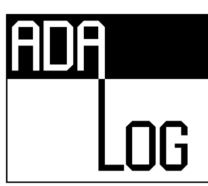


Hierarchical ***O***bject ***O***riented ***D***esign

Initially designed on behalf of ESA (*European Space Agency*)

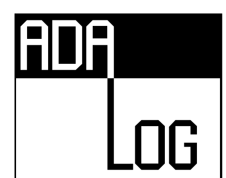
By a European consortium:

CRI(Dk) + CISI Ingénierie(F) + Matra(F)

Goal:

> To make OOD fit big software projects, involving many subcontractants

© ADALOG J-P. Rosen



HOOD's History

1986	Call for bids from European Space Agency for a design method targeting Ada
	CRI - CISI Ingénierie - Matra consortium is chosen
1987-1988	Pilot projects.
	Revision 2.2. HOOD chosen for COLUMBUS.
	Creation of the Hood Users Group (HUG) and of the Hood Working Group (HWG)
1989	Revision 3.0. HOOD chosen for HERMES.
1992	Revision 3.1
	The HUG agrees to freeze this version for at least two years.
1993	HRT-HOOD
1995	Version 4 is issued
1997	Publication of the reference book: "HOOD: an Industrial approach for software design".

© ADALOG J-P. Rosen

What is HOOD?

ADA LOG

A method

- Merging of notions from:
  - Object Oriented Design
  - Abstract States Machines
- Initially oriented towards Ada (C++, FORTRAN and other languages now supported)

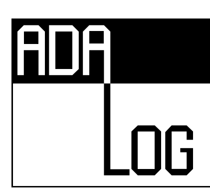
A formalism

- Uniformed representations
  - Textual representation
  - Graphical representation
- Rules

Supporting tools

A standard

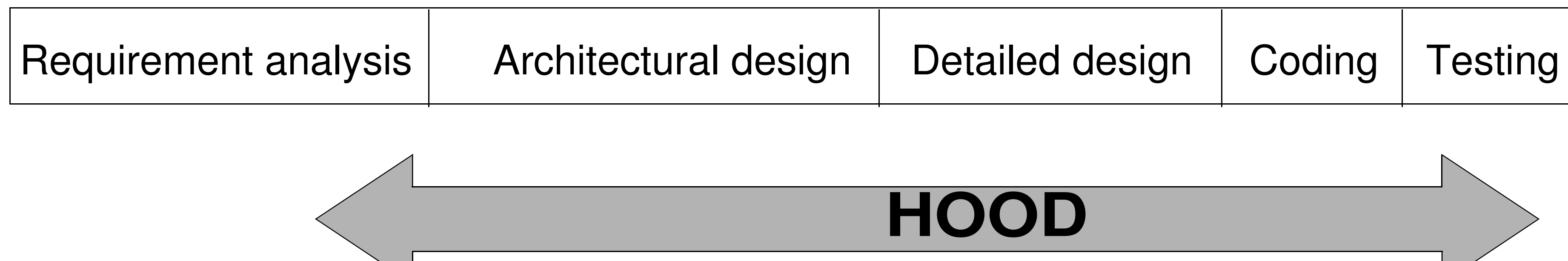
© ADALOG J-P. Rosen



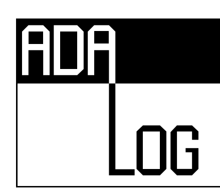
HOOD in the life-cycle

HOOD is a detailed design method

- ☞ Starts after analysis
- ☞ Extends down to coding and testing



© ADALOG J-P. Rosen



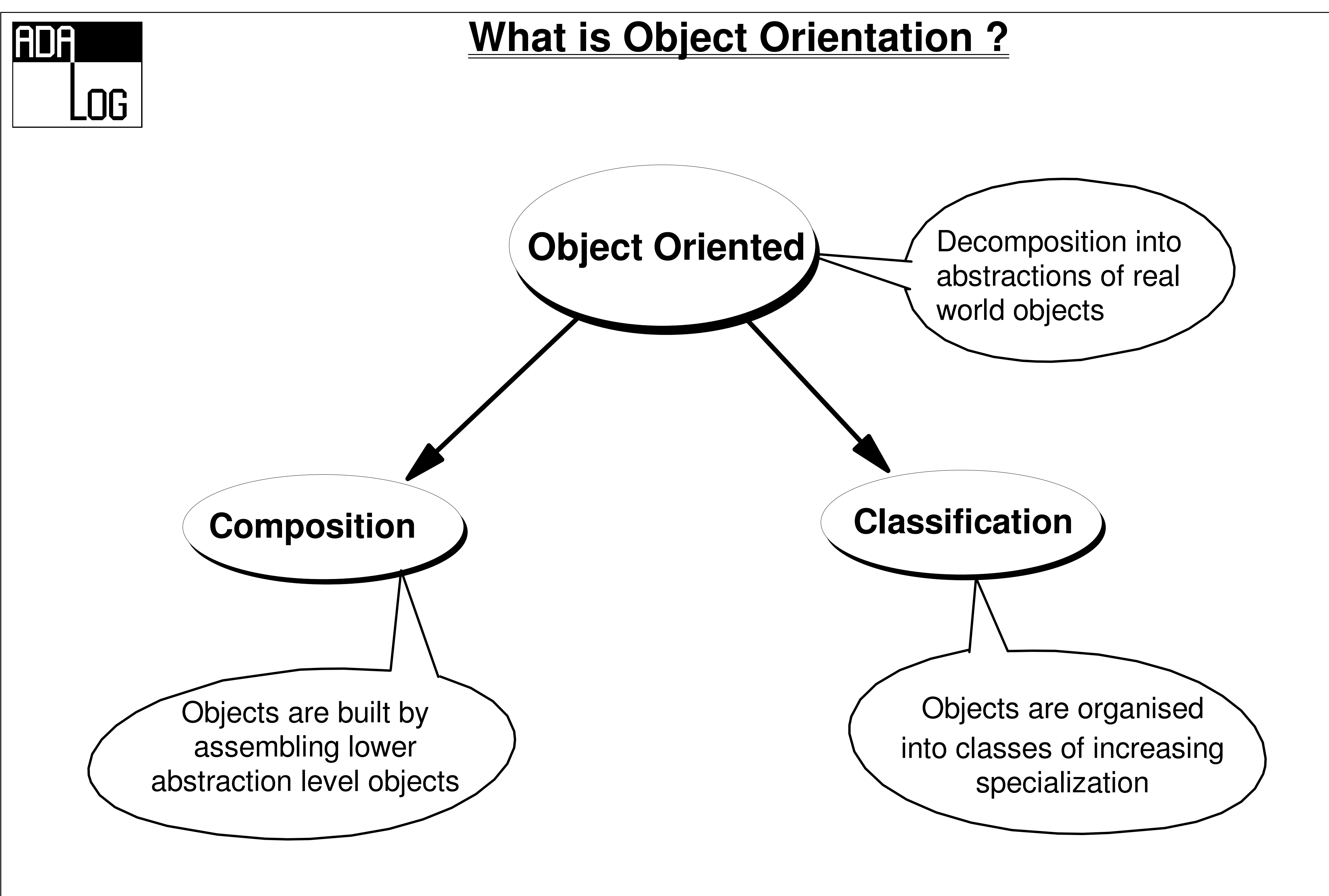
Object Oriented Paradigms

Unit of decomposition is no more the *action*, but the *object*.

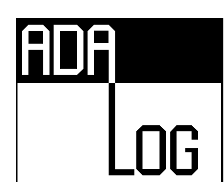
An object is an *abstraction* of a *real world* object.

A program is no more a *sequence of statements* to be executed by a machine, but a *model* of the real world.

© ADALOG J-P. Rosen



© ADALOG J-P. Rosen



HOOD's goals and principles

Isolating and splitting

Modules with high coherence and low coupling.

- ☞ Limit visibility.
- ☞ Enforce abstraction (external view independent from implementation).

Split analyses

- ☞ structural analysis (organization of project, ☞ *what*)
- ☞ functional analysis (description of actions, ☞ *how*)
- ☞ behavioral analysis (management of events, ☞ *when*)

Mastering interfaces

Who uses what at which time.

- ☞ Provided interface
- ☞ Required interface

Provide a framework for formal verification methods.

ADA LOG

HOOD diagrams

Represent the result of a design

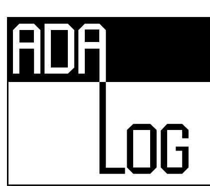
- ❏ A method is required

Do *not* hold all relevant information

- ❏ Another kind of documentation is required to state:
  - ☞ Constraints
  - ☞ Details of calls
  - ☞ Meaning of data flows
  - ☞ Conditions for the raising of exceptions
  - ☞ ...

Designing is not only drawing arrows and boxes!

© ADALOG J-P. Rosen



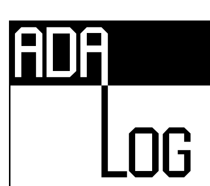
ODS

Object Description Skeleton

The ODS is the formalized description of an object.

- Every object in the system is described by an ODS.
- All informal descriptions start with --| and end with |--

© ADALOG J-P. Rosen



SIF

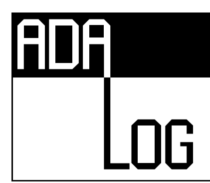
Standard Interchange Format

- *ASCII* text, following the BNF notation of an ODS.

- Allows *exchanging designs* between tools from various origin.

- Every tool must allow (at least) exchanging:
  - ☞ A complete design
  - ☞ An object
  - ☞ An object and all its descendents.

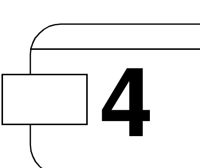
© ADALOG J-P. Rosen



HOOD objects

HOOD objects are *modules*

Can be, although not necessarily, "objects" in the sense of object oriented methods.



