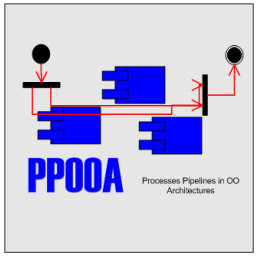
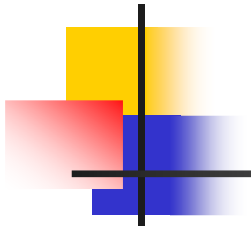


An Integrated Systems and Software Engineering Process (ISE&PPOOA)

Seminar at Aula Artigas. Industrial Engineering School. ETSII-UPM. Madrid (Spain). May 18, 2012.

José Luis Fernández Sánchez
Profesor titular ETSII-UPM
jose.fernandez@incose.org



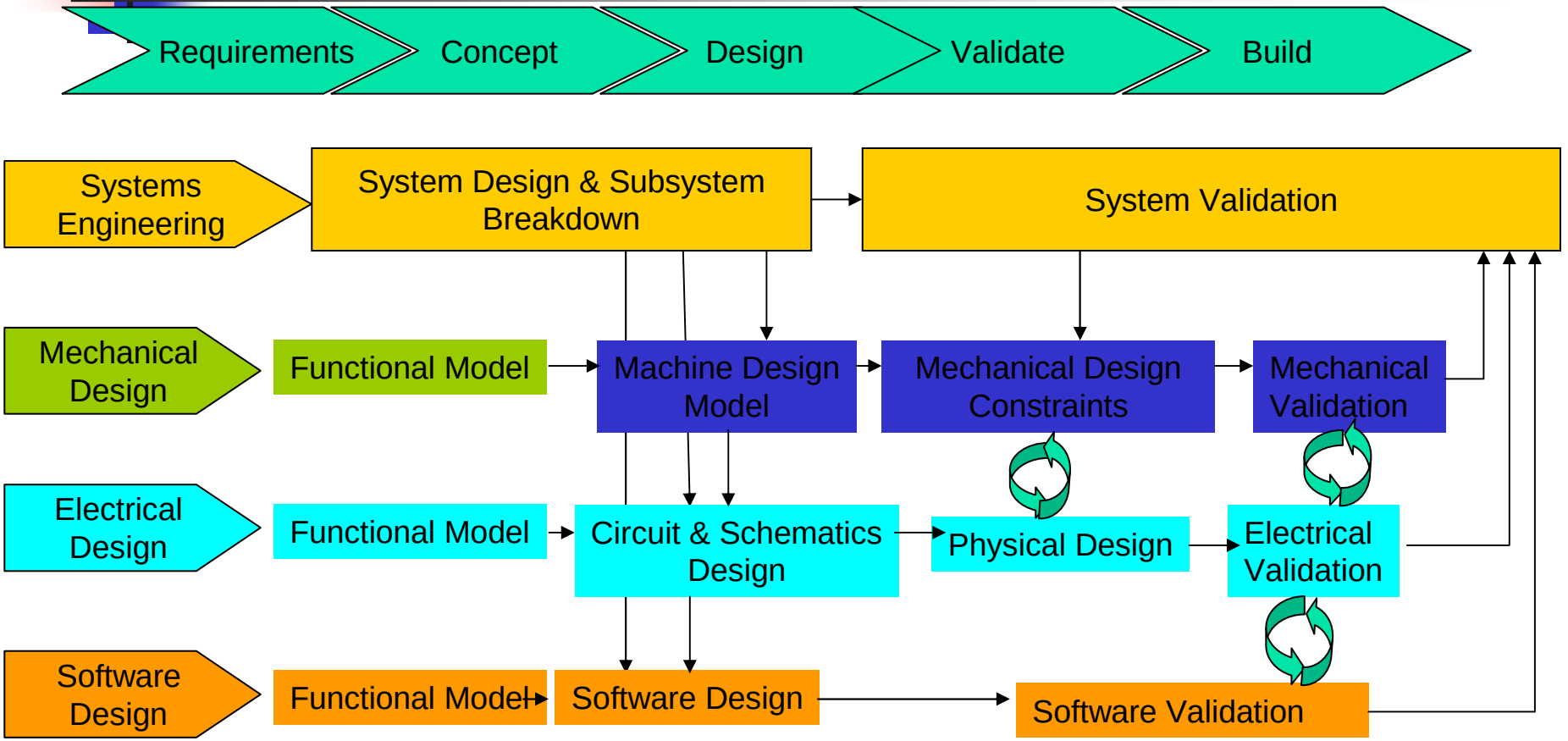
The facts

- Functions performed by software in a military aircraft has increased from 8% for the F-4 Phantom II to 80% for the F-22 Raptor.
- Today it is estimated that a premium automobile takes dozens of microprocessors running 100 million SLOC. Software made up more than 25% of a car's total value.

Scope of the presentation

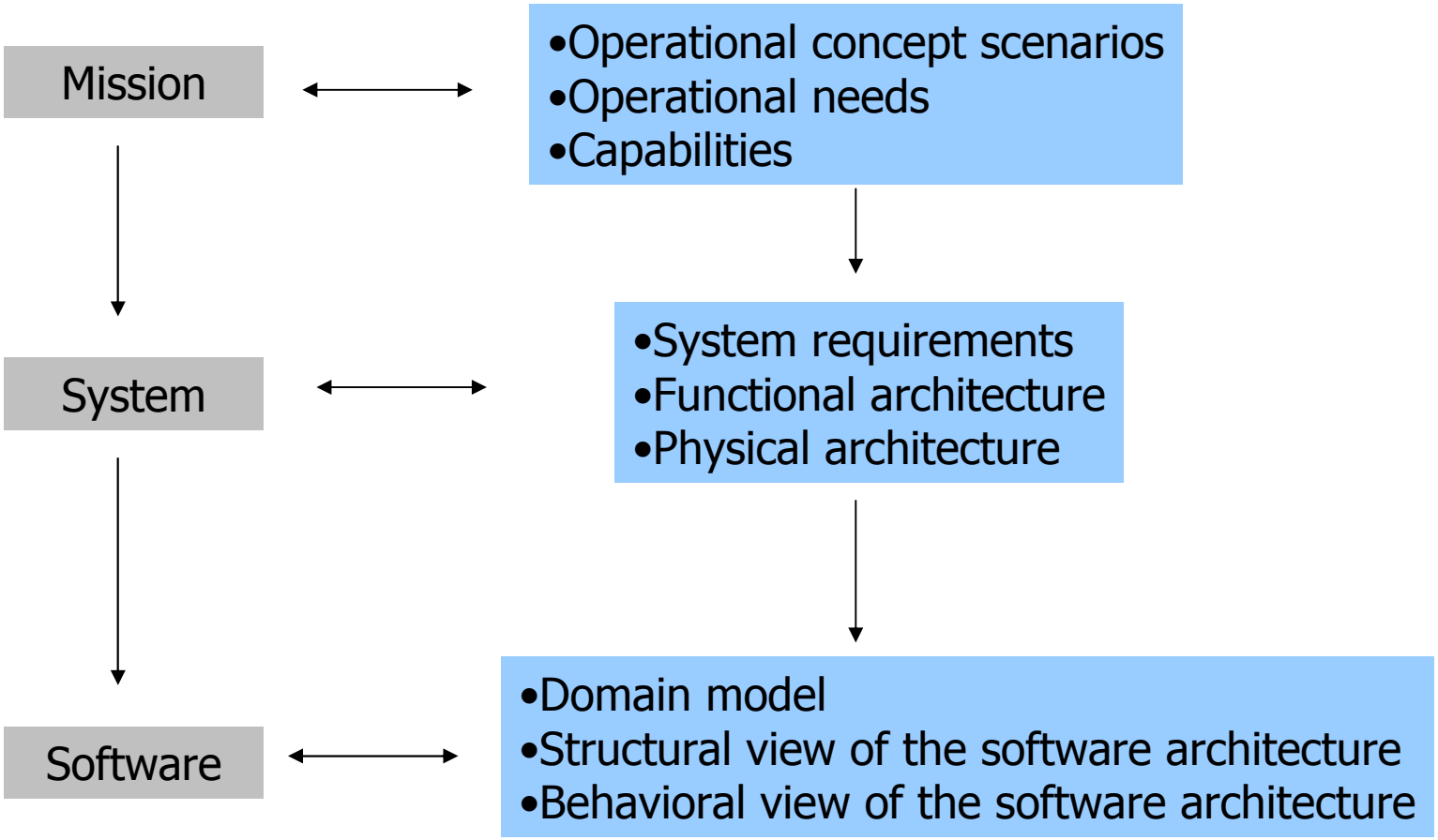
- To present an **integrated** systems and software engineering process named **ISE&PPOOA** (Integrated Systems Engineering and Pipelines of Processes in OO Architectures).
- This process applies the **functional paradigm** all over the systems development lifecycle, allowing the combination of **traditional SE, MBSE and software component based development (CBD)** using standard notations such as SysML and UML with some extensions and refinements for the software subsystems.
- Development of a system can be envisioned as an amalgamation of three aspects: **mission, system and software**.

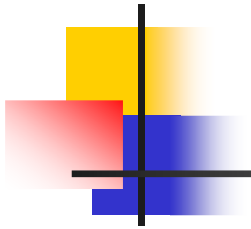
Systems engineering in mechatronic systems



From ARC Advisory Group

ISE&PPOOA in a nutshell

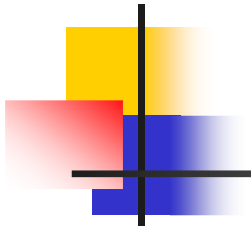




Concepts used

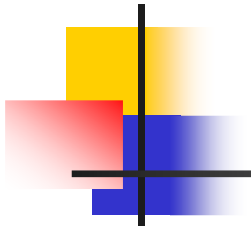
- **Operational concept**: abstract model of the operations of a specific system or group of systems.
- **Scenario**: instance of how the system is used in specific circumstances
- **Operational need**: a need that complements the system usage but is not contained in the scenarios
- **Capability**: the ability to perform an effect.

Some issues considered regarding requirements

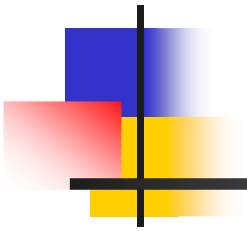


- **Functional requirements** define specific behavior or functions of the system.
- **Non-Functional requirements**, known as **quality requirements**, specify criteria that can be used to judge the development or usage of a system, rather than specific behaviors. They act to **constrain** the architecture of the solution.
- In contrast to functional requirements that are allocated to system parts, non-functional requirements allocation is essentially different. Sometimes non-functional requirements are budgeted to the whole system or one of its parts. In other situations the non-functional requirement is satisfied by a design **heuristic** or **tactic**.

