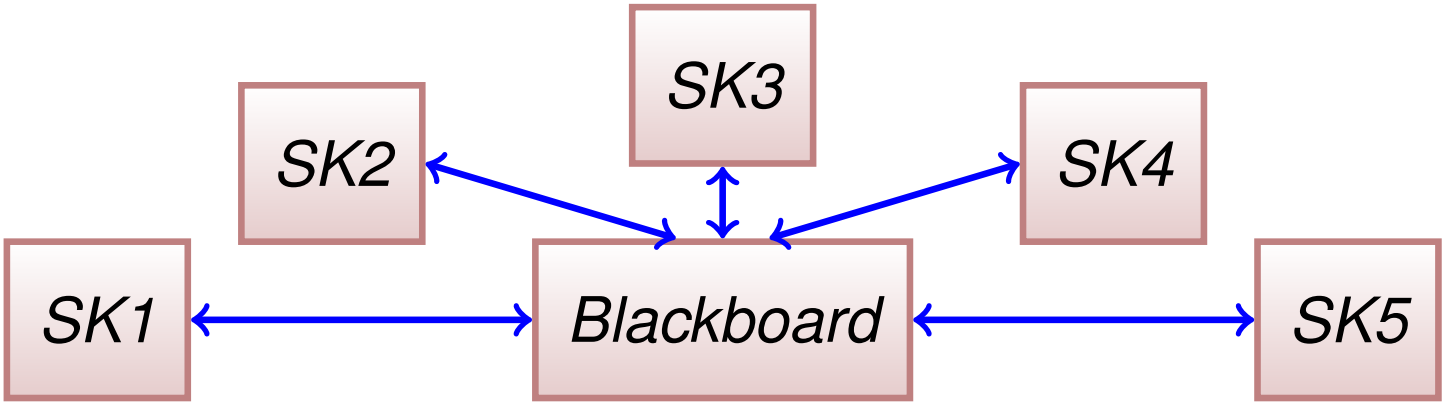
Do you know of any other comparable situations ?
Patterns (design, organisation, . . .)



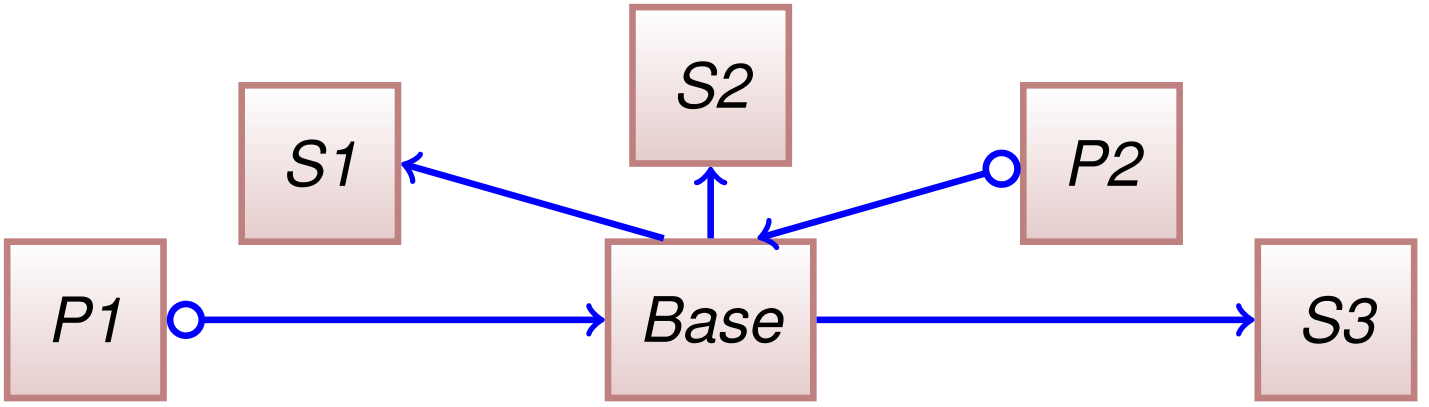
Components	*Connectors*
Filter	Mono-directional data transfer
Advantages	*Disadvantages*
Simple composition Weak coupling Interchangeable boxes	1 input, 1 output only Easier with homogeneous data