Characterized by:

- ☞ An external interface (OOD): PROVIDED_INTERFACE
- ☞ A behavioral description (ASM): OBCS
- ☞ A functional description (SP): OPCS

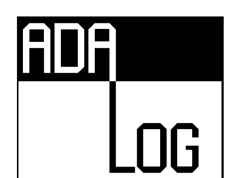
Implemented by:

- ☞ Other child objects (non terminal objects) : IMPLEMENTED_BY
- ☞ Code (terminal objects) : CODE

Using:

- ☞ Other external elements : REQUIRED_INTERFACE
- ☞ Internal data : INTERNALS

© ADALOG J-P. Rosen



Representation of a HOOD object

Special kind

Name

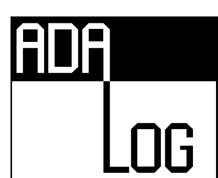
Provided interface

E

Internal structure

Required interface

© ADALOG J-P. Rosen



Notion of implementation delegation

Volume

Channels

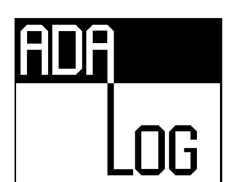
On/Off

Audio amplifier

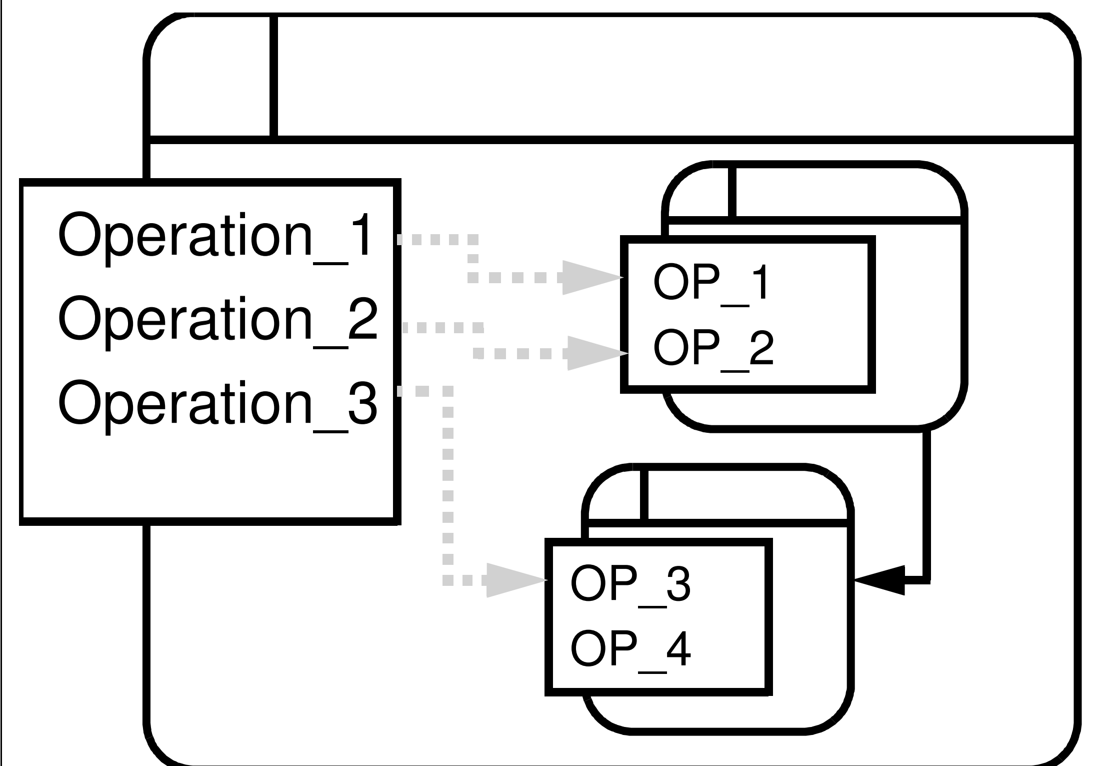
Tuner

Power supply

© ADALOG J-P. Rosen



Hierarchical structure: the INCLUDE relationship

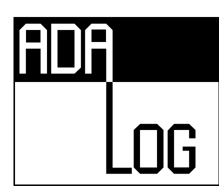


An object is *composed* of other child objects

The parent *implements* its operations by its children

The hierarchical relationship expresses the *static* structure of the system

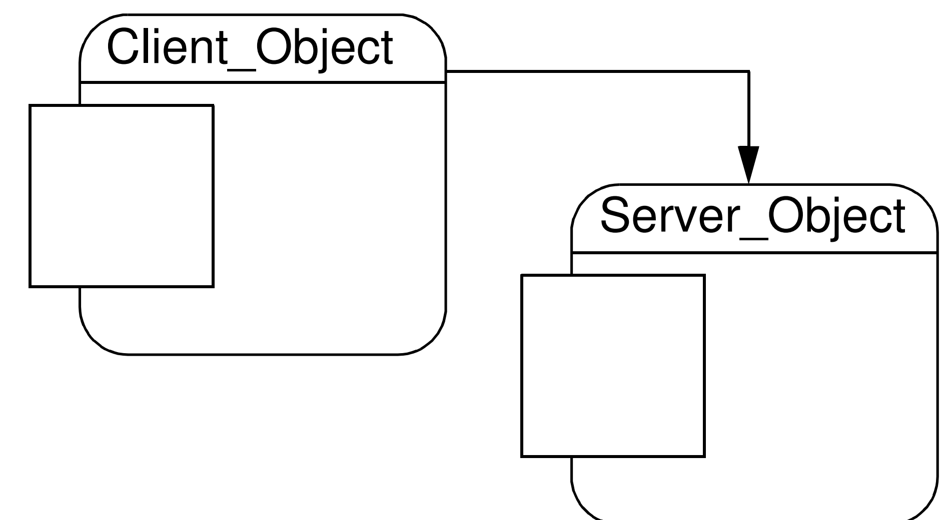
© ADALOG J-P. Rosen



Composition: the USE relationship

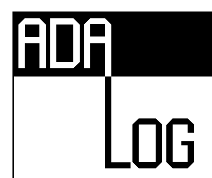
An object uses services provided by other objects

- ☞ The object that provides the service is a *server*.
- ☞ The object that uses the service is a *client.*



The "use" relationship expresses the *dynamic* structure of the programme

© ADALOG J-P. Rosen

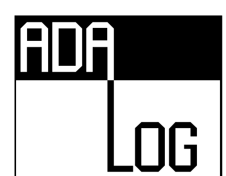


Uncles and nephews

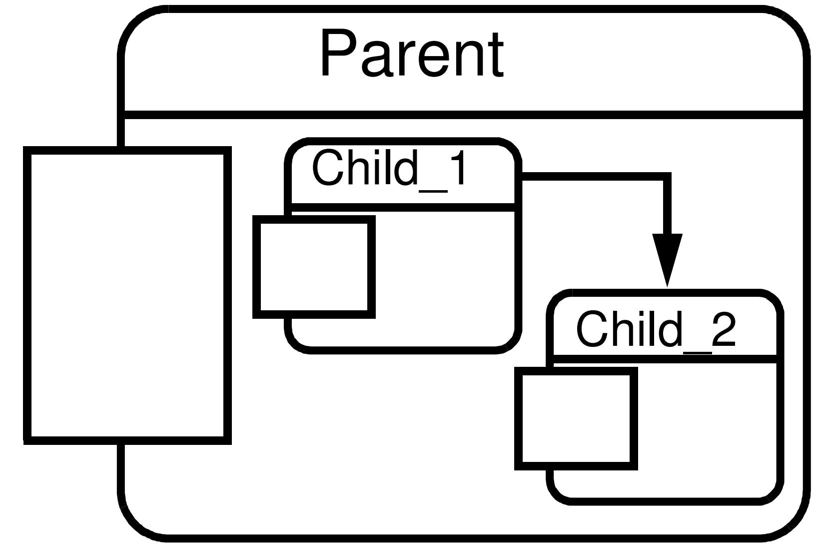
If Child_1 uses Child_2, then one of its own sub-objects must use it itself.
Sub_Object_1 uses one of its father's brothers.

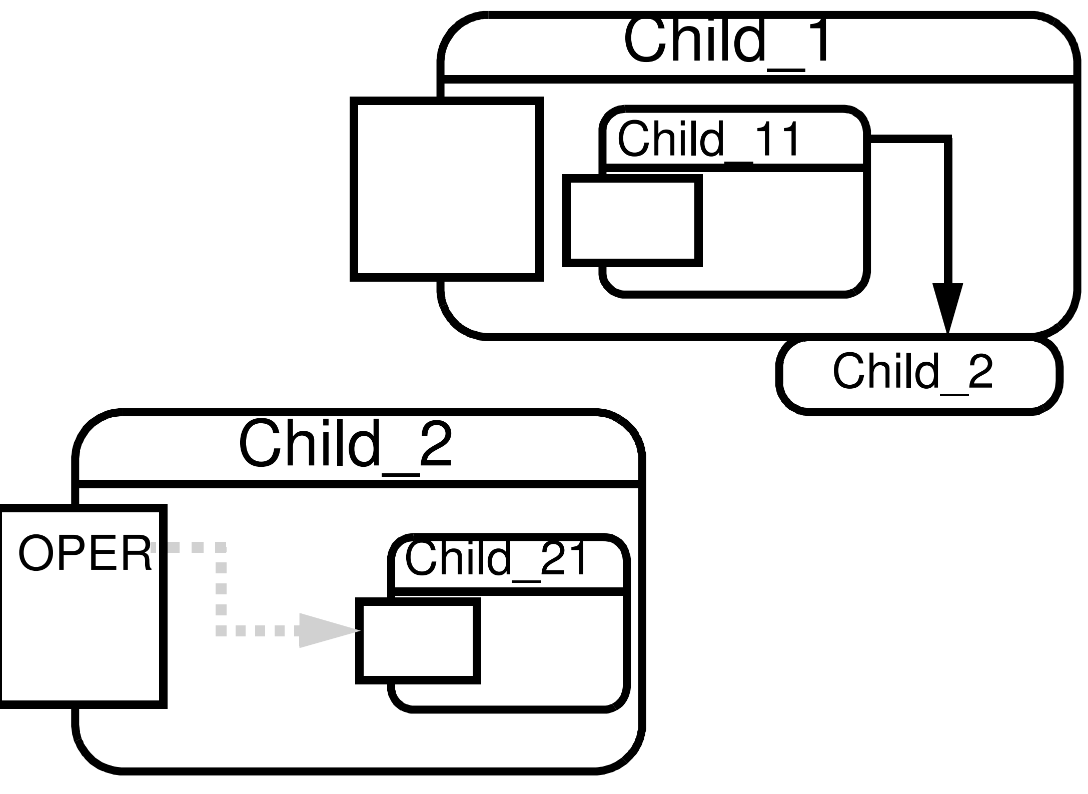
In the diagram for Child_1, Child_2 appears as an *uncle*.

© ADALOG J-P. Rosen



Static structure, dynamic structure

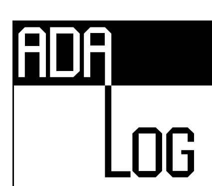




- Structurally, opacity of abstractions is preserved.
- At execution time, Child_11 calls Child_21 directly

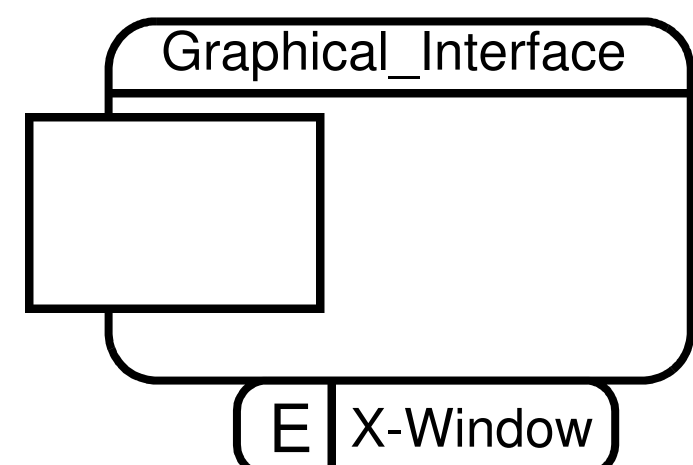
The layered structuring does *not* imply additional procedure calls.