Some issues considered regarding interfaces



- Missing or incorrect interfaces are a major cause of project costs overruns and system failures.
- **ISE&PPOOA** uses **SysML diagrams**, text and **tables (N^2 charts)** to describe system external and internal interfaces.
- **N^2 charts** are very compact, allowing the overview of even the most complex systems.
- **N^2 chart** is a very helpful tool to allocate functions to subsystems or system parts such that there is minimal interaction among the components



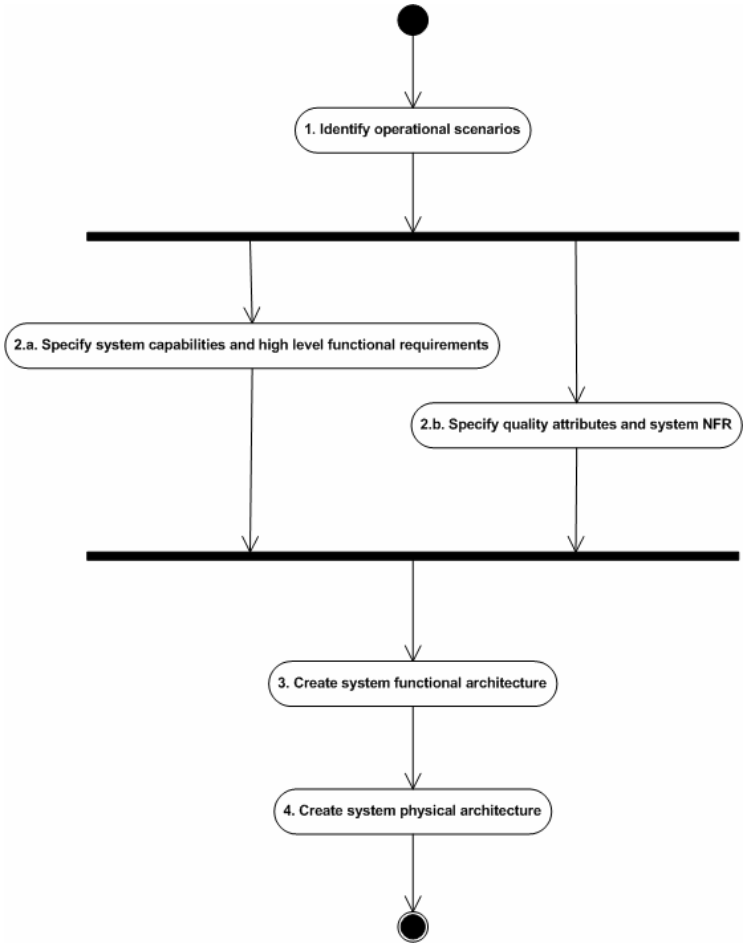
The ISE&PPOOA process

How to develop a software intensive system from the system to its components

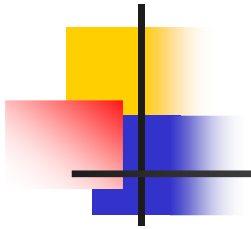
The ISE subprocess

- The main goal is the creation of the **functional and physical architectures** of a system identifying the subsystems and their interfaces.
- The system may have subsystems software intensive and/or non software intensive where **Physics conservation laws of mass, energy and momentum** are an important issue that should be considered when representing the system views.

The ISE subprocess



The ISE subprocess. Step 1. Identify operational scenarios



- Identify the **operational context** of the system and describe its **operational scenarios** for different **modes of operation**
- The system intended behaviors are described by the operational scenarios, where additionally to the preconditions, post conditions and steps of each scenario, the **operational needs** are identified also

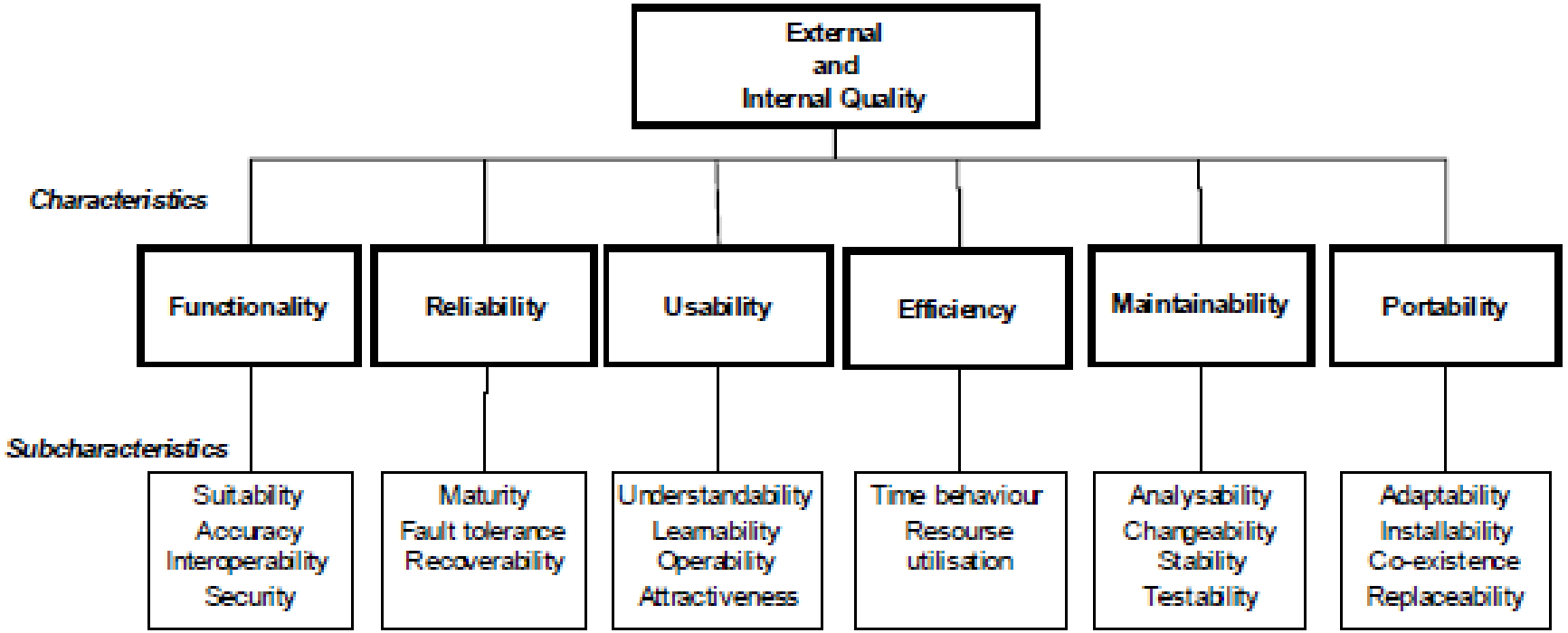
The ISE subprocess. Step 2a. Identify system capabilities and HLR

- Transform scenarios and operational needs into a set of **system capabilities** and high level system requirements.
- The deliverable is the representation of capabilities with a **hierarchical decomposition** using the **block definition diagram** of SysML. System functional requirements are specified in **natural language.**

The ISE subprocess. Step 2b. Specify quality attributes and system NFRs

- Transform operational needs into a set of **quality attributes** for example reliability, availability, security and others including the associated non-functional requirements.
- In decomposing a non-functional requirement, the systems engineer can chose to decompose its type (security, reliability, etc) based on a selected **quality model,** or its **topic** considering if they apply to the whole system or one of its parts.
- It is possible and should be taken into account, that some non-functional requirements may be affected either positively or negatively at the same time.

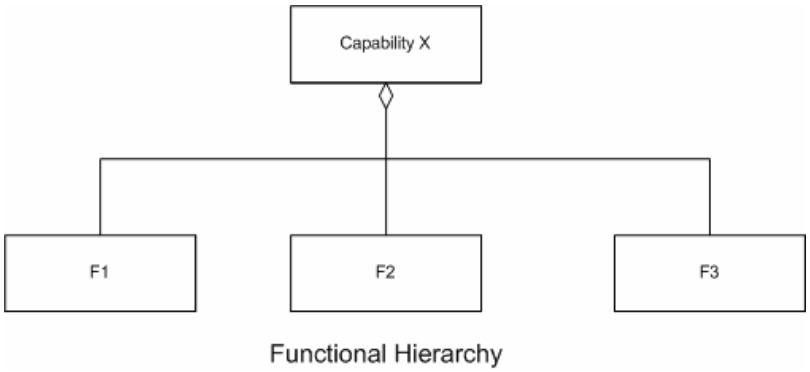
Quality model- ISO/IEC 9126



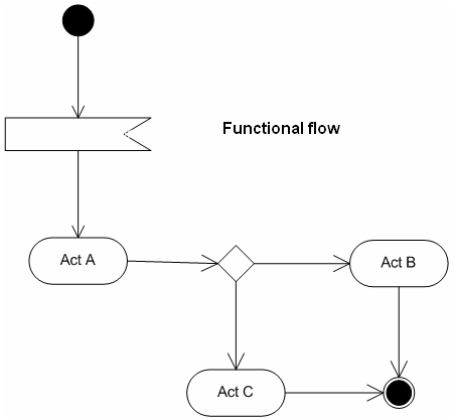
The ISE subprocess. Step 3. Create system functional architecture

- Transform functional requirements into a functional architecture identifying the functional hierarchy, functional flows and functional interfaces.
- The deliverable is the functional architecture representing the functional hierarchy using a **SysML block definition diagram**. This diagram is complemented with **activity diagrams** for the main system functional flows. The **N^2 diagram** is used as an interface diagram where the main functional interfaces are identified. A **textual description** of the system functions is also provided. We use structured natural language to describe textually each of the system functions.

Functional architecture



Functional Hierarchy



Función (F1)	F1->F2	F1->F3	F1->F4	F1->F5
F2->F1	**Función (F2)**	F2->F3	F2->F4	F2->F5
F3->F1	F3->F2	**Función (F3)**	F3->F4	F3->F5
F4->F1	F4->F2	F4->F3	**Función (F4)**	F4->F5
F5->F1	F5->F2	F5->F3	F5->F4	**Función (F5)**

Functional interfaces

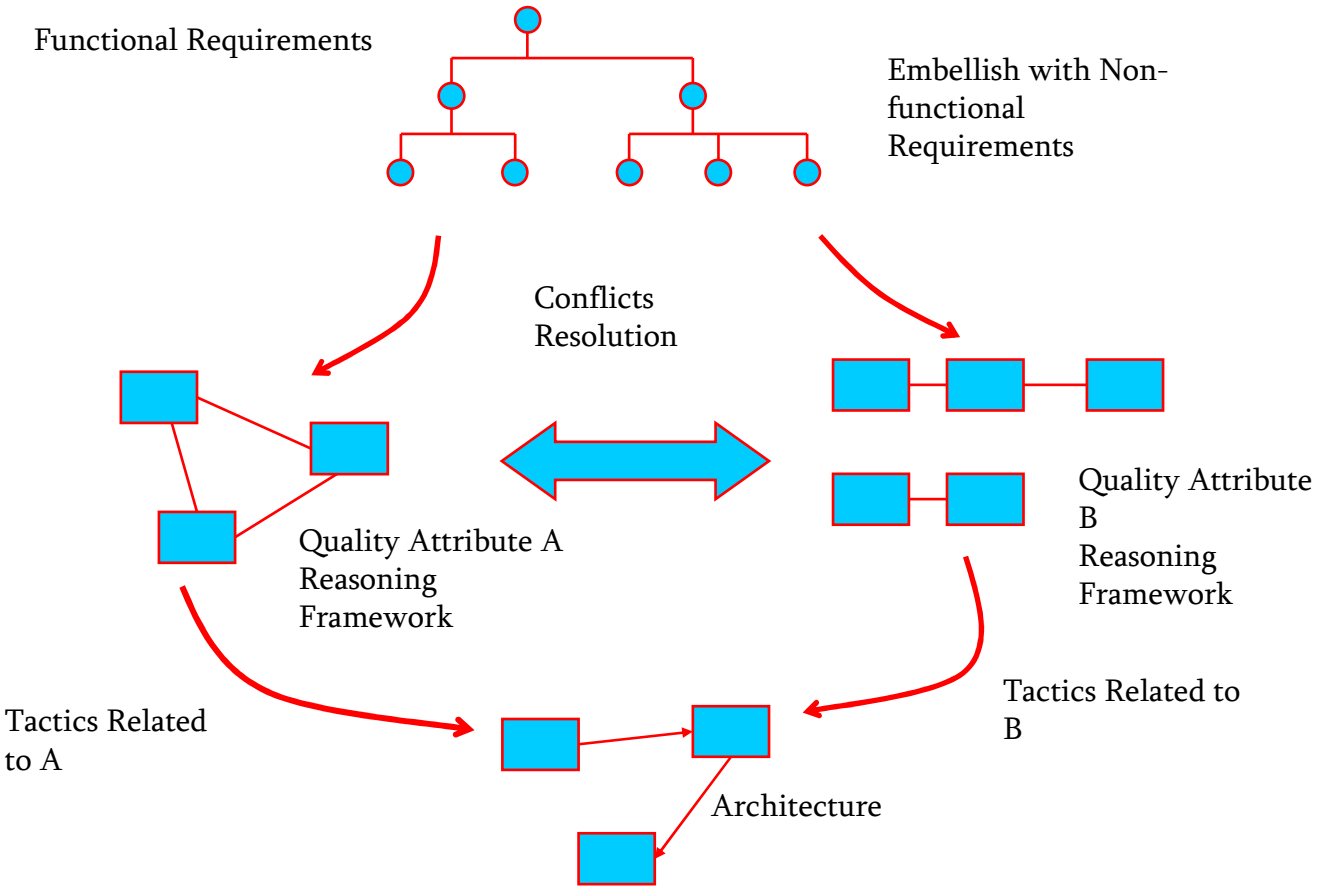
If ALTITUDE is greater than TRANSITION ALTITUDE, then:
set SPEED to MACH,
otherwise
set SPEED to AIRSPEED.