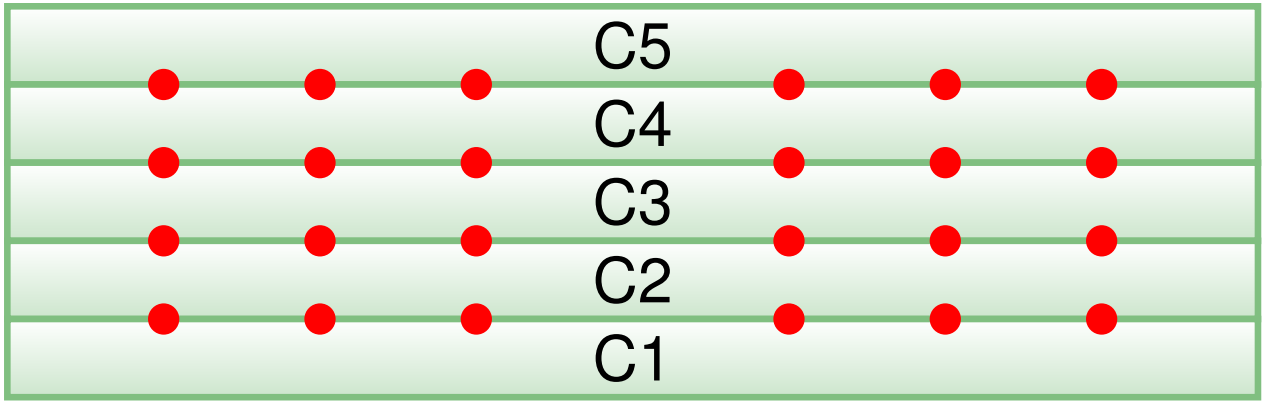
Components	*Connectors*
Filter	Pipe (with buffer)
Advantages	*Disadvantages*
Simple composition Weak coupling Interchangeable boxes	Risk of deadlock Easier with homogeneous data finite size buffer



Components	*Connectors*
Blackboard Source of knowledge	bi-directionnel protocol
Advantages	*Disadvantages*
Fast error correction Little information loss Extensible Weak coupling Simple composition	Risk of unnecessary data accumulation Vocabulary constraints Domain-dependent application

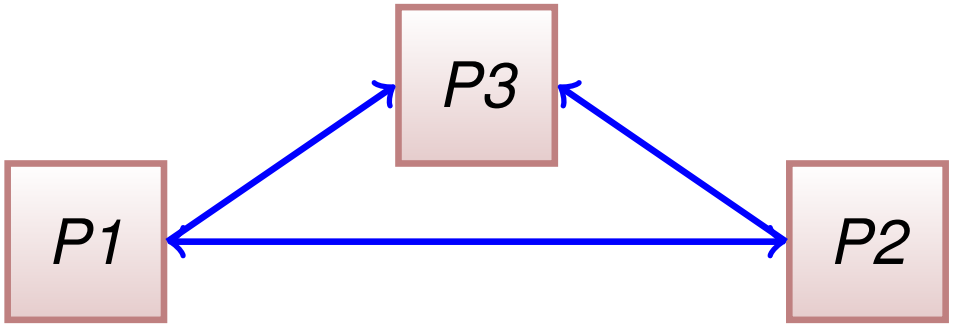


Components	*Connectors*
Base Publisher Subscriber	dedicated protocol message
Advantages	*Disadvantages*
Anonymity Low coupling Expandable	Performance Asynchronous (?)



Components	*Connectors*
Layer	Procedure call
Advantages	*Disadvantages*
Incremental design Maintenance Reuse	Level size Performance

Components	*Connectors*
Server Client	protocol *pull*
Advantages	*Disadvantages*
Simple composition Weak coupling Extensible Performance	Point to point only



Components	*Connectors*
Peer	dedicated protocol
Advantages	*Disadvantages*
Symmetry Fault tolerance Extensible	Performance

“client-serveur” or “Layers” variant

A tier per responsibility.

Example :

1. Data tier (store)
2. Application tier (business)
3. Presentation tier (HMI)

- Multitier and layered styles are similar ; the nature of the connection is different : more flexible in multitier.
- The blackboard, publish subscribe styles are similar ; the nature of the roles of the components that use the shared area is more or less specialised.
- The 2-tier styles, client server, blackboard, or publish-subscribe look alike

A complex system uses several styles :

Horizontal At a given level, a part of the system may be of one style and another part of another style.

Vertical When we study the internal structure of a component of a style, one can find another style there.

Interaction and architecture

An architecture defines an interaction topologie.

It shows (or can show) :

- topology
- coupling
- location of interfaces
- direction of exchanges
- initiative of exchanges

It (usually) hides :

- details of interface
- order of exchanges (temporal aspects)

References I

P Eugster, P Felber, R Guerraoui, and A Kermarrec.

The Many Faces of Publish/Subscribe.

ACM Computing Surveys (CSUR), 2003.

David Garlan and M Shaw.

An Introduction to Software Architecture.

In *Advances in Software Engineering and Knowledge Engineering*, pages 1–40. 1993.

Nenad Medvidovic and Richard N Taylor.

A classification and comparison framework for software architecture description languages.

IEEE Transactions on Software Engineering, 2000.