© ADALOG J-P. Rosen



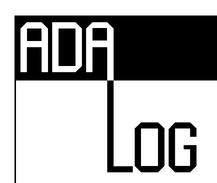
Environment objects

An "escape" to too strict a hierarchy.



- ❑ Are *uncles of everybody*, that may appear at any level in the hierarchy
- ❑ Once introduced, dependences follow normal HOOD rules.
- ❑ Correspond to:
  - ☞ Off-the-shelf components (services, libraries, interfaces...);
  - ☞ Objects developed by other programming teams;
  - ☞ roots from other design trees.

© ADALOG J-P. Rosen



Terminal object, non terminal object

Non terminal object

- Is decomposed into several child objects.
- All properties (operations and types) *must* be subcontracted.
- Purely conceptual (*empty shell*).

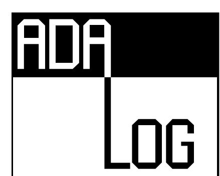
Terminal object

- Is implemented by programming language code.

An object is either terminal or non terminal

☞ If a non terminal object requires code, it must subcontract it to a terminal subobject (OP_CONTROL).

© ADALOG J-P. Rosen

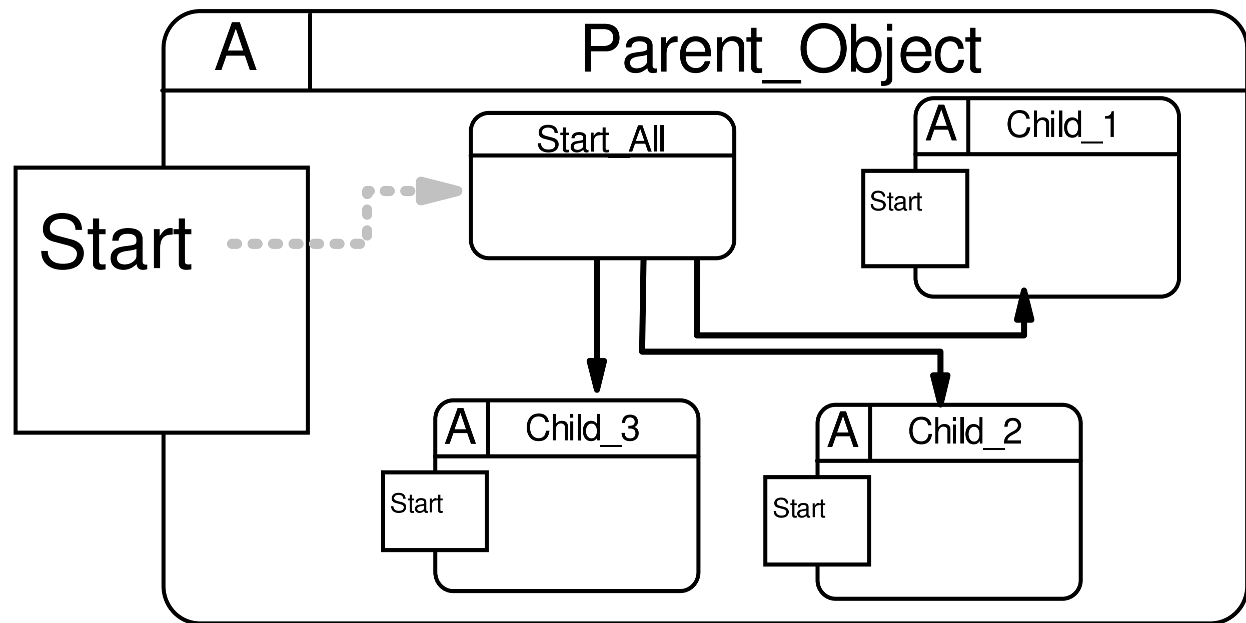


OP_CONTROL

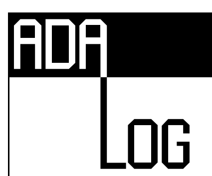
Object reduced to a single operation

Nom_d'Opération

- Allows implementation of an *unconstrained* operation of a parent by several operations provided by brothers.
- Always *terminal*.
- Resorts to structured programming analysis.



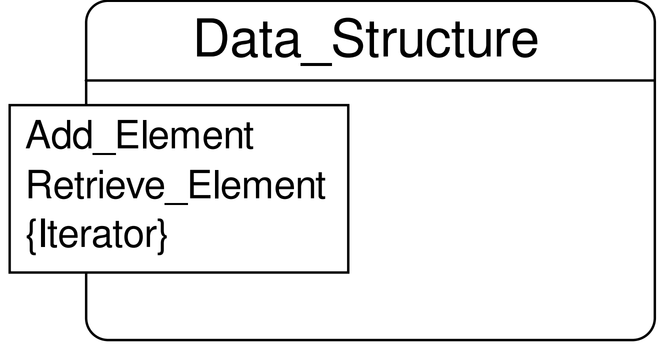
© ADALOG J-P. Rosen



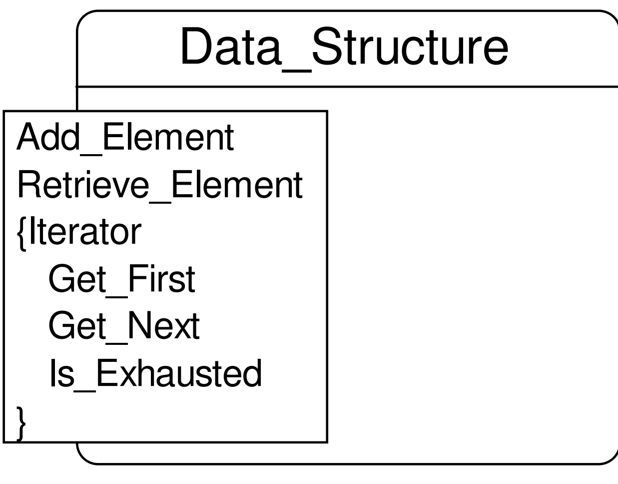
Operation Set

A set of logically related operations that can be considered as a whole

- ❑ Simplification of the description
  - ☞ Can include operations and subsets
  - ☞ The textual description declares the set
  - ☞ Each member operation contains a MEMBER_OF clause

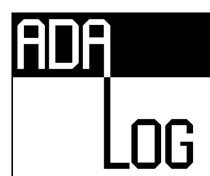


Closed set



Open set

© ADALOG J-P. Rosen



Data analysis

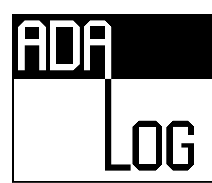
Objects exchange data.

- Data are typed.
  - Operations analysis leads to data analysis.
- Three kinds of data:
  - Simple types
  - Abstract data types (HADT = *HOOD Abstract Data Type*)
  - Classes. 4

A HOOD object can be viewed under two aspects

- Functional (client-server view)
- Data (structural view) 4

© ADALOG J-P. Rosen



Base Types

Types from the target language

Precise semantics *depends on the language.*

- Predefined types
  - ☞ Do not belong to the problem domain.

    INTEGER, int, ...

- User defined types
  - ☞ Have *at least a name* that identifies what they represent.

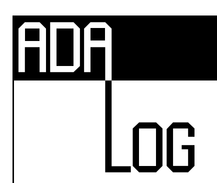
In Ada

```
type Length is range 0..10_000;
```

In C

```
typedef int Length;
```

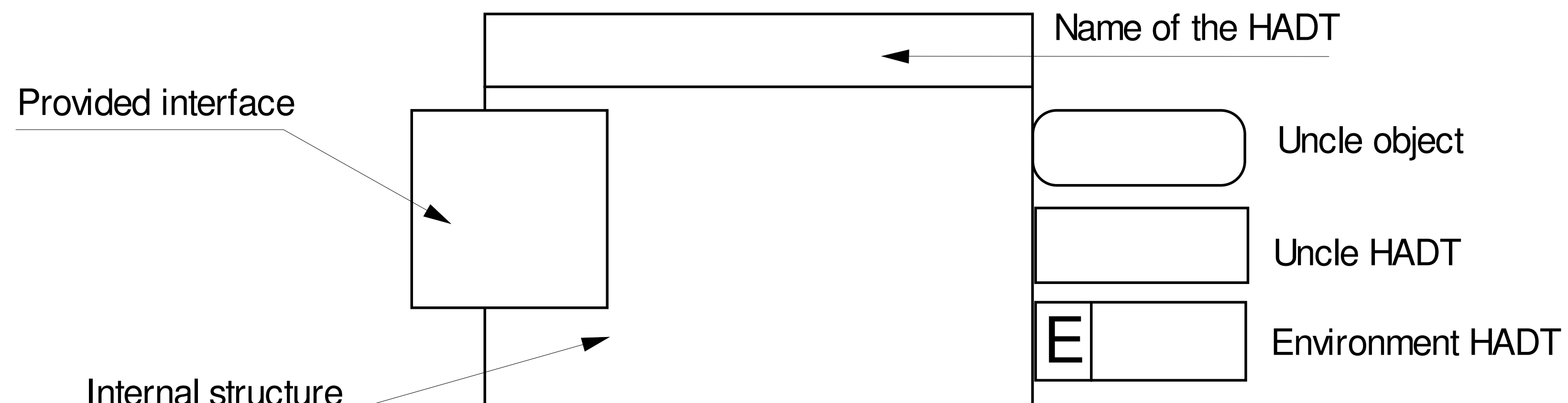
© ADALOG J-P. Rosen

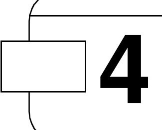


Abstract data types

HADT (HOOD Abstract Data Type)

A HOOD object that represents *one* data type together with associated *operations*.