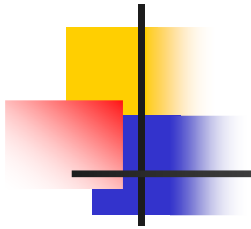
Funtion specification

The ISE subprocess. Step 4. Create system physical architecture

- Transform the functional architecture into the architecture of the solution or physical architecture.
- The selection of the solution is based on **functions clustering** and design **heuristics or tactics.**
- The deliverable is the physical architecture representing the system decomposition into subsystems and parts using a **SysML block definition diagram**. This diagram is complemented with **SysML internal block diagrams** for each subsystem and activity and state diagrams as needed. A textual description of the system blocks is also provided. The **tactics used** for the particular architecture solution are identified and documented.

Use of tactics

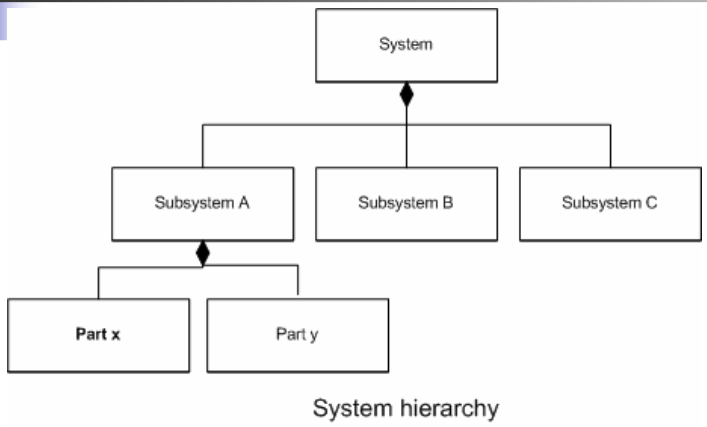




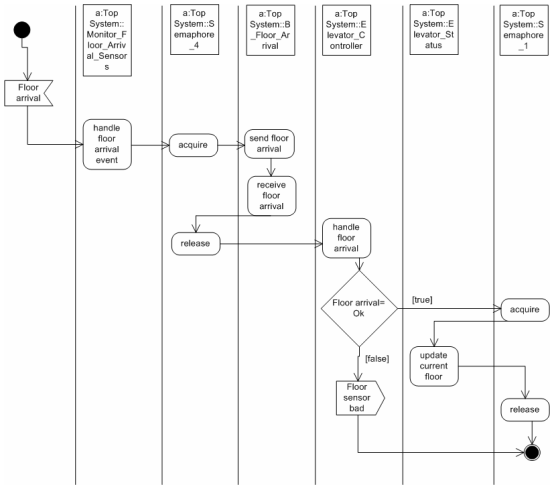
Catalogue of tactics

- The tactics catalogued are those related to:
  - General systems architecting tactics
  - Maintainability tactics
  - Efficiency tactics
  - Safety tactics
- Tactics are not necessarily independent. The application of a tactic may also require additional tactics to be applied.

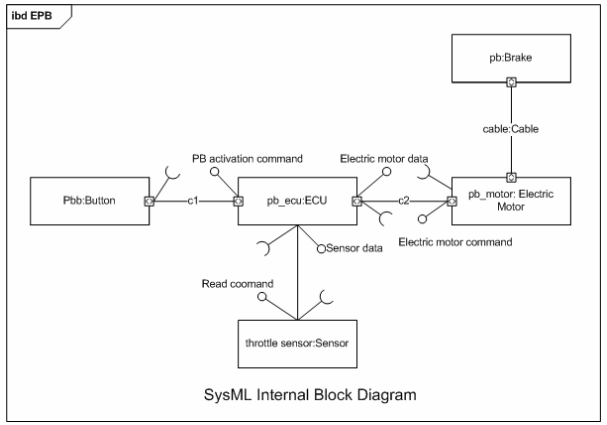
Physical architecture



System hierarchy



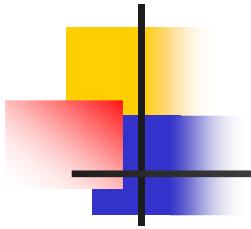
Activity diagram with swimlanes



SysML Internal Block Diagram

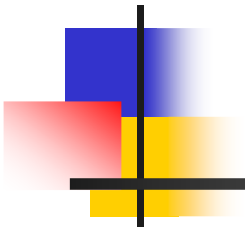
Name	Central fuel tank
Description	Fuel tank which delivers the fuel to the propulsion subsystem through the control valve of this tank. The fuel pump propels the fuel to obtain the mix together with the air coming from the environment. Finally the mix flows to the propulsion subsystem
Prov Interface	▪Fuel (from the Control Valve of the Auxiliary Fuel Tank located in the Wing to the Central Fuel Tank)
Req. Interface	•Fuel (to the Fuel pump) •Fuel Level (to the Central Fuel Tank Sensor) •Waste (to the Central Fuel Tank Drain Valve)

Part description



Experiences

- Reengineering of the Air Traffic Management system named iTEC, developed by Indra for their customers in UK and Germany. Here we applied steps 2 and 3 of the ISE subprocess to improve the systems requirements and functional architecture of the main subsystems of iTEC. The main achievements were a better structure and organization of the requirements specification documents.
- Reengineering of an Unmanned Aerial System developed by USol for civilian uses. The aerial vehicle considered was the K2B6 version of the UAVs K2 family. Main achievement is the improvement of product family evolution.
- Scenarios and Capabilities for the usage of UAVs for wildfires prevention and surveillance. Main achievement is a better understanding of the current situation and how UAVs can help.
- Scenarios, capabilities, functional and physical architecture for a home system for people with neurological disabilities (Alzheimer). Main achievement is a better understanding of the current situation and how home systems can help people with this kind of disabilities.

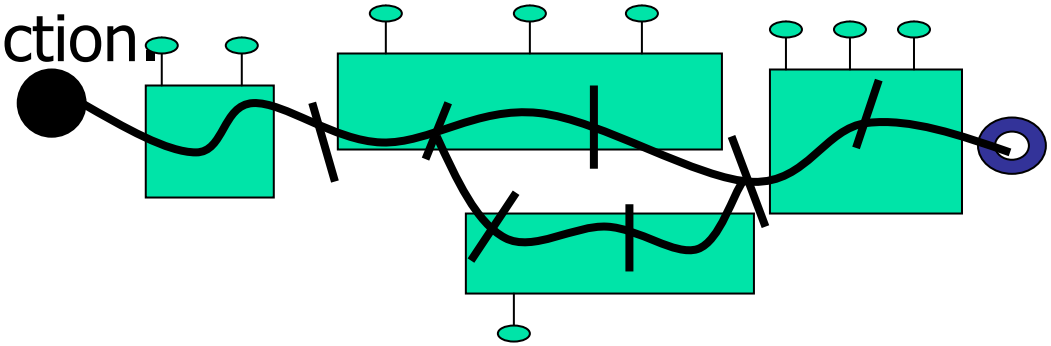


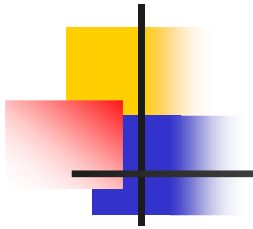
PPOOA

An architectural style and process
for architecting the software
intensive subsystems

PPOOA (I)

- PPOOA, "Processes Pipelines in Object Oriented Architectures" is an architectural style for concurrent object oriented architectures. It can be used when individual paths of execution are required to be concurrent and different processes may be positioned along the path to control the action.





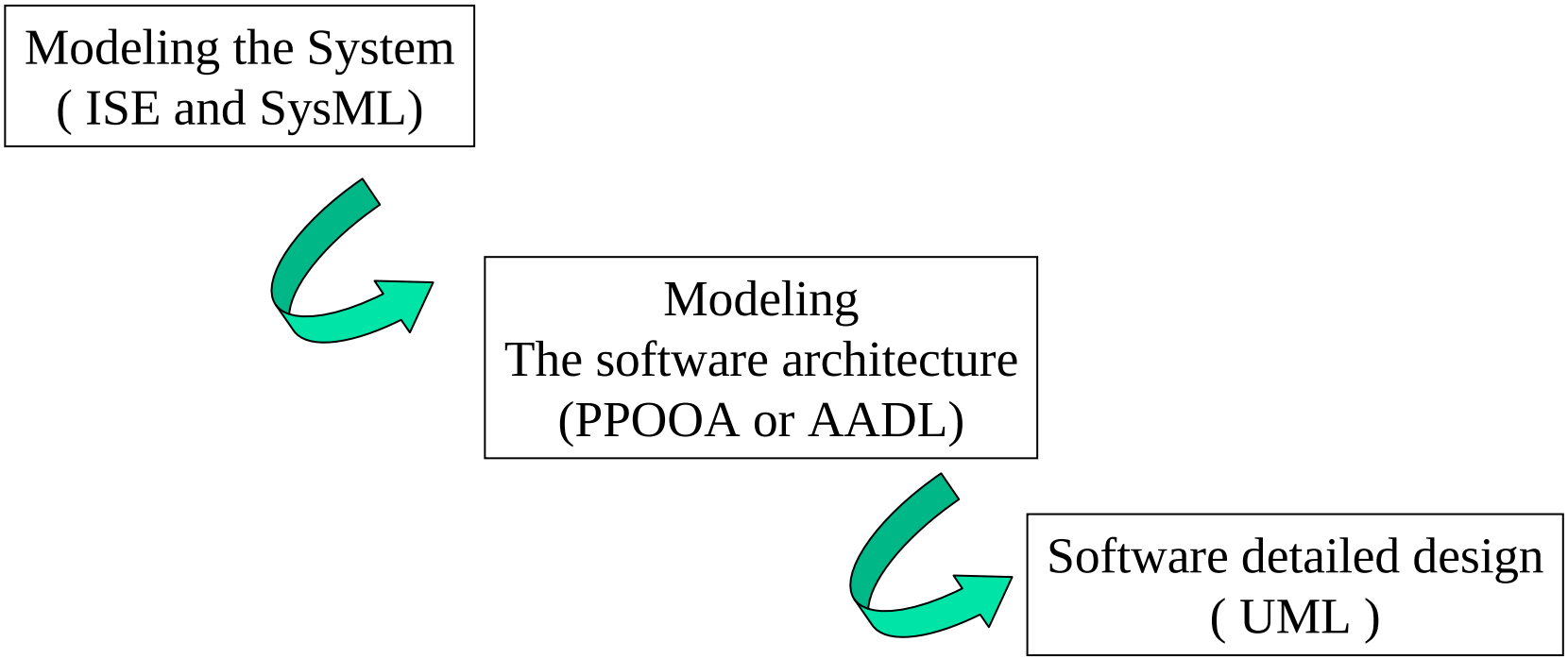
PPOOA (II)

- A model based approach for architecting software intensive real-time systems
  - Based on **UML** notation
  - Describes the system architecture using **two views** that may be supported by several diagrams; one view is the **static or structural representation**, and the other view is the **dynamic or behavioral view** of the system. The behavioral view is represented by modeling the system responses to events.
  - Supports a diversity of **components** and **coordination mechanisms** (for synchronization and communication) not found in UML.
  - Provides a tool agnostic **architecting process (PPOOA_AP),** defining the steps to build the architecture
  - Implemented in a **CASE** tool **(PPOOA- Microsoft Visio 2003)**

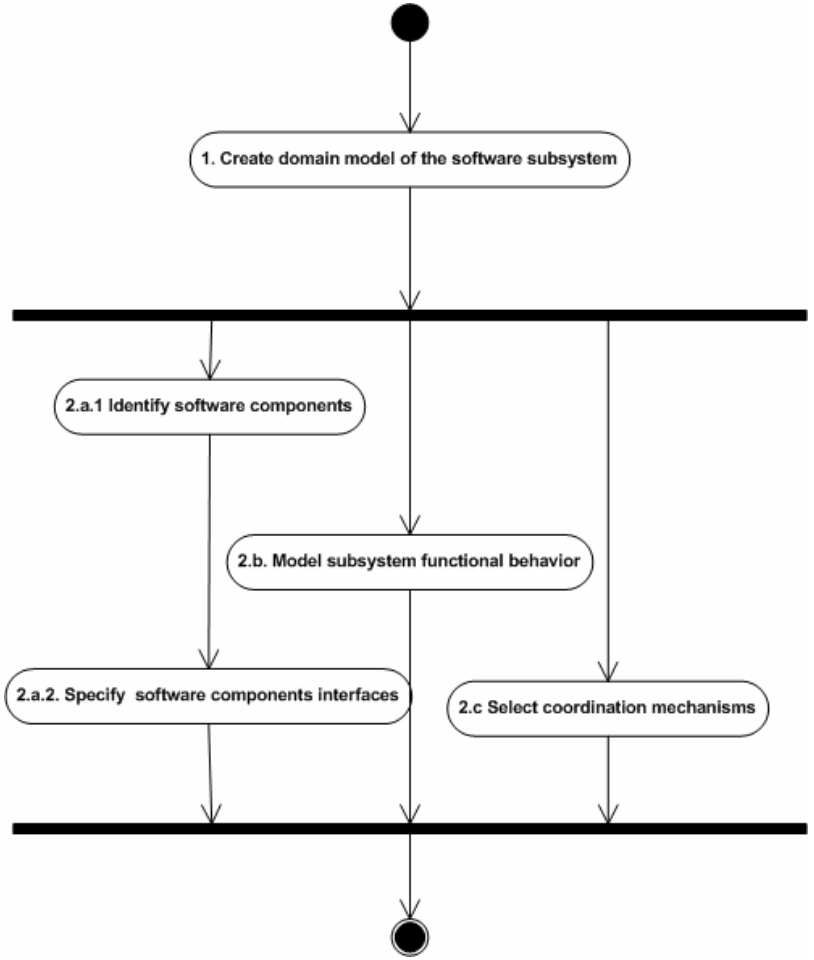
Why a new software architecting process?

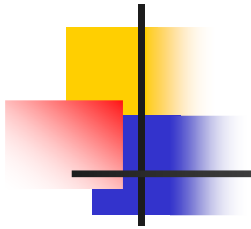
- Traditional Component Based Development and Object Oriented architecting approaches focus upon producing **encapsulations and abstractions** for system componentry.
- The effort of the resulting architecture on the ability of a system to meet its timing constraints requires additional understanding well beyond functionalities and their combined computational timing requirements.
- **Concurrency Modeling** and synchronization behavior become a dominant concern early in the architecture development whenever time is a critical factor
- Object definition and collaboration strategies should reflect meaningful **timing constraints**.

PPOOA in the MDD (Model Driven Development) lifecycle



The PPOOA subprocess





Domain model

- The integration between the systems engineering modeling subprocess (SE) and the PPOOA software engineering modeling subprocess is achieved by using a **responsibility driven** software analysis approach supported by **CRC cards**, a technique proposed in the OOPSLA´89 by Beck and Cuningham
- A **domain model** yields a more precise specification of requirements than we have in the results from earlier requirements specification. It is described using more formalism than textual descriptions, for example **UML class diagrams**, and can be used to reason about the internal workings of the software intensive subsystems.

PPOOA Subprocess

I. Identify independent types.

IDENTIFY SW. COMPONENTS

II. Assign responsibilities to the components identified.

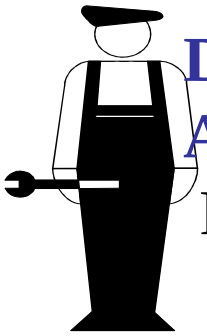
III. Select the most suitable PPOOA vocabulary element for the components identified.

IV. Assign real-time attributes to components.

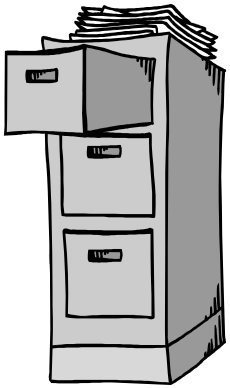
V. Determine composition relationships between components.

Select the most suitable component

Controller :
Manages external events



Domain component/ Algorithmic component:
Performs operations



Structure:
Maintains relations between objects

Process:
Coordinates work to others

PPOOA Subprocess

SPECIFY COMPONENT INTERFACES

I. Identify component operations.

II. Group component operations in interfaces to be provided by the Component.

III. Determine component required interfaces.

PPOOA Subprocess

I. Identify events and their arrival patterns.

Model functional behavior

II. Identify CFAs.

III. Assign real-time attributes to resources and activities participating in a CFA.

IV. Establish coordinating CFAs.

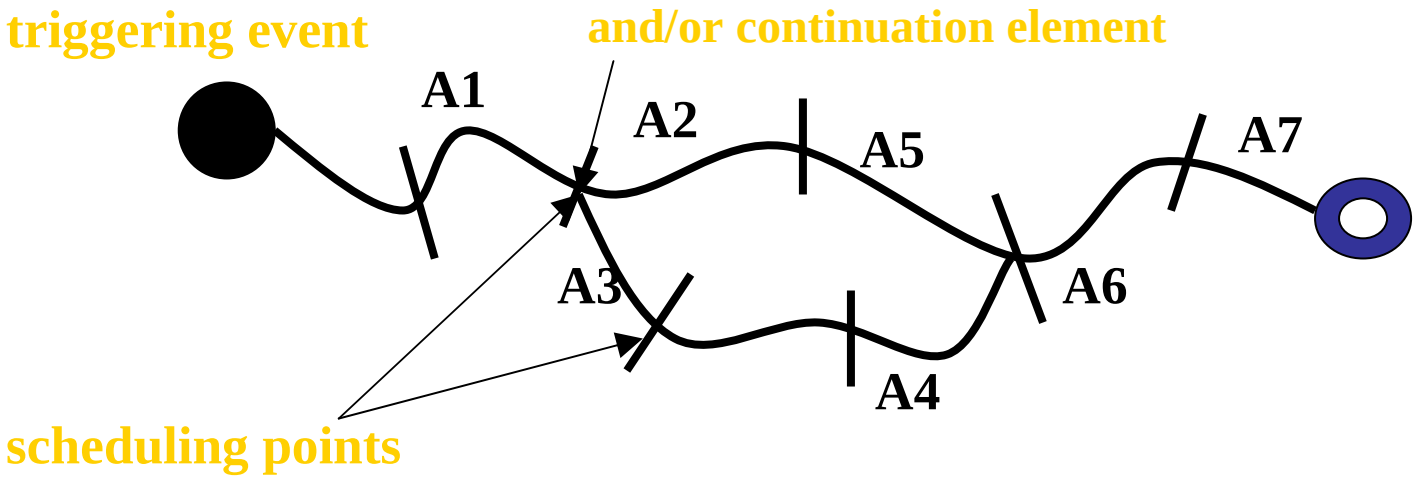
V. Allocate each activity to the component that implements it.

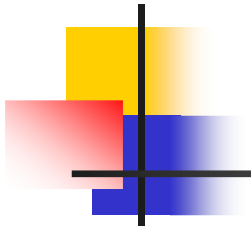
VI. Build PPOOA Dynamic View of the System.

CFA (Behavioral view)

DEFINITION

CFA means Causal Flow of Activities. Therefore, a CFA is a chain of activities that is triggered by an event.





CFA Building Elements

- **Event**: something that occurs in the system or in the environment and something to which the system must react to and handle.
- **Action**: Computation block where no decision of assigning resources is taken. In PPOOA it represents an operation, task to perform, resource usage, etc.
- **Operators** that allow branching, parallel execution, etc.

PPOOA Architecting Process

SELECT COORDINATION MECHANISMS

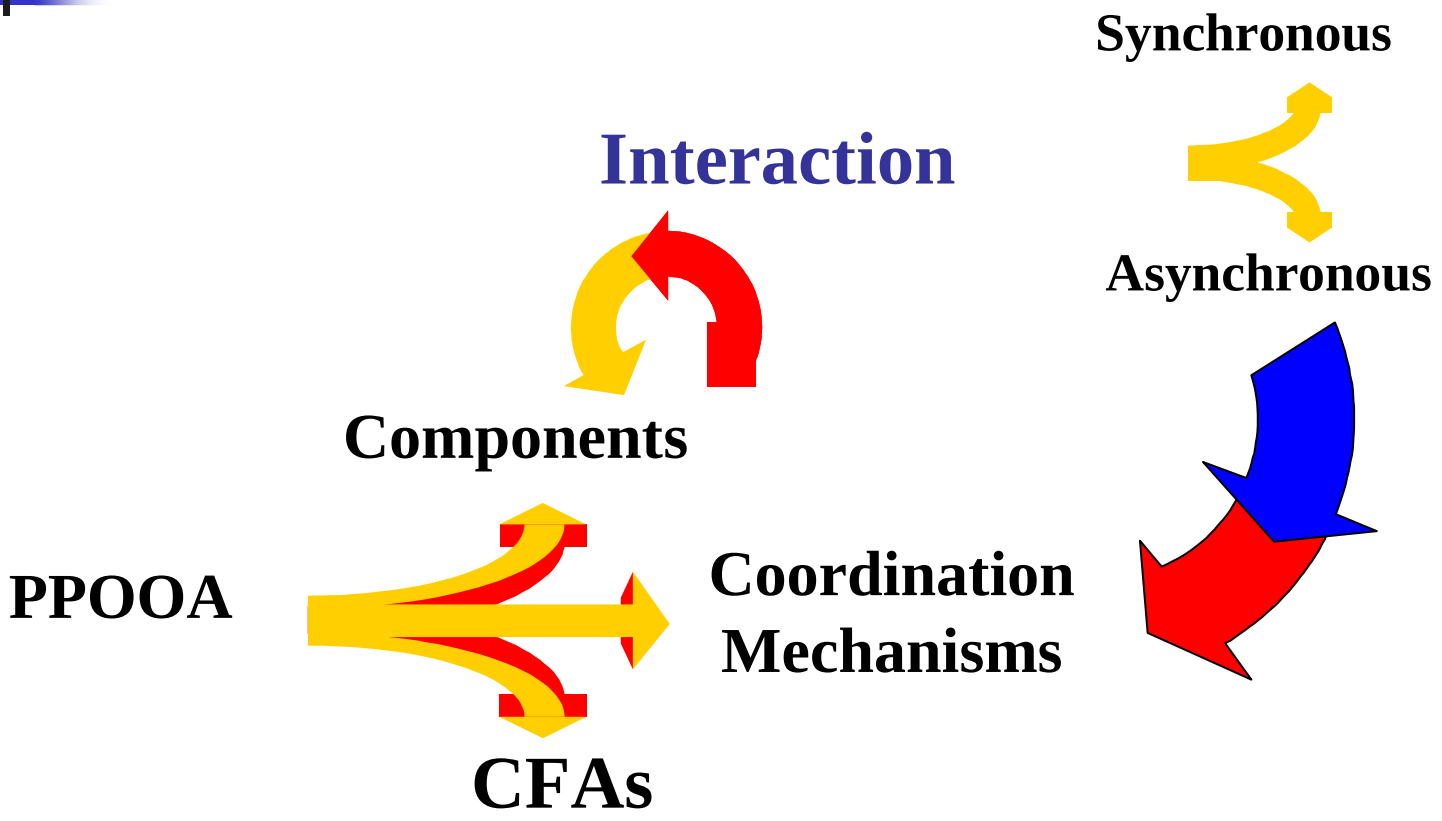
I. Discover the concurrency problem to solve.