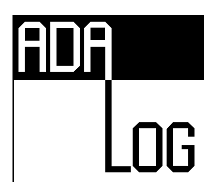




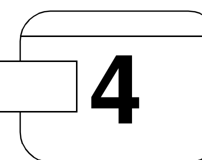
- ❑ Can be decomposed
- ❑ Can aggregate other objects

Aggregation arrow

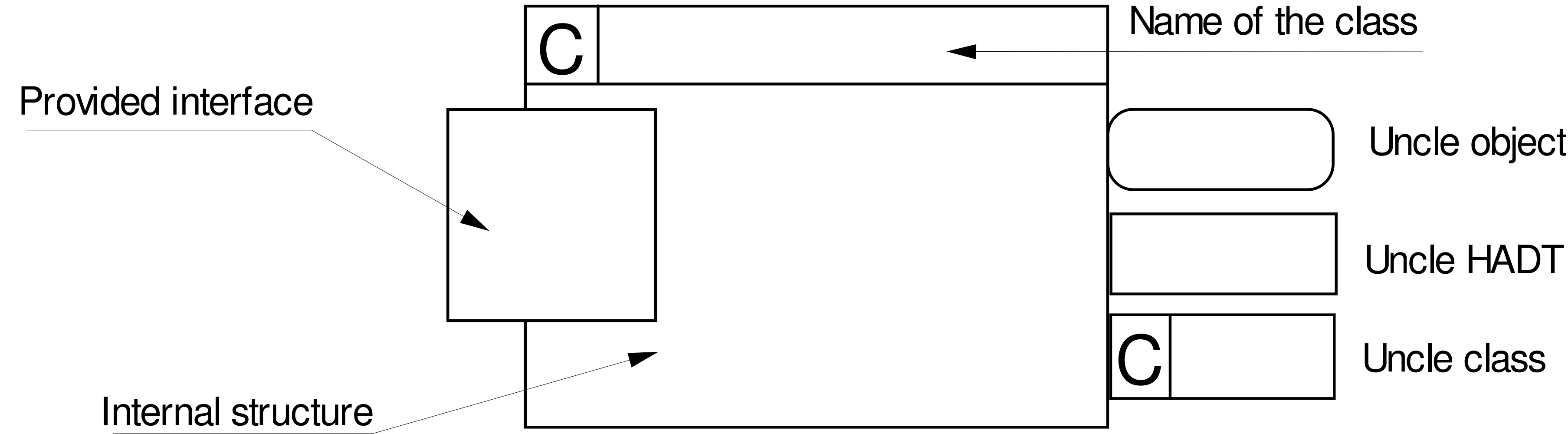
© ADALOG J-P. Rosen



Classes



Are actually *normal* HADTs for which *inheritance* is allowed

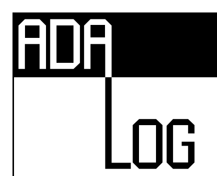


- Can aggregate other objects
- Can inherit from other objects

Inheritance arrow

- Are always *terminal* (cannot be decomposed)

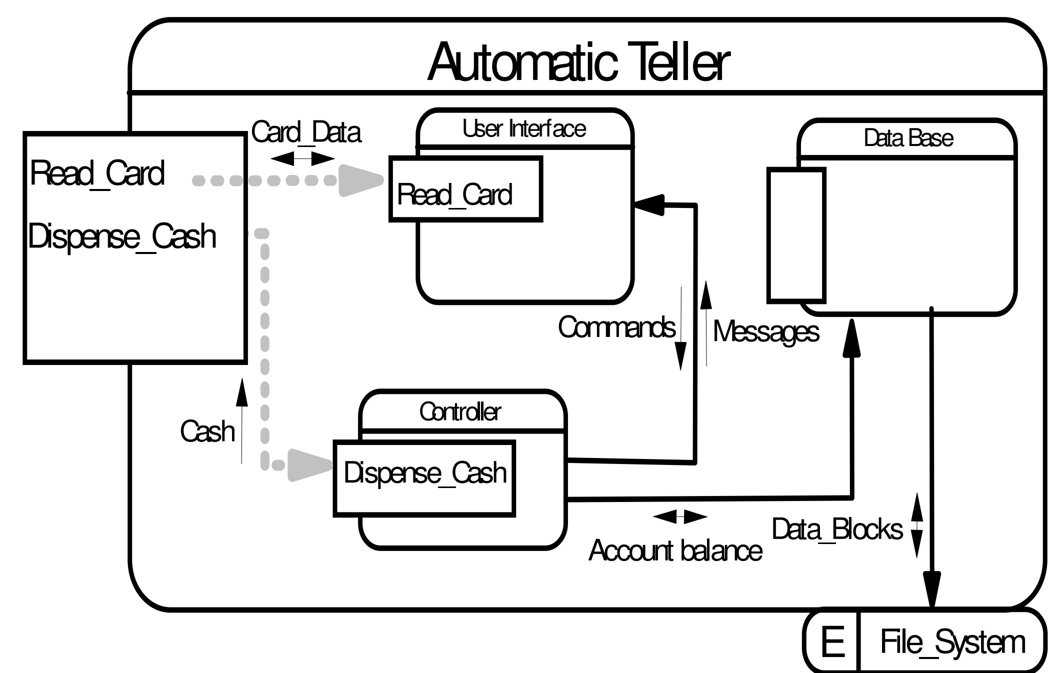
© ADALOG J-P. Rosen



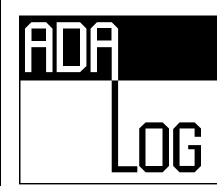
Data flows

Data flows that are *important* for understanding the system are documented.

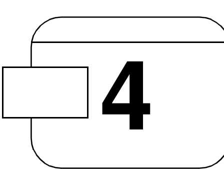
- ❑ The direction of the data flow is not related to the calling direction.
- ❑ Only data flows that are important for understanding the system are represented
- ❑ The precise meaning of data is documented in the DATAFLOWS section of the ODS.



© ADALOG J-P. Rosen



The structural view



"USE" relationship

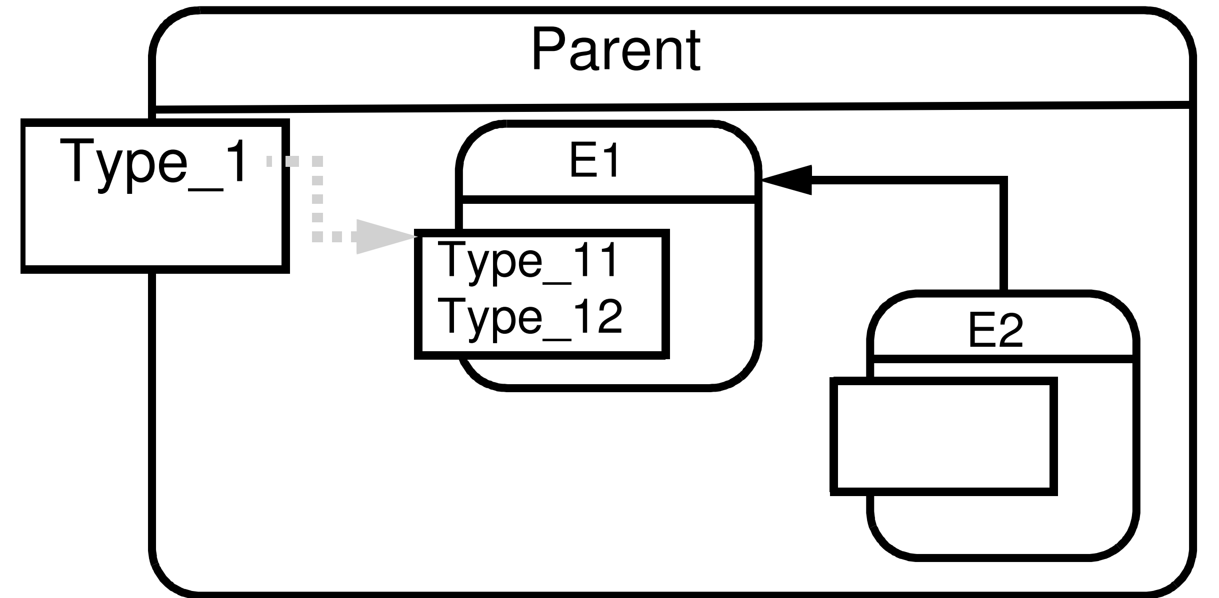
In the structural view, "USE" relationships are "TYPE-USE" relationships.

☞ Means that the client *declares objects* whose type is provided by the server

"IMPLEMENTED_BY" relationship

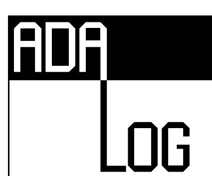
Always between a provided type of the parent and a provided type of the child.

☞ Means that a type exported by a parent is actually a type *defined in a child*.



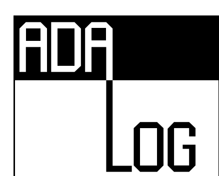
- Type_1 of Parent is implemented by Type_11 of E1.
- E2 declares objects of a type exported by E1.

© ADALOG J-P. Rosen



Example of a structural view

© ADALOG J-P. Rosen

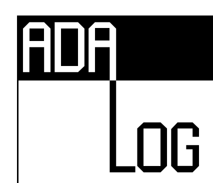


Notion of constraint

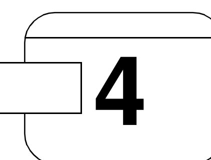
What makes that an operation cannot be called at any time.

- State constraints 4
  - The object accepts the operation only when its internal state allows it.
- Concurrency constraints 4
  - The object accepts the operation only if other objects are not using it at the same time.
- Protocol constraints
  - The dialogue between objects must obey certain temporal properties.

© ADALOG J-P. Rosen



State constraints

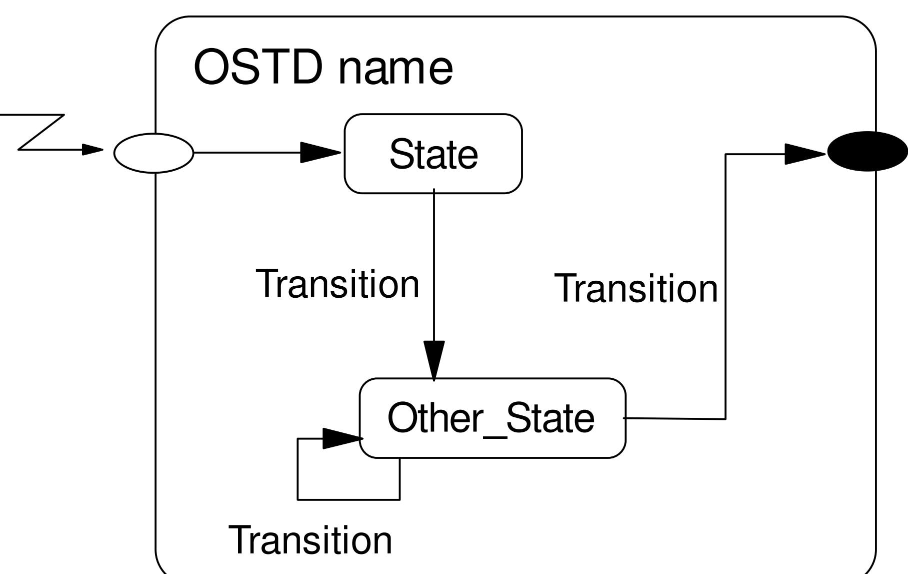


The object has a hidden state, and provides some operations only when in certain states

Trying to call an operation when the object is in a state that does not allow it is a programming error that triggers an exception.

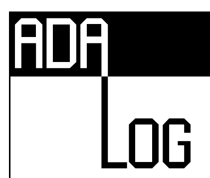
OSTD (Object State Transition Diagram)

- ☞ A way of representing *states* that allow calls to certain operations
- ☞ The operations are *constrained* by the states
- ☞ *Only operations* may make states change.