II. Select the most suitable PPOOA coordination mechanism.

III. Assign real-time attributes to the coordination mechanism.

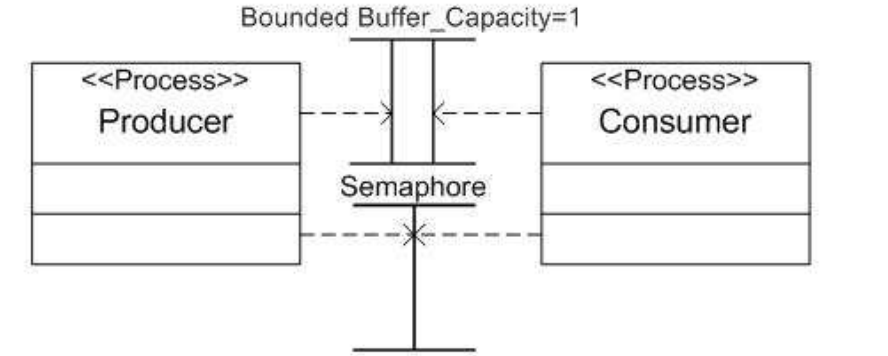
IV. Build the PPOOA Architecture Diagram of the System.

Interaction between Components (I)

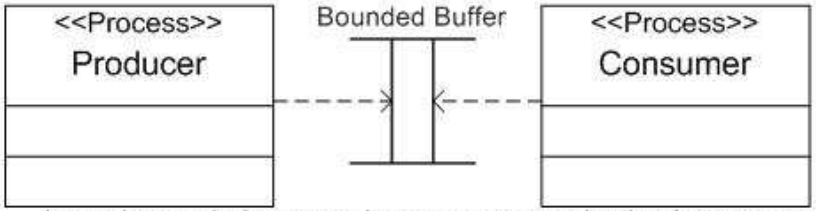


Interaction between Components (II)

Synchronous communication between passive components through operations

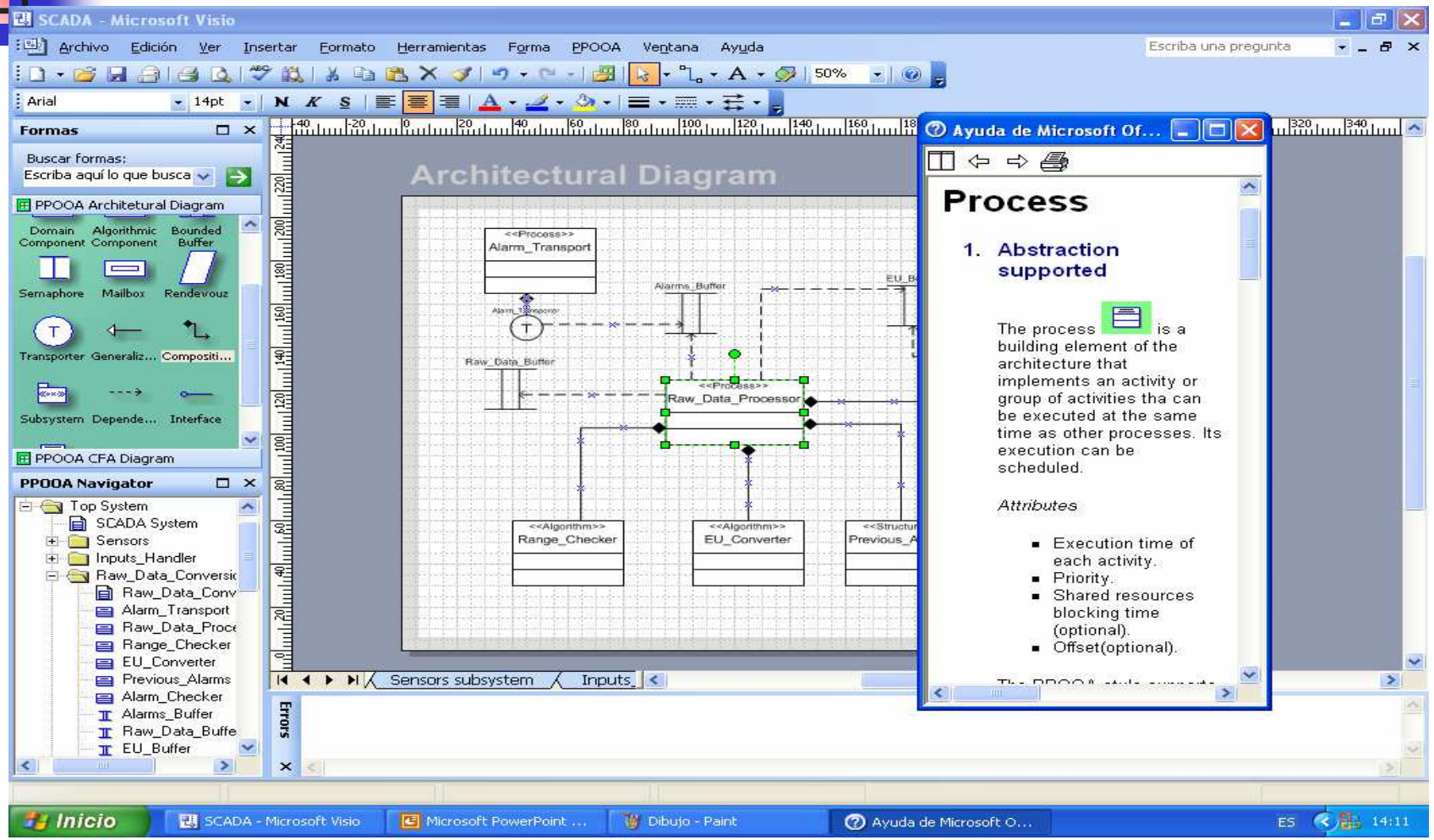


Tightly coupled message communication between active components without reply

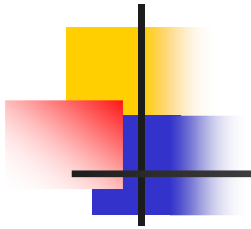


Loosely coupled or asynchronous communication between active components

PPOOA-Visio tool

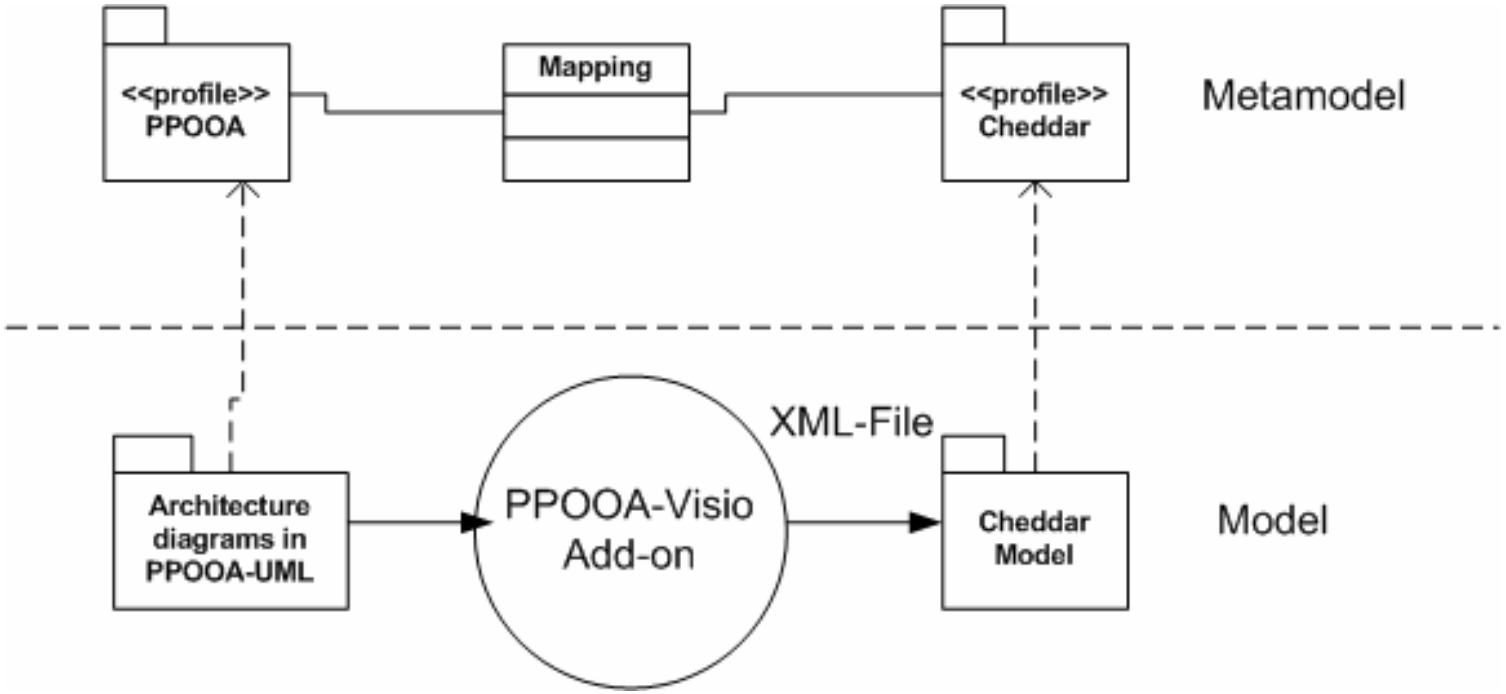


Architecture assessment: Cheddar

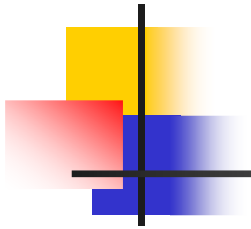


- Cheddar is a framework implemented in Ada by the University of Brest. It provides tools to check if a real-time system meets its temporal requirements.
- Cheddar provides **real-time feasibility tests** in the case of monoprocessor, multiprocessor and distributed systems.
  - The first feasibility test consists in comparing the processor **utilization factor** to a given bound.
  - The second feasibility test consists in comparing the **worst response time** of each system task with its deadline.
- Cheddar provides a **simulation engine** which allows the performance engineer to describe and run simulations of specific real-time systems

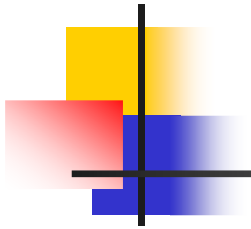
PPOOA-Cheddar implementation. Transforming architecture models



Working with PPOOA-Cheddar tools

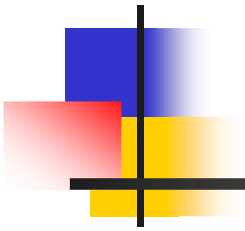


1. Create architecture models with PPOOA-Visio tool
2. Execute the **PPOOA-XML add-on**
3. The add-on automatically identifies the architecture building elements and their relations and generates an XML file of the architecture
4. The **XML file** is used as input to Cheddar
5. The engineer has to assign time parameters to the architecture elements
6. Cheddar runs the **simulation and schedulability feasibility tests**



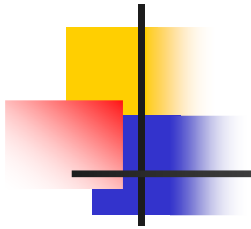
Performance evaluation results

- Cheddar offers a **simulation engine** which allows the performance engineer to describe and run simulations of the architected system. When the simulation is executed, Cheddar determines for each system task and during the simulation time:
  - The number of task preemptions
  - The number of context switches
  - The blocking times
  - The missed deadlines.
- The Cheddar tool offers **real-time feasibility checks** based on scheduling theory for example **"Rate Monotonic Analysis" (RMA),** without the need of running the system. Cheddar indicates if the task set is schedulable.



Deadlock risk assessment

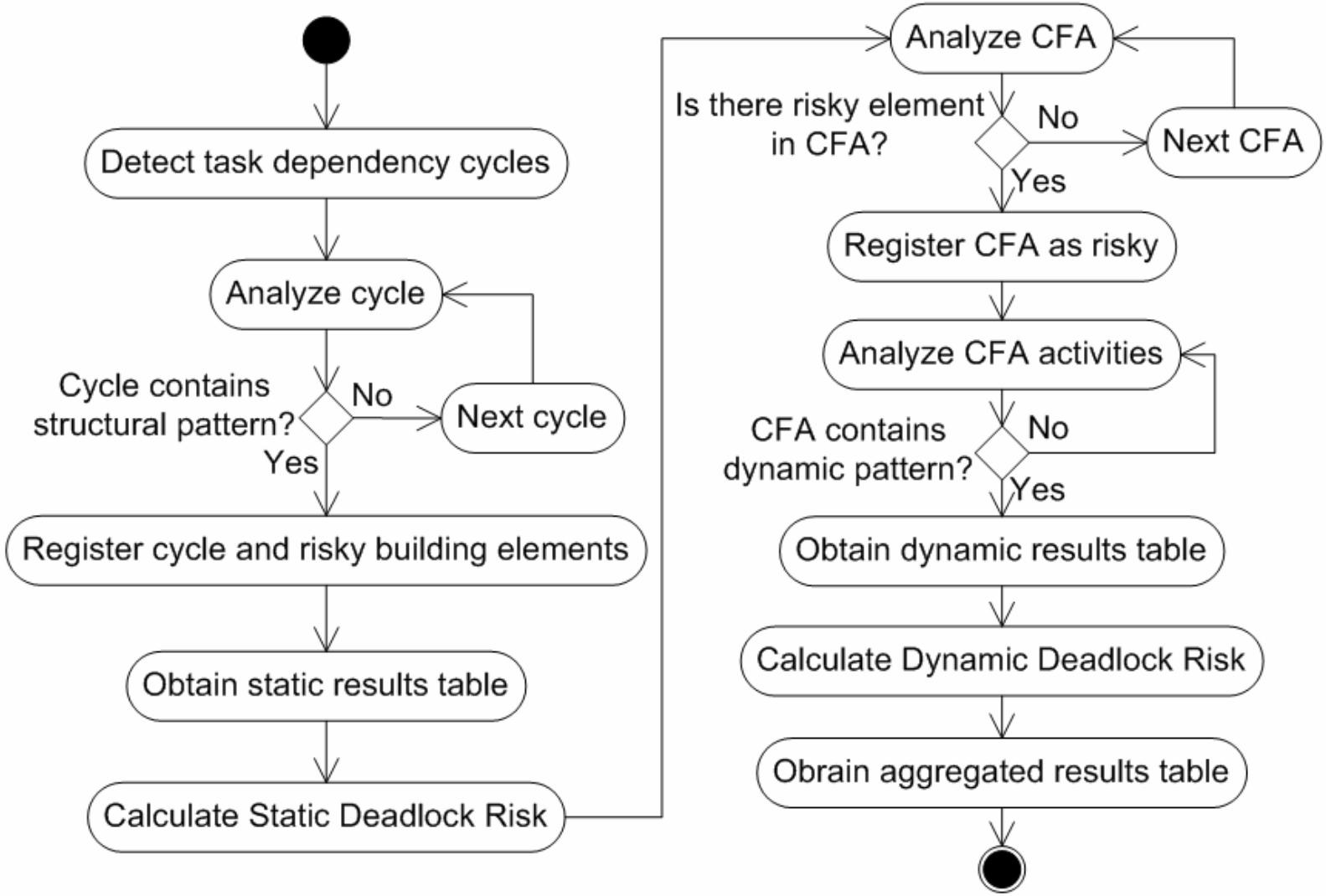
A method and a tool for evaluating the deadlock risk of an architecture



Deadlock

- Deadlock is an execution-time problem potentially catastrophic in **safety- and mission-critical systems**
- It is difficult to detect in design-time and most of the times transparent to traditional testing procedures because it is very unlikely
- Its correction may be complex and costly at late development phases
- It can be treated with very few temporal information at **early stages of conceptual design of software architecture**
- It is possible to identify the **intrinsic deadlock risk** of a model before the corresponding model is created

Deadlock assessment (A. Monzón thesis 2010)



Deadlock assessment report

Section	Table
Static Deadlock Patterns (Initial Model)	
Dynamic Deadlock Patterns (Initial Model)	
Static Deadlock Patterns (Alternative Model)	
Dynamic Deadlock Patterns (Alternative Model)	
Design Trade-off	

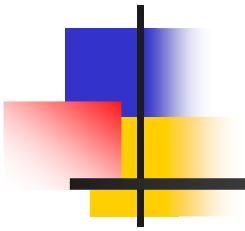
ID	Building Elements	Deadlock Patterns	Deadlock Risk?
1	B_New_Conditions -> PP_Update_A/C_Position -> Semaphore 3 -> PP_Manager_Autotuning_Plan ->	B_New_Conditions, PP_Update_A/C_Position, Semaphore 3, PP_Manager_Autotuning_Plan,	Yes
2	AP_Update_Waypoint_Passed -> B_New_Conditions -> PP_Update_A/C_Position -> Semaphore 3 ->	AP_Update_Waypoint_Passed, B_New_Conditions, PP_Update_A/C_Position, Semaphore 3,	Yes

ID	Name	Task-Buffer Patterns	Task-Semaphore-Buffer Patterns	Task-Semaphore-Resource Patterns	NFE
1	CFA01	None	Analyse waypoint(AP_Update_Waypoint_Passed) -> Acquire(Semaphore 3) -> Send new condition (B_New_Conditions)	None	0

ID	Building Elements	Deadlock Patterns	Deadlock Risk?
1	B_New_Conditions -> PP_Update_A/C_Position -> Semaphore 3 -> PP_Manager_Autotuning_Plan ->	B_New_Conditions, PP_Update_A/C_Position, Semaphore 3, PP_Manager_Autotuning_Plan,	Yes
2	AP_Update_Waypoint_Passed -> B_New_Conditions -> PP_Update_A/C_Position -> Semaphore 3 ->	AP_Update_Waypoint_Passed, B_New_Conditions, PP_Update_A/C_Position, Semaphore 3,	Yes

ID	Name	Task-Buffer Patterns	Task-Semaphore-Buffer Patterns	Task-Semaphore-Resource Patterns	NFE
1	CFA01	None	Analyse waypoint(AP_Update_Waypoint_Passed) -> Acquire(Semaphore 3) -> Send new condition (B_New_Conditions)	None	0

	Initial Model	Model Alternative 1
Number of Elements	34	35 (2.9%)
Number of Arcs	36	34 (-5.6%)
Number of Cycles	7	1 (-85.7%)
SDR	6	0 (-100.0%)
Number of Risky Elements	6	0 (-100.0%)
Number of Static Patterns	6	0 (-100.0%)
Number of Dynamic Patterns	4	0 (-100.0%)
Number of Parallel Flows of Activities	1	0 (-100.0%)
Number of Activities and Decision Points	44	36 (-18.2%)
Number of Risky Activities and Decision Points	16	0 (-100.0%)



Example

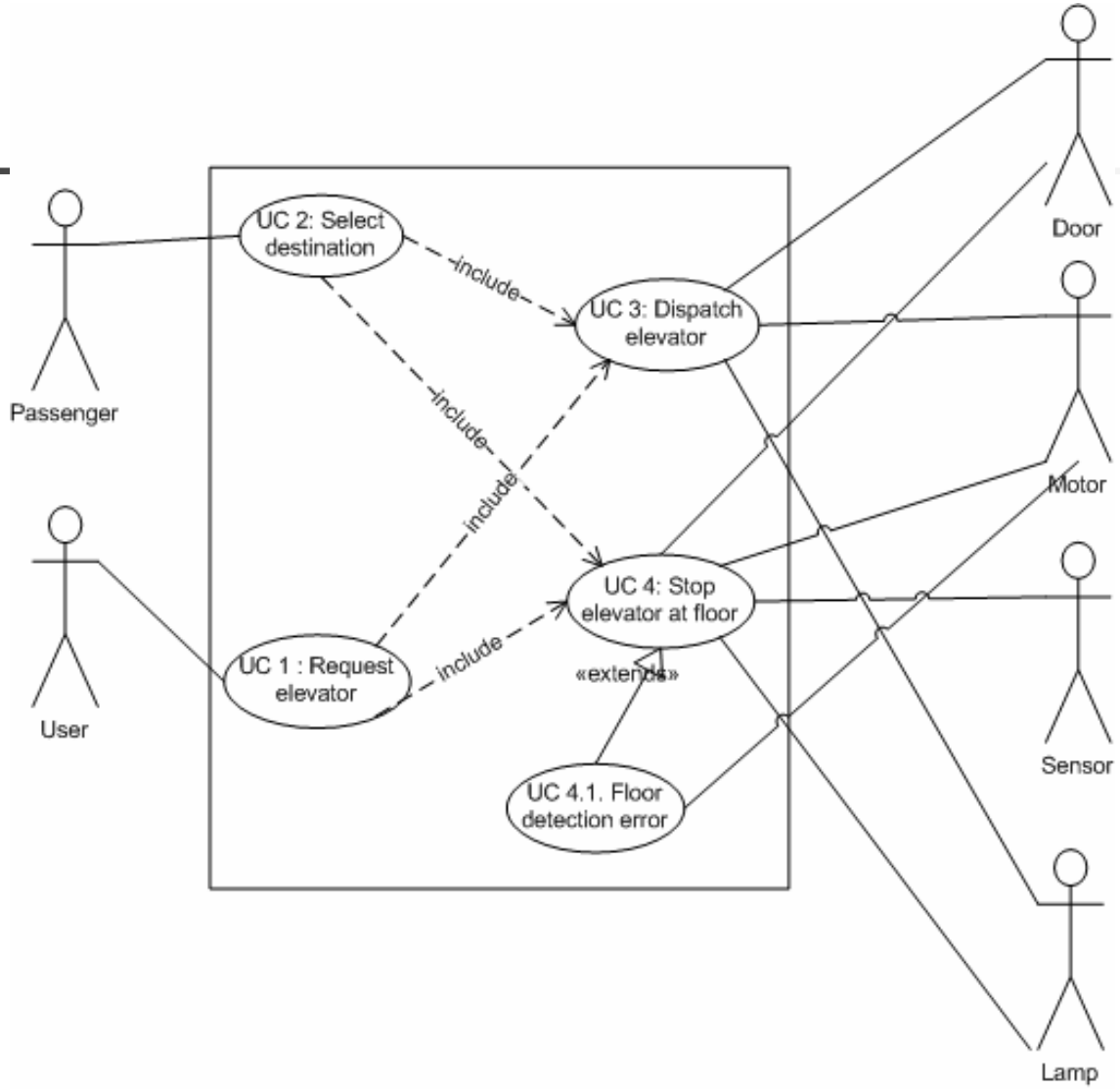
Elevator Control System

Elevator Control System (ECS)