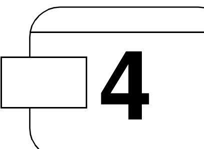


It is *not* a way of describing the *internal processing* of the object

© ADALOG J-P. Rosen



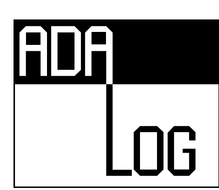
OSTD example



Operation of a water lock

Close_Lower_Gate
Open_Lower_Flow
start
Lower_Gate_Open
Close_Lower_Gate
Open_Lower_Gate
Close_Upper_Gate
Open_Upper_Flow
Gates_Closed_Water_Down
Open_Upper_Flow
Close_Lower_Flow
Flowing_Up
Flowing_Down
Close_Upper_Flow
Open_Lower_Flow
Gates_Closed_Water_Up
Open_Upper_Gate
Close_Upper_Gate
Upper_Gate_Open

© ADALOG J-P. Rosen



Concurrency constraints

MTEX (MuTual EXclusion request)

- Lock at *operation* level.
- No other call to the operation can execute simultaneously.

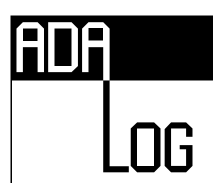
RWER (Read Write Execution Request)

- Complete lock at *object* level
- No other call of a RWER or ROER operation from the same object can execute simultaneously.

ROER (Read Only Execution Request)

- Read lock at *object* level
- No other call of a RWER operation from the same object can execute simultaneously (calls to ROER operations *allowed*).

© ADALOG J-P. Rosen



Protocol constraints (1)

HSER

Highly Synchronous Execution Request
Wait till the end of the service

LSER

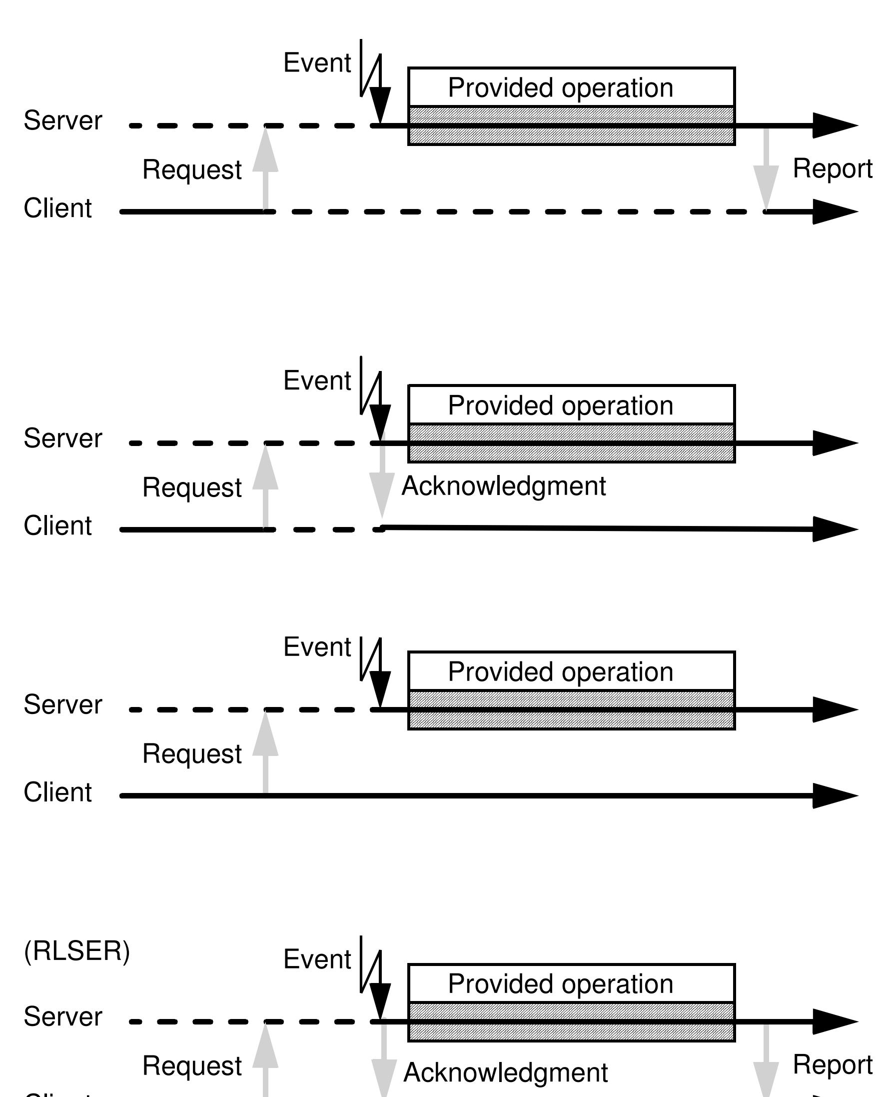
Loosely Synchronous Execution Request
Wait till the service is accepted

ASER

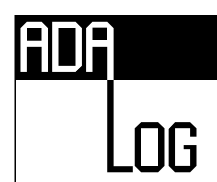
Asynchronous Execution Request
Don't wait

RLSER, RASER 4

Reporting LSER, Reporting ASER
As LSER and ASER, then waits for the result



© ADALOG J-P. Rosen

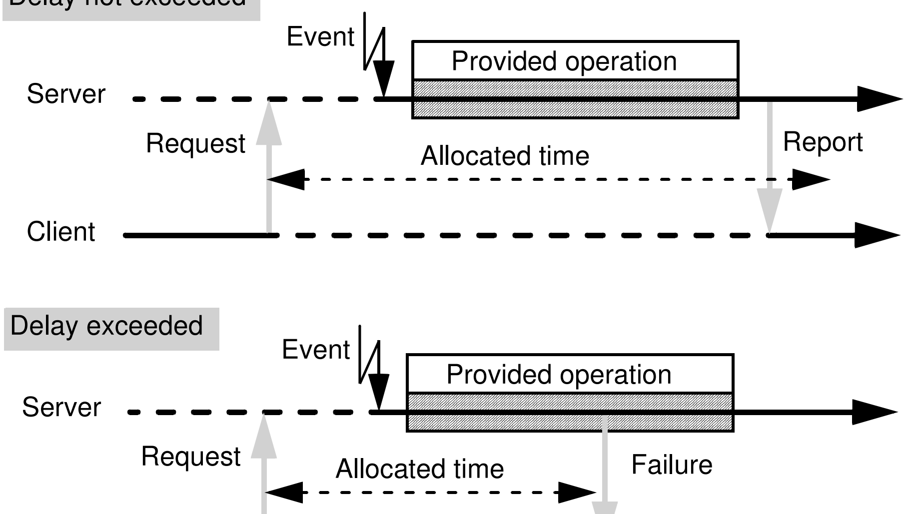


Protocol constraints (2)

TOER

Timed Out Execution Request

Additional qualification over some other constraint (HSER_TOER, LSER_TOER, etc.) to limit waiting time.



- If time limit not exceeded:
  - ☞ as before
- If time limit exceeded
  - ☞ The client is awaken with an exception.
  - ☞ The server proceeds normally (the report is *lost*).

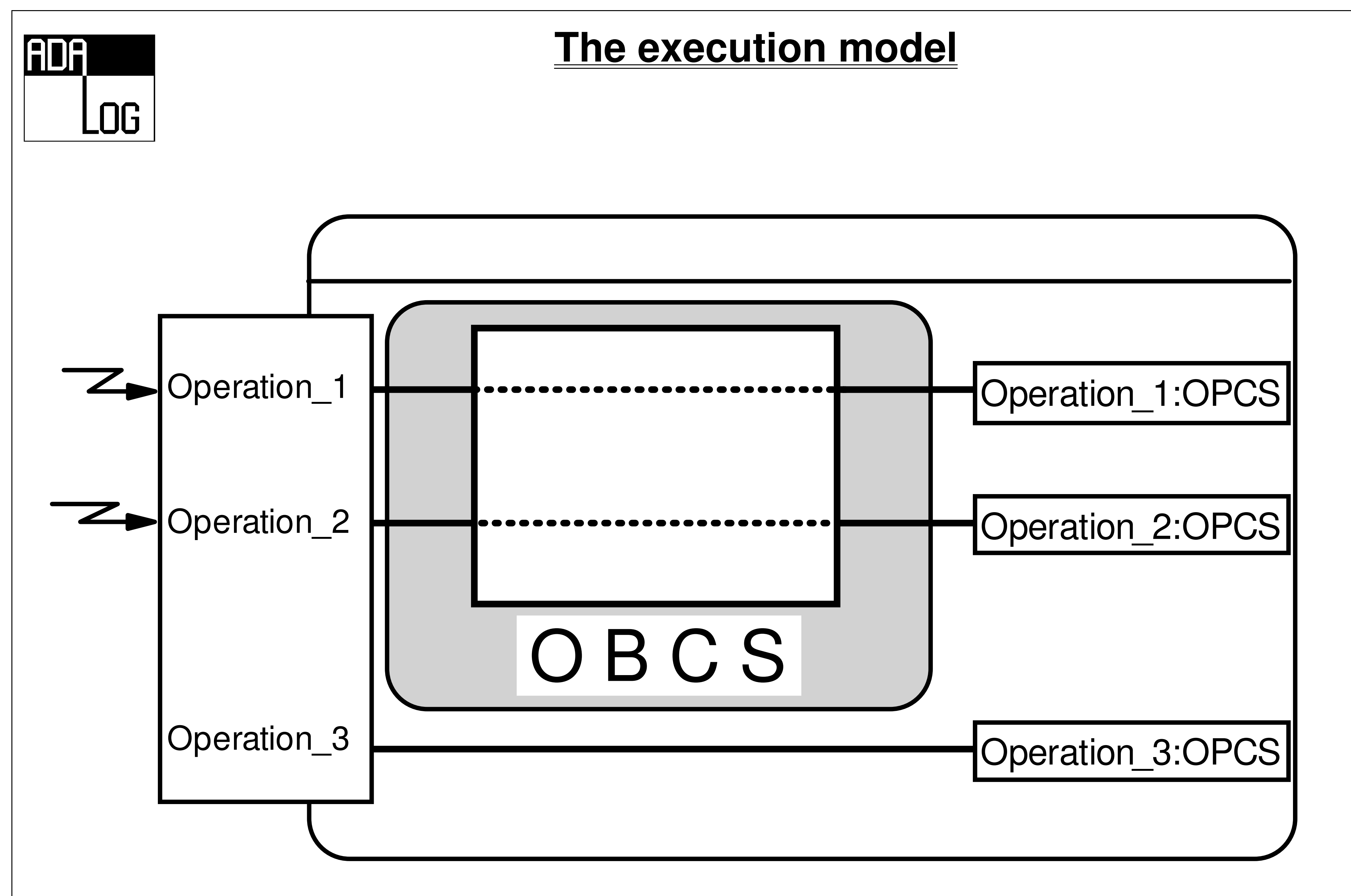
ASER_BY_IT

Asynchronous Execution Request by Interrupt

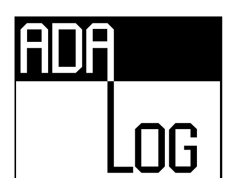
Modeling of interrupts

- ☞ Normal ASER, but the client is some hardware device

© ADALOG J-P. Rosen



© ADALOG J-P. Rosen

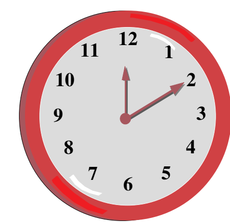


Passive objects and active objects

Passive Objects

- Do not own any flow of control of their own, but are activated by other flows of control.
  - ☞ The state of the object does not depend on time; it can be modified only by calling provided operations.
  - ☞ Subprogram semantics.
- Only state constraints allowed.
- No circularity in the USE relationship.