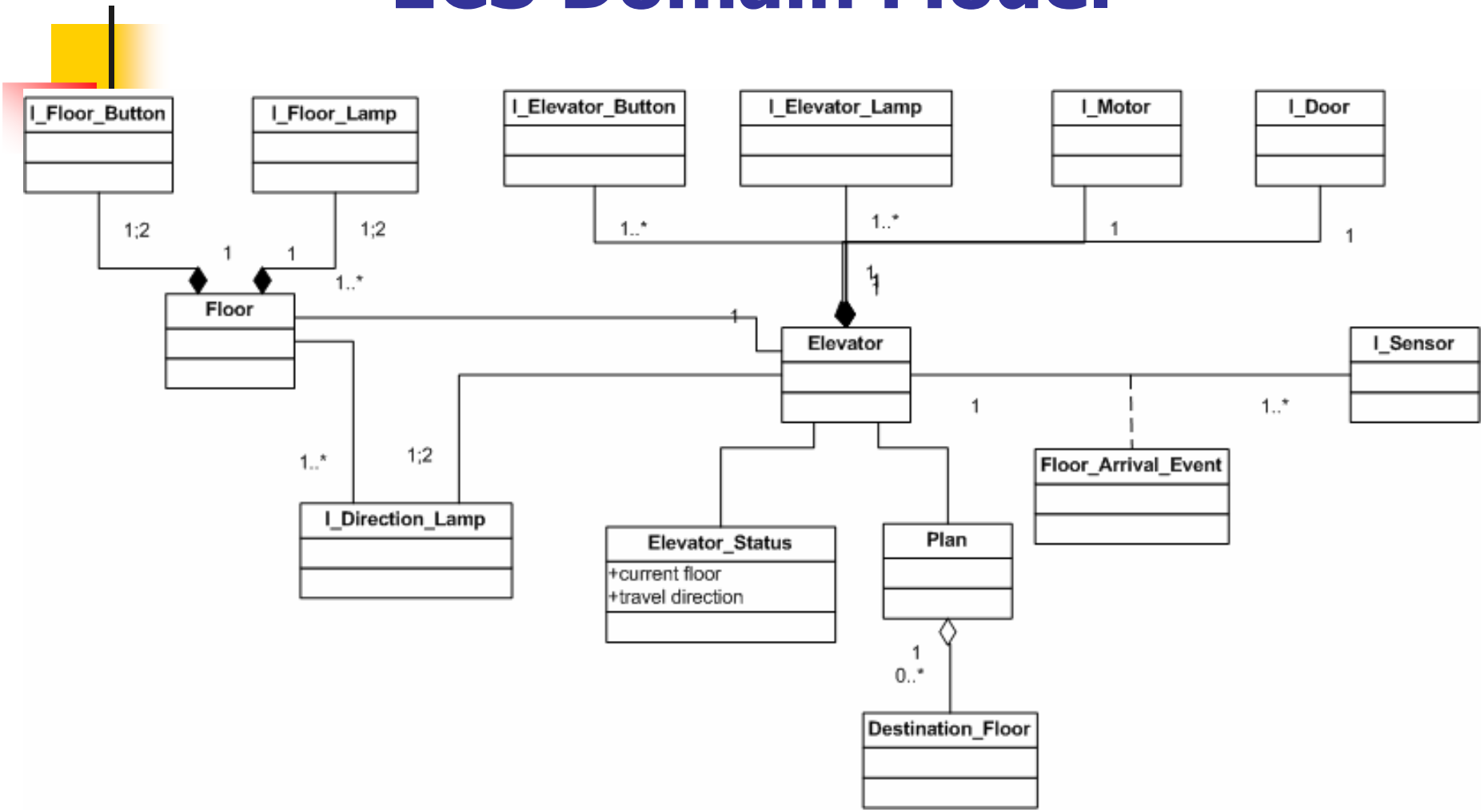
The system controls a single elevator which responds to requests from elevator passengers and users at various floors. Based on these requests and the information received from floor arrival sensors, the system builds a plan to control the motion and stops of the elevator.

- We consider a ten floor building, so for the house elevator, there are:
  - 10 arrival sensors, one at each floor in the elevator shaft to detect the arrival of the elevator at the corresponding floor.
  - 10 elevator buttons. An elevator passenger presses a button to select a destination.
  - The elevator motor controlled by commands to move up, move down and stop.
  - The elevator door controlled by commands to open and close it.
  - Up and down floor buttons. A user in a floor of the house presses a floor button to request the elevator.
  - A corresponding pair of floor lamps which indicate the directions which have been requested.

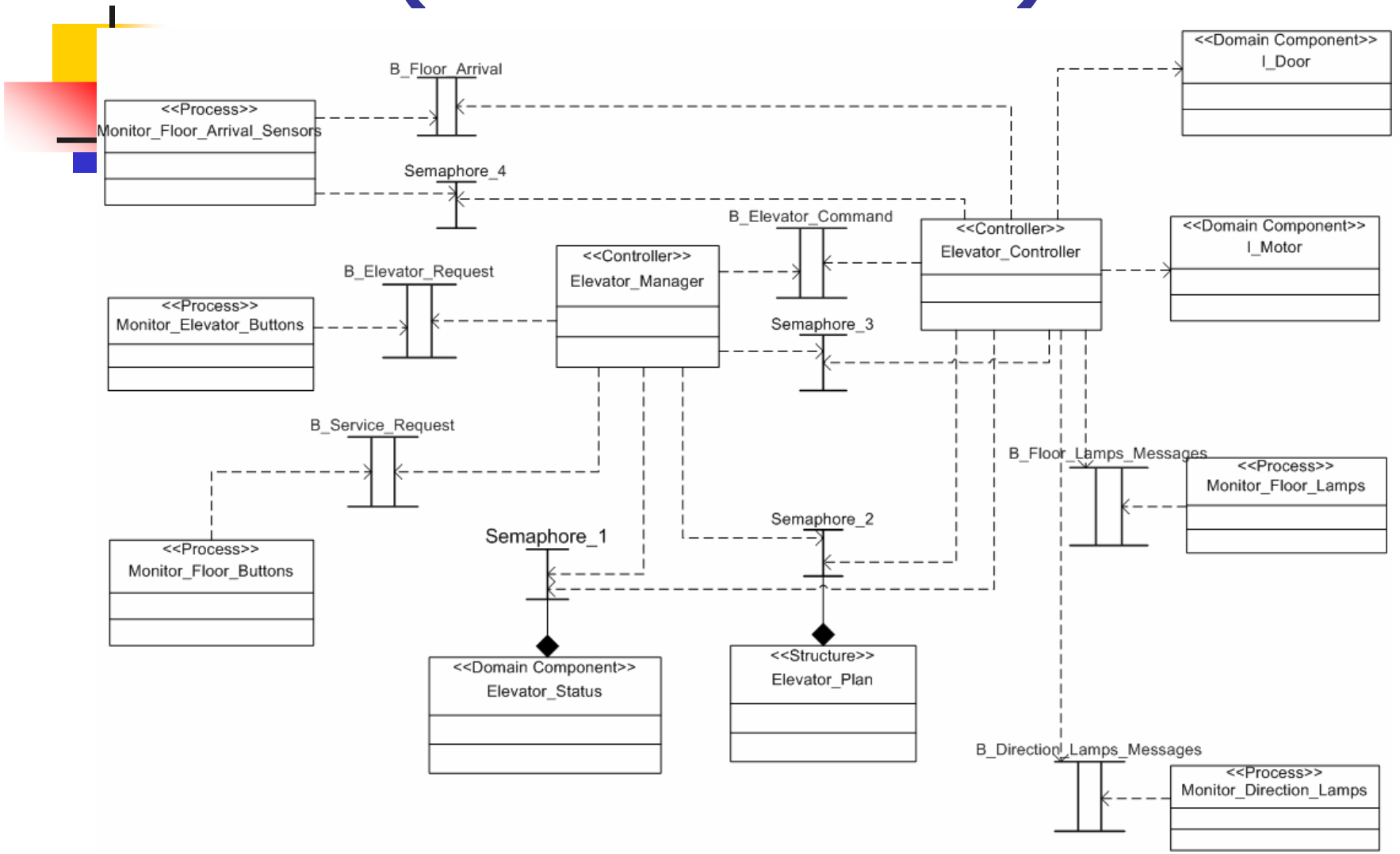
ECS use cases



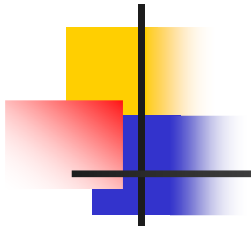
ECS Domain Model



ECS Architecture Diagram (Structural view)



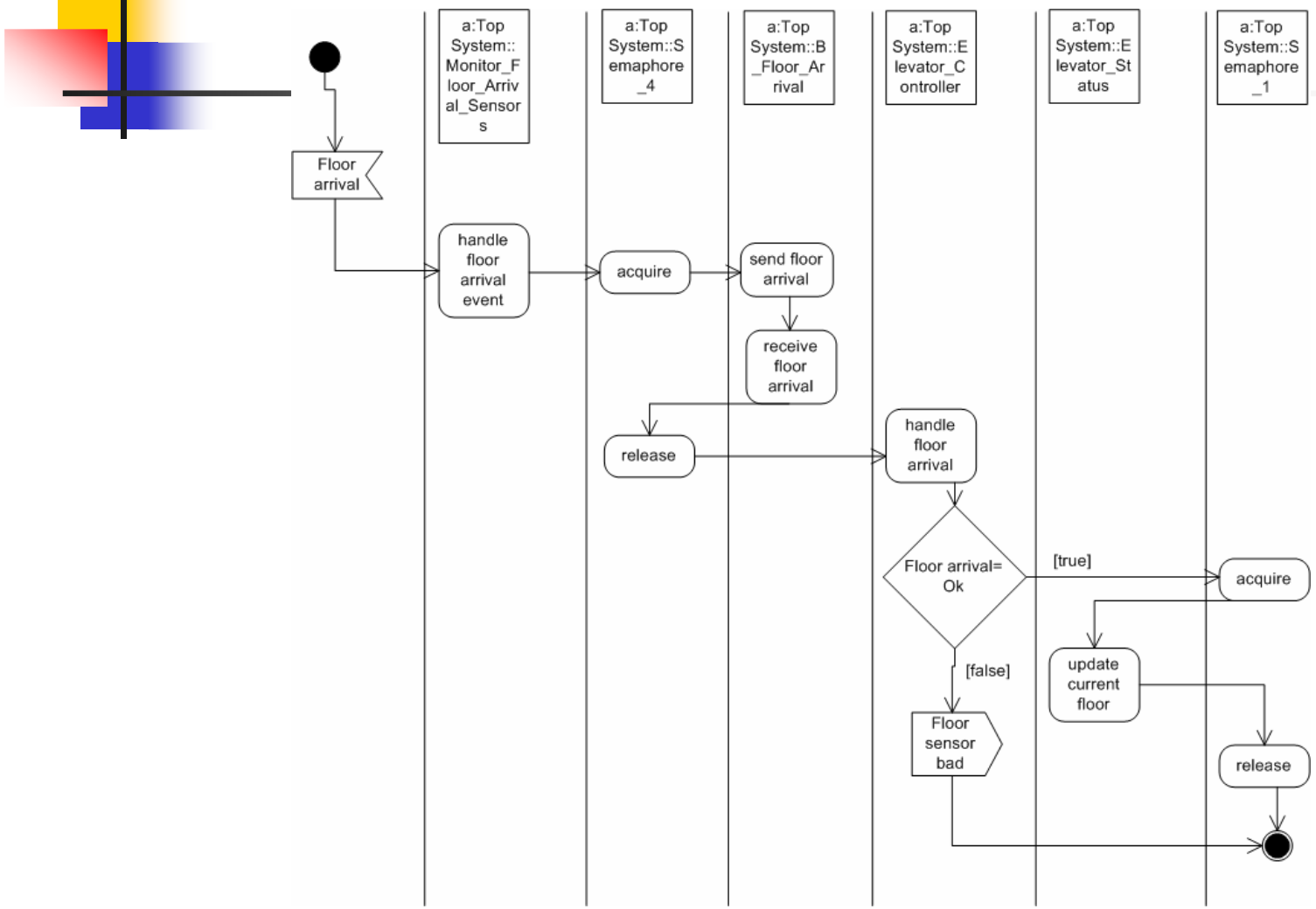
System responses of the ECS



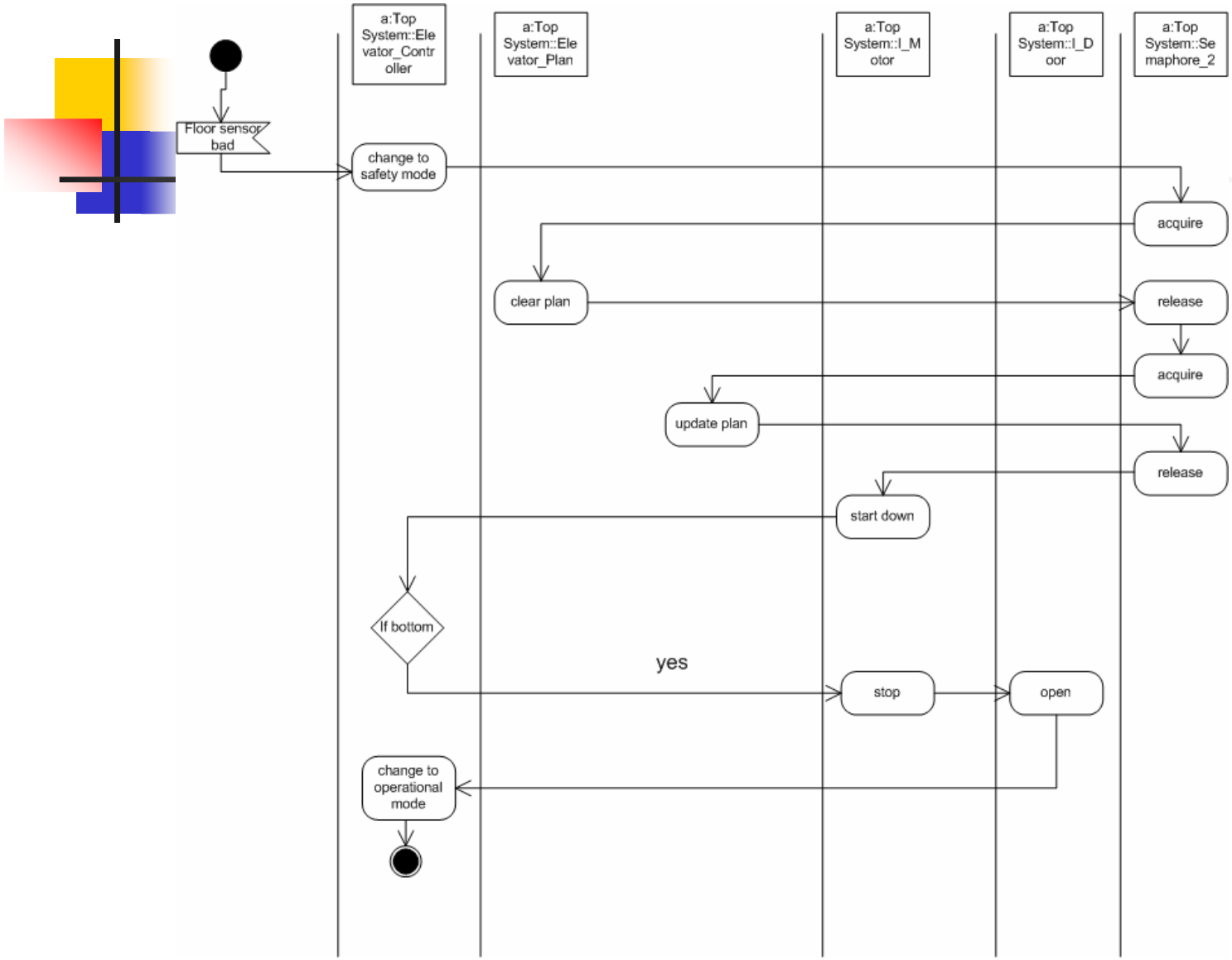
For the elevator control system we identified the following system responses and represented them as CFAs:

- CFA 1: Request floor from elevator
- CFA 2: Request elevator from floor
- CFA 3: Update current floor
- CFA 4: Stop elevator at floor
- CFA 5: Dispatch elevator to next destination
- CFA 6: Bad floor arrival sensor event

CFA 3: Update Current Floor



CFA 6: Bad floor arrival sensor event

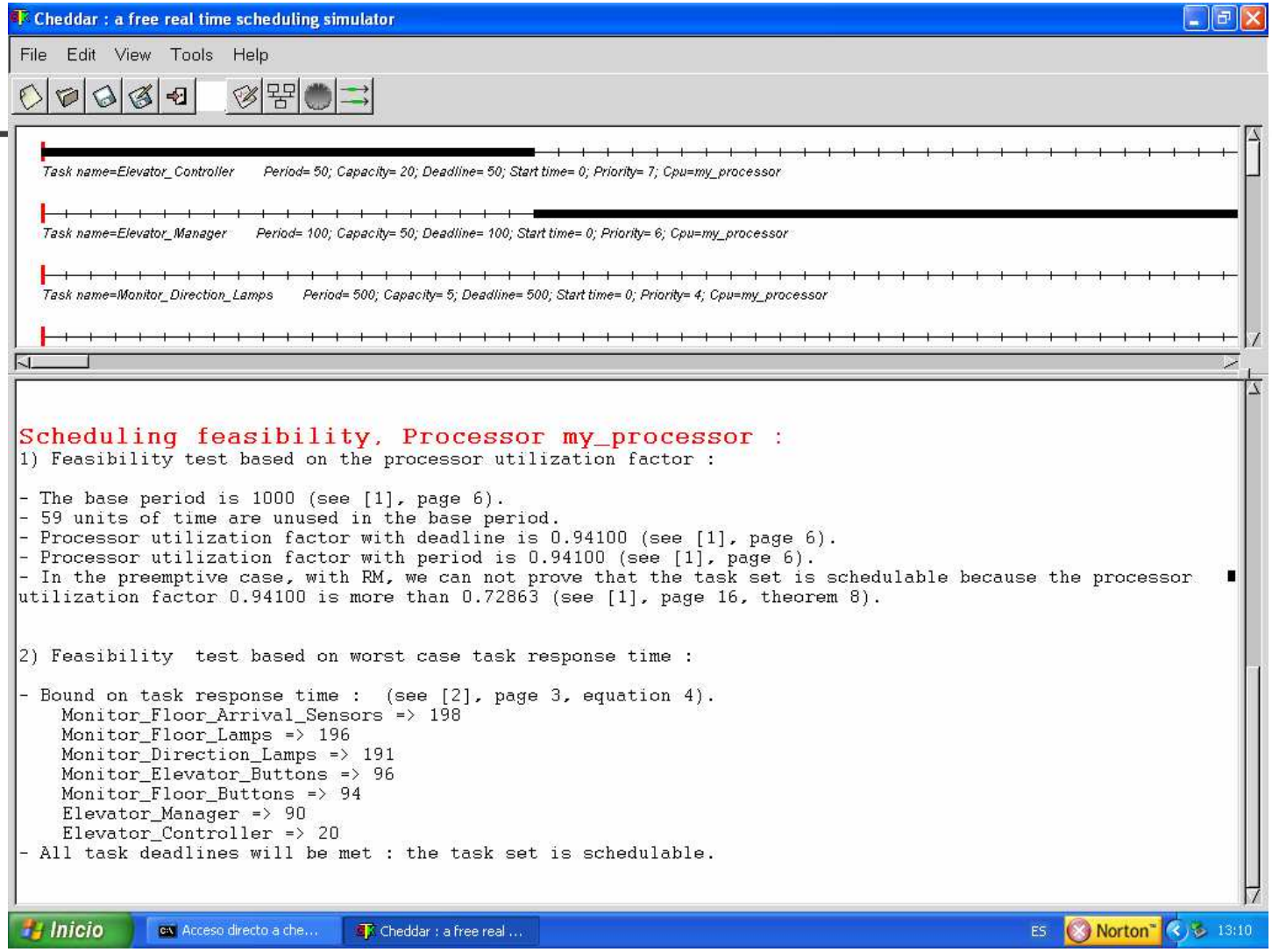


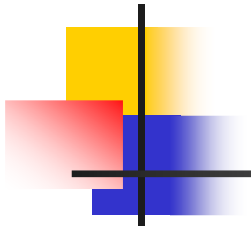
Numeric inputs to Cheddar

- The software engineer had to **estimate the execution period and capacity of the system tasks**, see table below.
- Also he or she had to estimate the **time instant each task begins using each buffer and resource**. The CFAs or system responses are the inputs used for this usage estimation.

Task name	Type	Capacity	Period	Deadline
Elevator_Controller	periodic	20	50	50
Elevator_Manager	periodic	50	100	100
Monitor_Floor_Lamps	periodic	5	1000	1000
Monitor_Elevator_Buttons	periodic	2	500	500
Monitor_Direction_Lamps	periodic	5	500	500
Monitor_Floor_Arr_Sensors	periodic	2	1000	1000
Monitor_Floor_Buttons	periodic	4	200	200

ECS Performance Evaluation



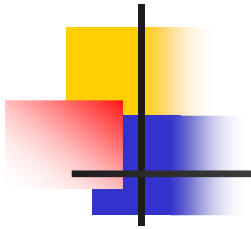


Experiences

- Unmanned underwater vehicle developed by Qinetic (UK)
- Autotunning function of the Airbus A400M
- Space systems developed by Artal (France) and TCP (Spain)
- Student projects at the ETSII-UPM (Madrid)

Users and some links

- PPOOA free Visio add on and stencils have been requested by software architects from USA, Spain, Germany, France, Finland and other countries
- During 2011, PPOOA web site had **19863** sessions and **43149** publications downloads.
- Tutorials and/or mentoring has been given to engineers from Airbus Military, Audi, Indra, Eurocopter, Isdefe, Optimitive and Polar.
- Additional information and publications can be found at **www.ppooa.com.es**
- ISE&PPOOA is included in the OMG wiki of MBSE methodologies: **http://www.omgwiki.org/MBSE/doku.php?id=mbse:methodology**
- Cheddar tool: **http://beru.univ-brest.fr/~singhoff/cheddar/contribs/examples_of_use/00readme.html**



To Conclude

- Development of a system can be envisioned as an amalgation of three dimensions operational, system and software.
  - The **operational dimension** is concerned about the operational issues and the overall system structure.
  - The **system dimension** is concerned about the overall functional and technical dimensions of the system,
  - and the **software dimension** is concerned about the software items contained in the system.
- The ISE&PPOOA process **integrates** the **system and software engineering dimensions** using a common behavioural representation based mainly on SysML/UML **activity diagrams** and using the **responsibilities** concept and the domain model for bridging the gap between the system and software dimensions.
- The process combines the **model based systems engineering** paradigm with some classical systems engineering best practices such as the **N^2 charts** for the interfaces, and textual descriptions in tabular form that complement the information presented in the **SysML** models