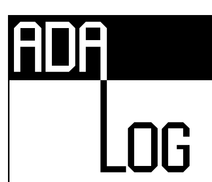
Active Objects



- Own one or several concurrent flows of control.
  - ☞ The state of the object may change with time, even if no provided operation is called.
  - ☞ Task(s) semantics.
- All constraints allowed.
- Actions of an active object are not related (in time) to those of other object; *synchronisation* might be necessary.

Most real world objects are active !

© ADALOG J-P. Rosen

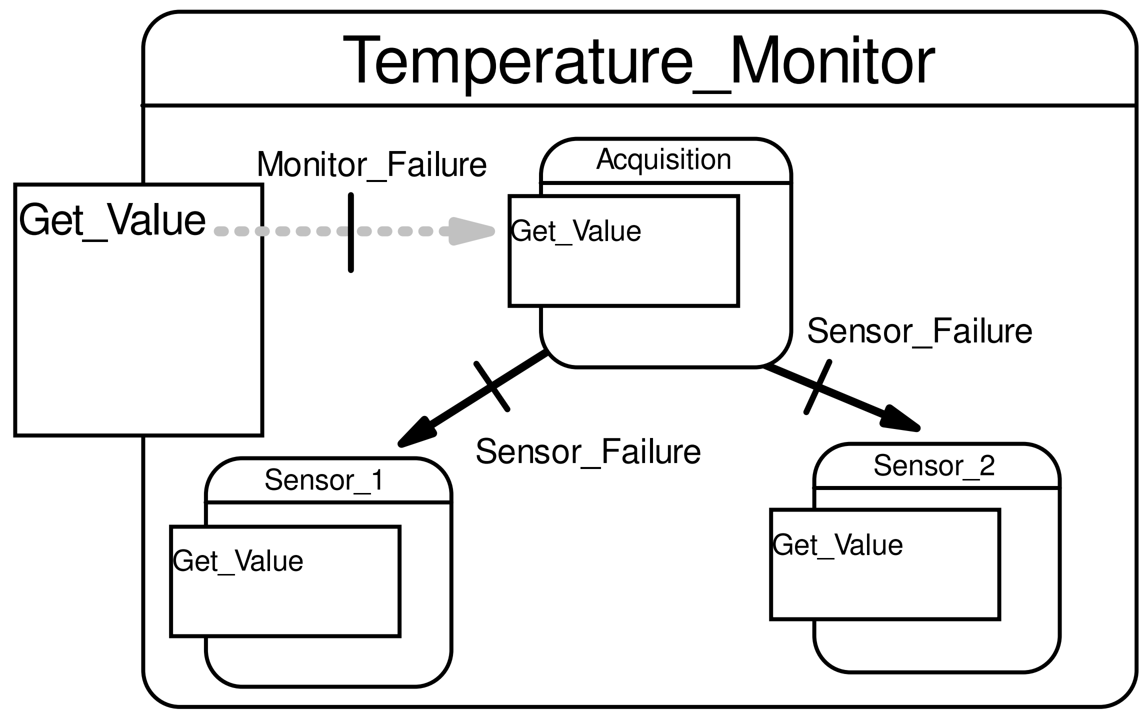


Exceptions

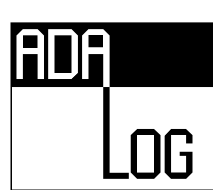
Follow Ada's semantics for exceptions.

Are transmitted from server to client.

Precise meaning of exceptions is documented in the EXCEPTIONS section of the ODS, and in each OPCS.



© ADALOG J-P. Rosen

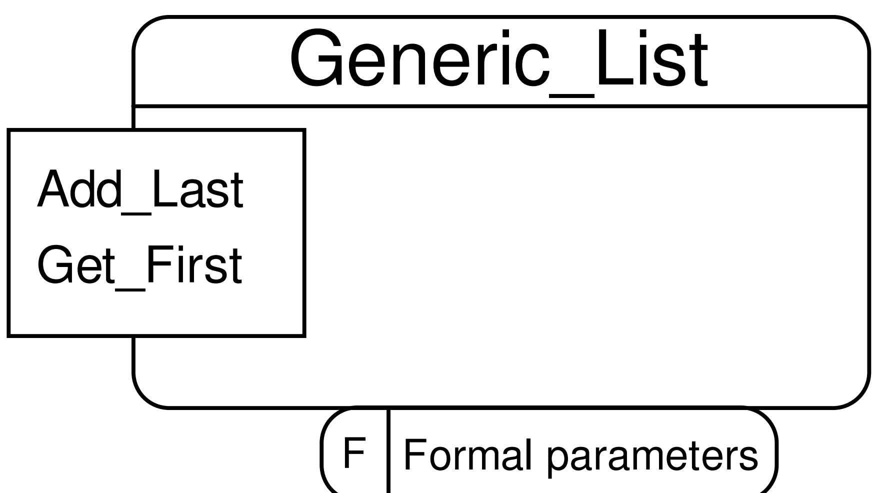


Generics

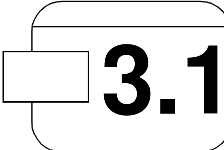
Parameterizable models that provide objects, HADTs or classes.

A generic:

- ❑ Is always a root
- ❑ May use only environment objects.
- ❑ May contain instances of other generics.

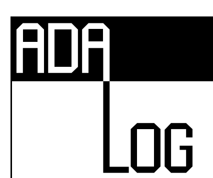


Beware!

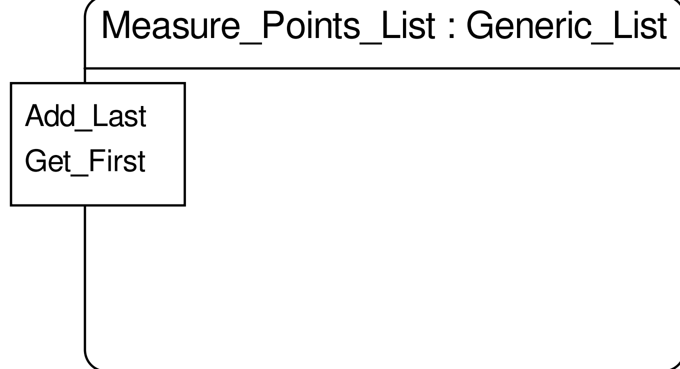


Generics were called classes in HOOD <= 3.1

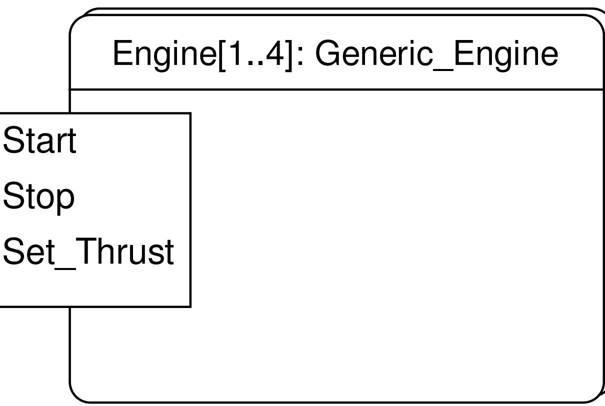
© ADALOG J-P. Rosen



Generic Instances



Single instance

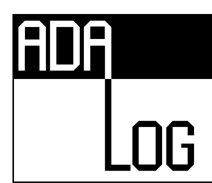


Multiple instance

An instance:

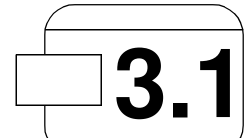
- Can be single or multiple
- Is recognized by the "type" in its name
- The dependency from the instance to the generic is *not* traced.

© ADALOG J-P. Rosen



Notion of virtual node

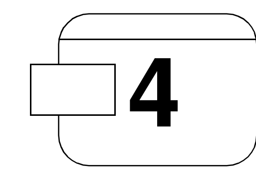
A virtual node is an *object* that can be allocated to a different physical node than other virtual nodes.



- ☞ A virtual node is decomposed either into other virtual nodes, or into active of passive objects.

  (The parent of a virtual node is always a virtual node.)
- ☞ A "terminal" virtual node executes on a single physical node.
- ☞ Several virtual nodes may execute on the same physical node.
- ☞ Allocation of physical nodes to virtual nodes can be static or dynamic.

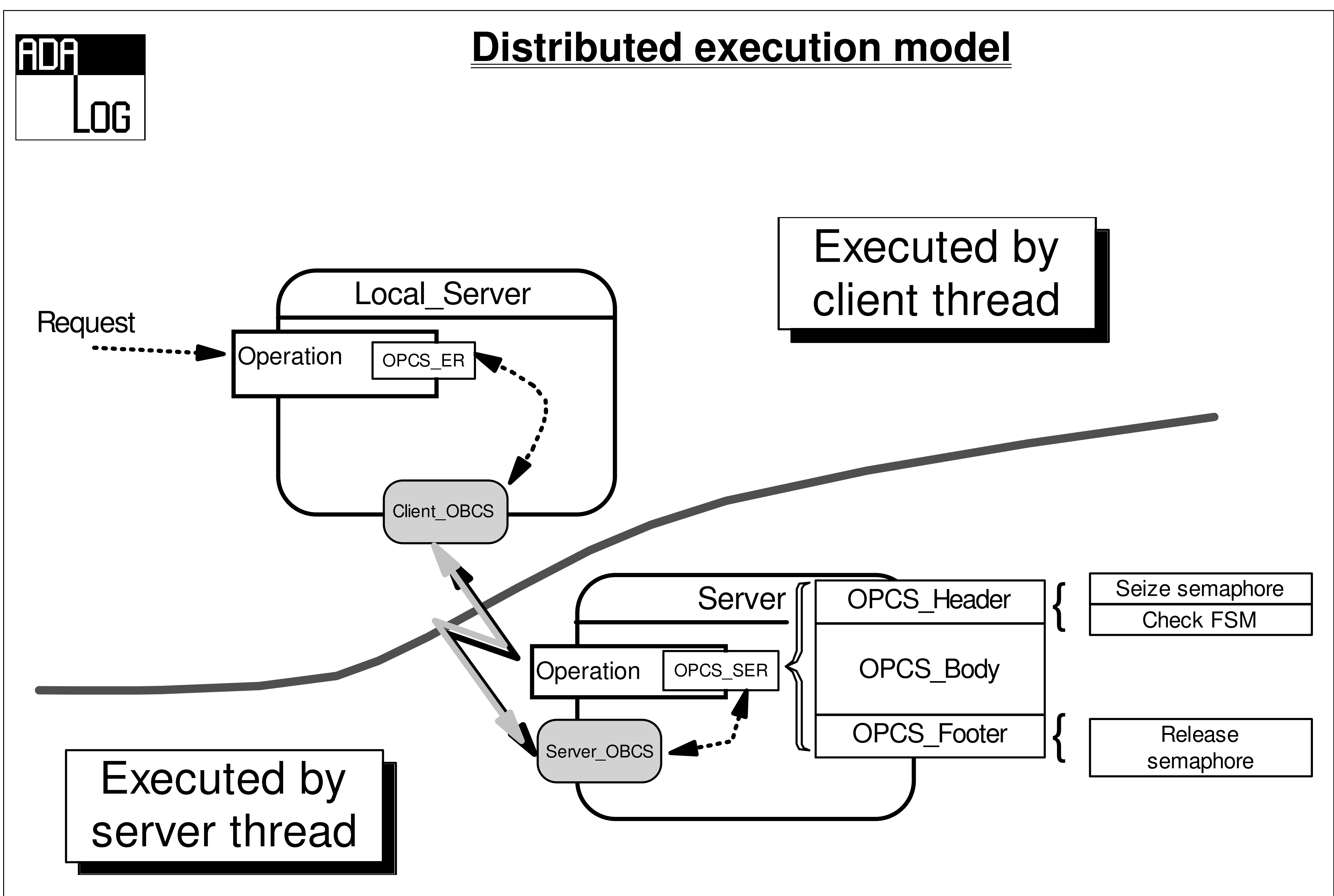
Virtual nodes represent the *logical* architecture of the system



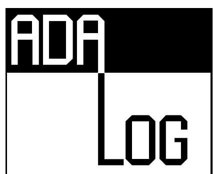
- ☞ A virtual node can be decomposed only into other virtual nodes.

An extra step is used to allocate objects to virtual nodes, then virtual nodes to physical nodes.

© ADALOG J-P. Rosen



© ADALOG J-P. Rosen



Complete example, client-server view

E | Network

EMSystem

boot
ems
{GUI_Call_Backs:
exit_ems
{Mail_Tool_Call_Backs}
{Administration_Call_Backs}
{Configuration_Call_Backs}
}

A | Mail_Server
Run
Send
Receive

ASER

T_Messages

GUI
Run
End_Run
{Mail_Tool_Call_Backs}
{Administration_Call_Backs}
{Configuration_Call_Backs}

Buffer
Put_Message
Get_Message
Size

Unauthorized_User

Administration
Add_User
Delete_User
Change_User

T_Directory_Data

T_Letters
Create
Delete

T_Messages
Create
Delete

T_Configuration_Data

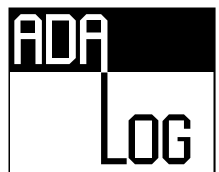
T_Letters

Configuration
Store
Retrieve
Default_Settings

Mail_Tool
Read
Write
Reply

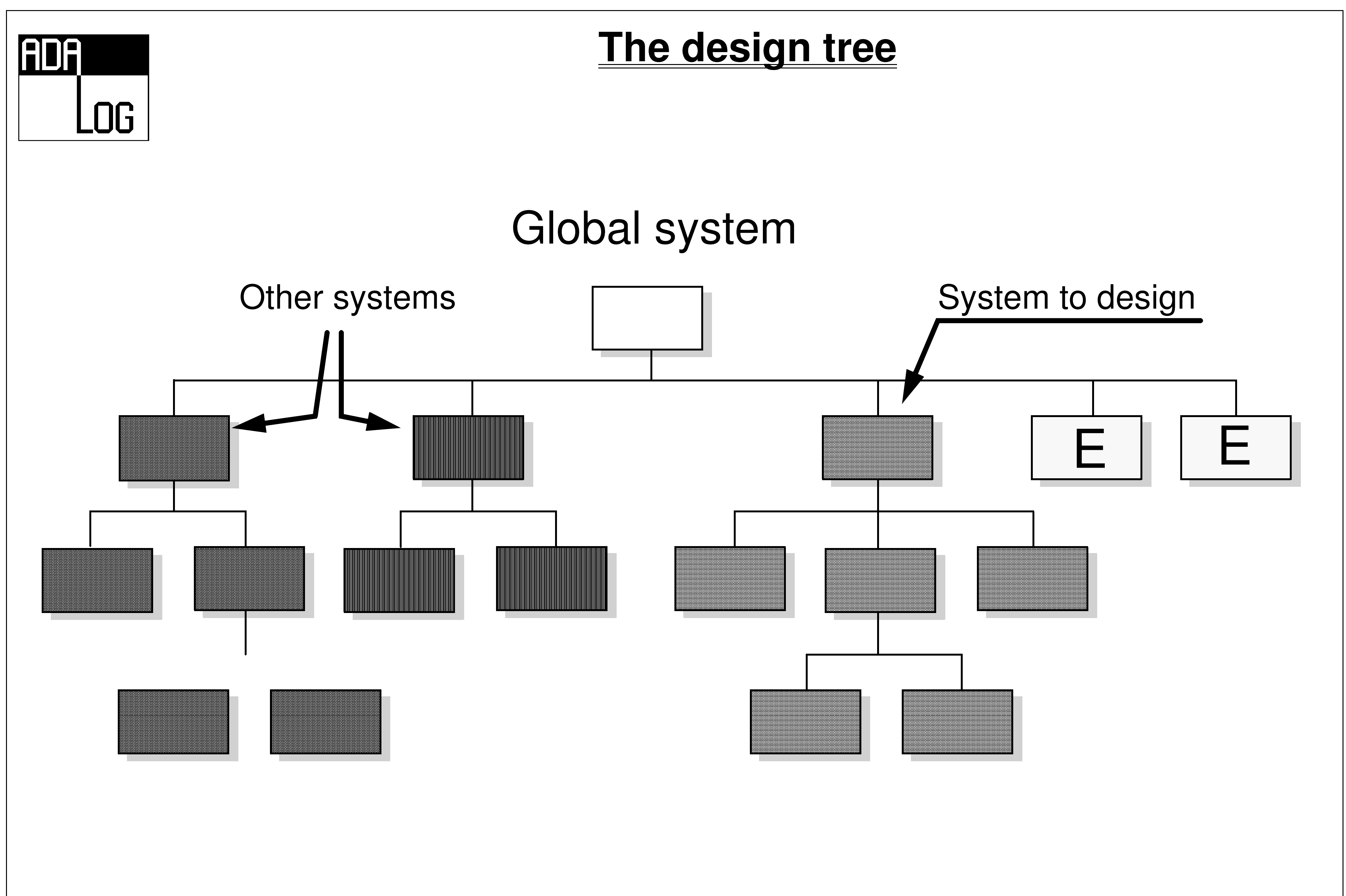
T_Messages

© ADALOG J-P. Rosen

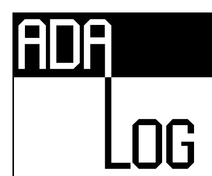


Complete example, structural view

© ADALOG J-P. Rosen



© ADALOG J-P. Rosen



System Configuration

A root object represents a *system to design*.

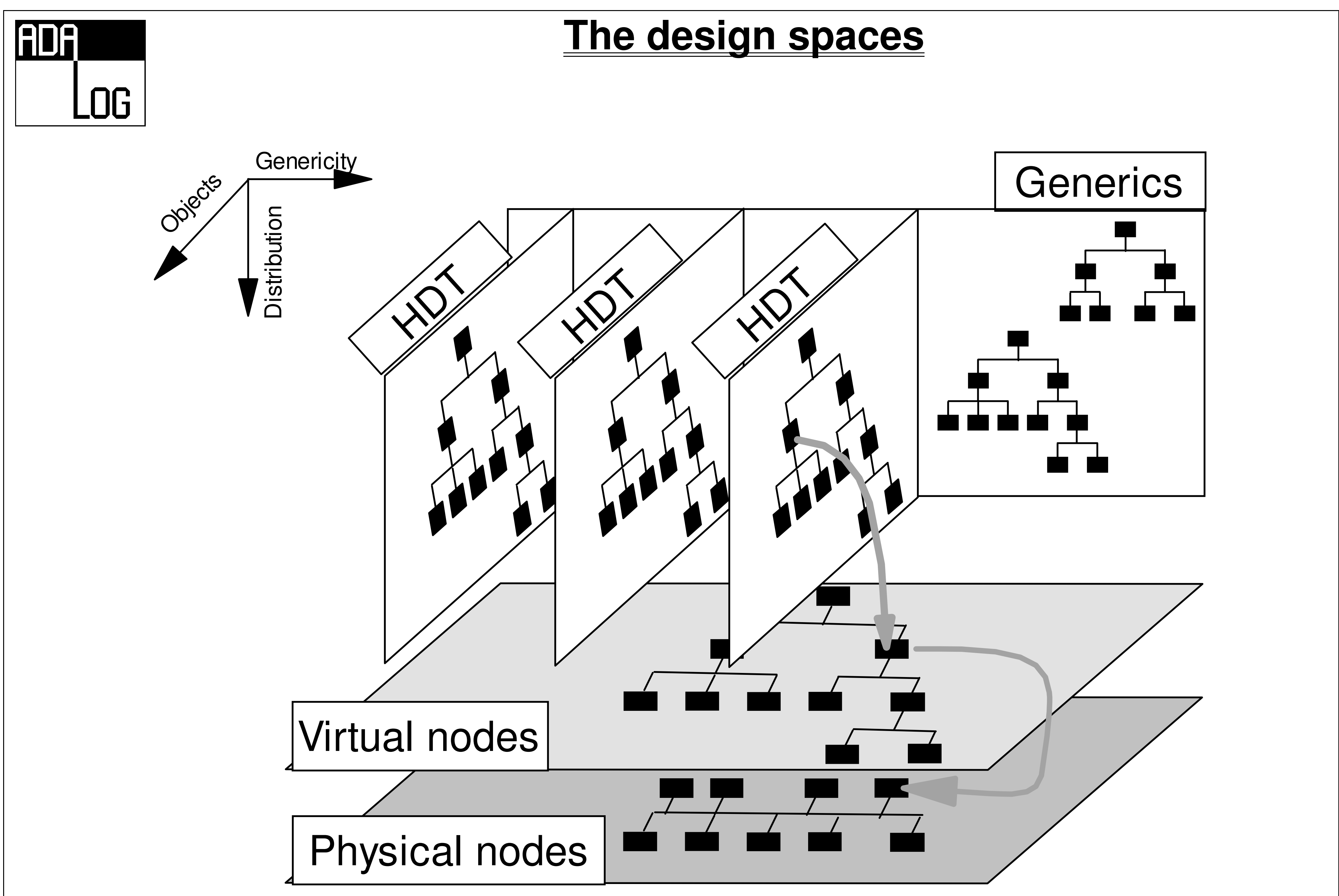
- there may be several root objects.
- Every root object is an environment object for other root objects
- A root object is defined by its interfaces to the external world

A *configuration* is a set of root objects, generics and virtual nodes.

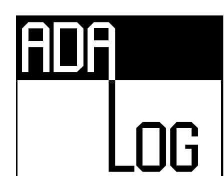
SYSTEM_CONFIGURATION
    ROOT_OBJECTS
        Root objects
    ROOT_GENERICS
        Generics
    ROOT_VN
        Virtual nodes
END

© ADALOG J-P. Rosen

The design spaces



© ADALOG J-P. Rosen

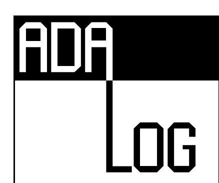


Design method with HOOD

Points to be met by *every* design method with HOOD

- Top-down analysis
- Understand the *problem* before looking for a *solution*.
- Proceed by *successive refinements*
  - ☞ Refine decompositions
  - ☞ Refine data
  - ☞ Refine physical implementation
- *Note* every design decision in the graphical representation, and update the ODS.
- Never forget to *justify every design decision*.

© ADALOG J-P. Rosen



An example of a methodological process

1 Definition of the problem
   ! Understand the problem: statement, analysis & restructuring of requirements.
   ✍ Describe the problem in the ODS.

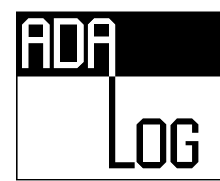
2 Elaboration of an informal strategy
   ! Find a solution
   ✍ .Describe the solution in the ODS.

3 Formalization of the strategy
   ! Express the solution clearly.
   ✍ .Graphical description; identify objects and operations.

4 Formalization of the solution
   ! Give details
   ✍ Fill in textual descriptions... with *relevant* information.

5 Analysis of the solution
   ! Justify choices, evaluate performances, identify reusable components, generics...
   ✍ Justification in the ODS

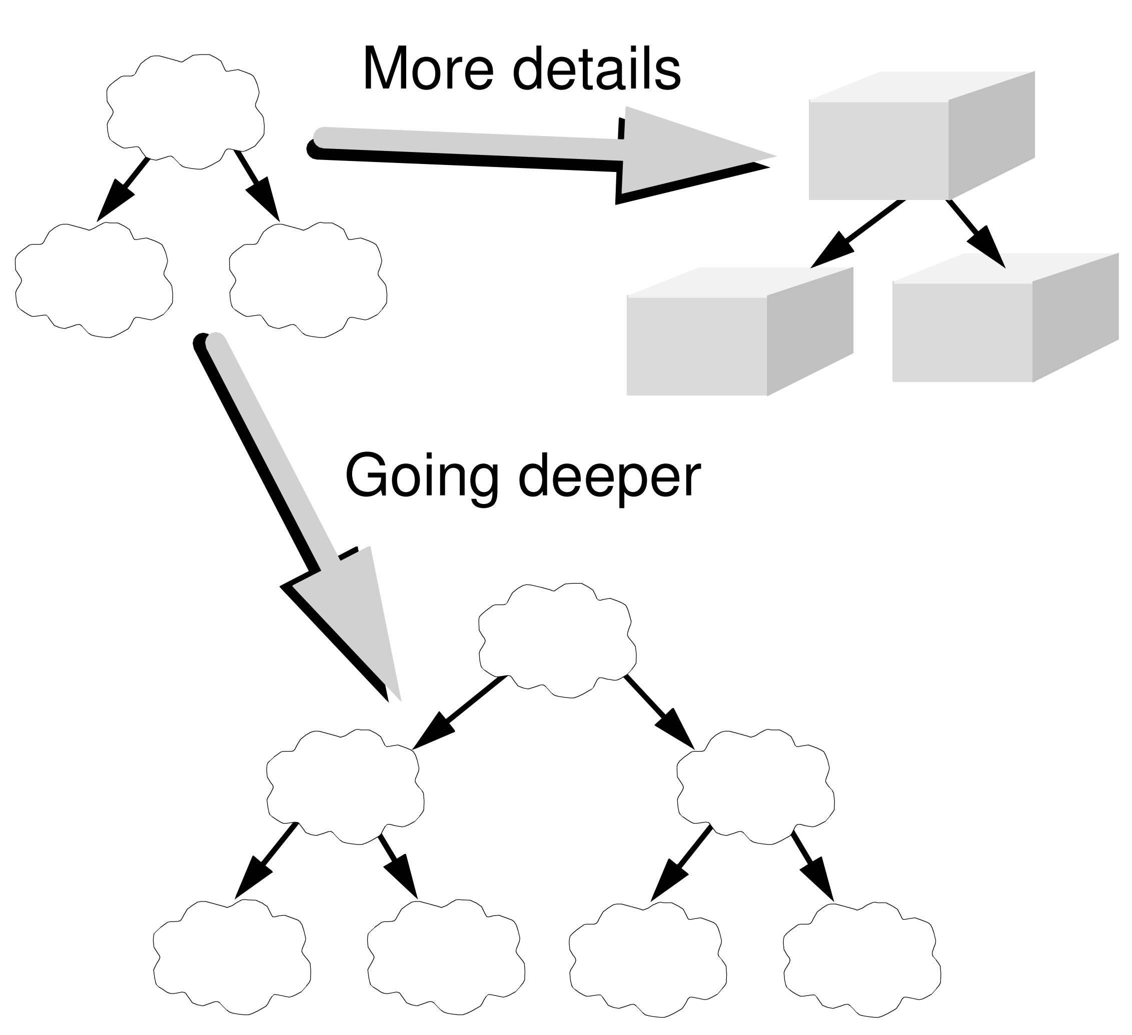
© ADALOG J-P. Rosen



Design refinements

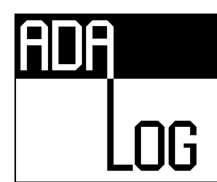
Two forms of refinements :

- By giving more details to the design
- By going deeper in the design



In practice, both happen simultaneously.

© ADALOG J-P. Rosen



HOOD rules

"C" rules

Coherence and completeness.

"G" rules

General rules

"I" rules

INCLUDE relationship

"O" rules

Operations

"P" rules

Provided Interface

"R" rules

Required Interface

"U" rules

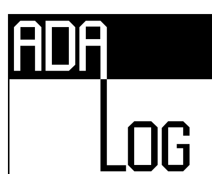
USE relationship and inheritance

"V" rules

Visibility

Rules verification is taken care of by the tool.

© ADALOG J-P. Rosen

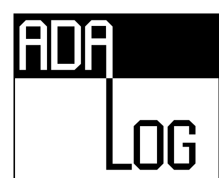


Example of HOOD/Ada translation

```
package Parent is
    procedure Operation_1;
    procedure Operation_2;
end Parent;

private package Parent.Child_1 is
    procedure Service_1;
end Parent.Child_1;

with Parent.Child_1;
private package Parent.Child_2 is
    procedure Service_2;
end Parent.Child_2;


with Parent.Child_1, Parent.Child_2;
package body Parent is
    procedure Operation_1 renames Parent.Child_1.Service_1;
    procedure Operation_2 renames Parent.Child_2.Service_2;
end Parent;
```

Parent
Operation_1
Operation_2
Child_1
Service_1
Child_2
Service_2

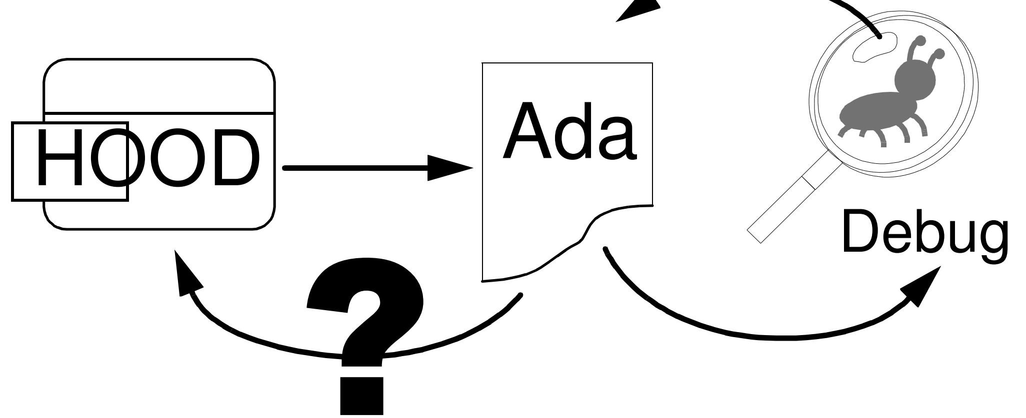
© ADALOG J-P. Rosen



From design to coding

The theory : always work within HOOD

- ❑ The tool takes care of code generation.
  - ☞ requires a quasi-perfect tool

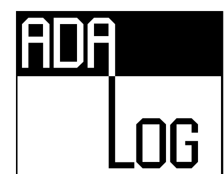


In practice: Adjustments of the generated code may be necessary.

How to keep the configuration up to date?

- ❑ Working with the HOOD tool
  - ☞ Automate corrections: scripts, patches...
  - ☞ risk of dirty hacking
- ❑ Reverse engineering
  - ☞ requires a reverse engineering tool
- ❑ Single code generation
  - ☞ Design is frozen at the time coding phase is started
  - ☞ Beware of consistency!

© ADALOG J-P. Rosen



HOOD and documentation

Documentation is a strong point of HOOD

Very complete:

- ❏ All details down to code
- ❏ All dependencies are traced both ways
- ❏ Formal and informal documentation

Ability of limiting a view of the documentation:

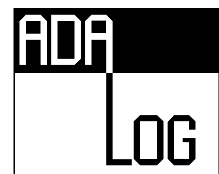
- ❏ Limiting to interesting abstraction levels
- ❏ Several levels of documentation.

Beware of abuse

With the tools, it is all too easy to produce heaps of paper...

Too much documentation = *No* documentation

© ADALOG J-P. Rosen



HOOD and UML

UML is a meta-notation

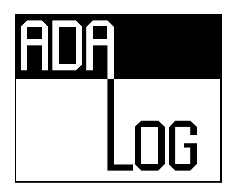
- No design process
- Requires an associated method (OMT...)
- Constructs of the method are represented by stereotypes

HOOD is a design process

- It defines its own notation.
- Nothing prevents HOOD objects from being represented in a different form.

There is no incompatibility between HOOD and UML

© ADALOG J-P. Rosen



HOOD users

Space (of course)

Spot, Helios, Jason[3], Ariane 5, ...

Transportation

Meteor, TGV speedometer, Airbus...

Energy

Nuclear plants (France, Belgium)

Military

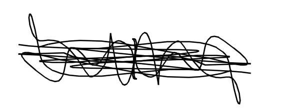
Tigre helicopter, Eurofighter,  ,...

Photo Aerospatiale Matra Airbus

Artist view of Topex - Poseidon operation diagram © CNES/ D.Ducros

Doël 3 control room © TRASYS

© ADALOG J-P. Rosen