AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

Architecture & Design of Embedded Real-Time Systems (TI-AREM)

Architecture and UML
(BPD. chapter 2)

Version: 29-1-2015

Agenda

1. Introduction to SW architecture
2. Architecture and UML
3. Architectural Views

1. Introduction to Architecture (1)

Opera House in Sydney (J.Utzon)

- Cost $AU 102,000,000 to build.
- Includes 1000 rooms.
- Is 185 m long and 120 m wide.
- Has 2194 pre-cast concrete sections as its roof.
- Has roof sections weighing up to 15 t.
- Has roof sections held together by 350 kms of tensioned steel cable.
- Has over 1 million tiles on the roof.
- Uses 6225 square metres of glass and 645 kilometres of electric cable.
- Was designed by Danish architect Jørn Utzon.
- Was opened 20 October 1973.

Introduction to Architecture (2)

Grand Arche Paris (Spreckelsen)

Finished in 1989,
200 years
after the revolution !

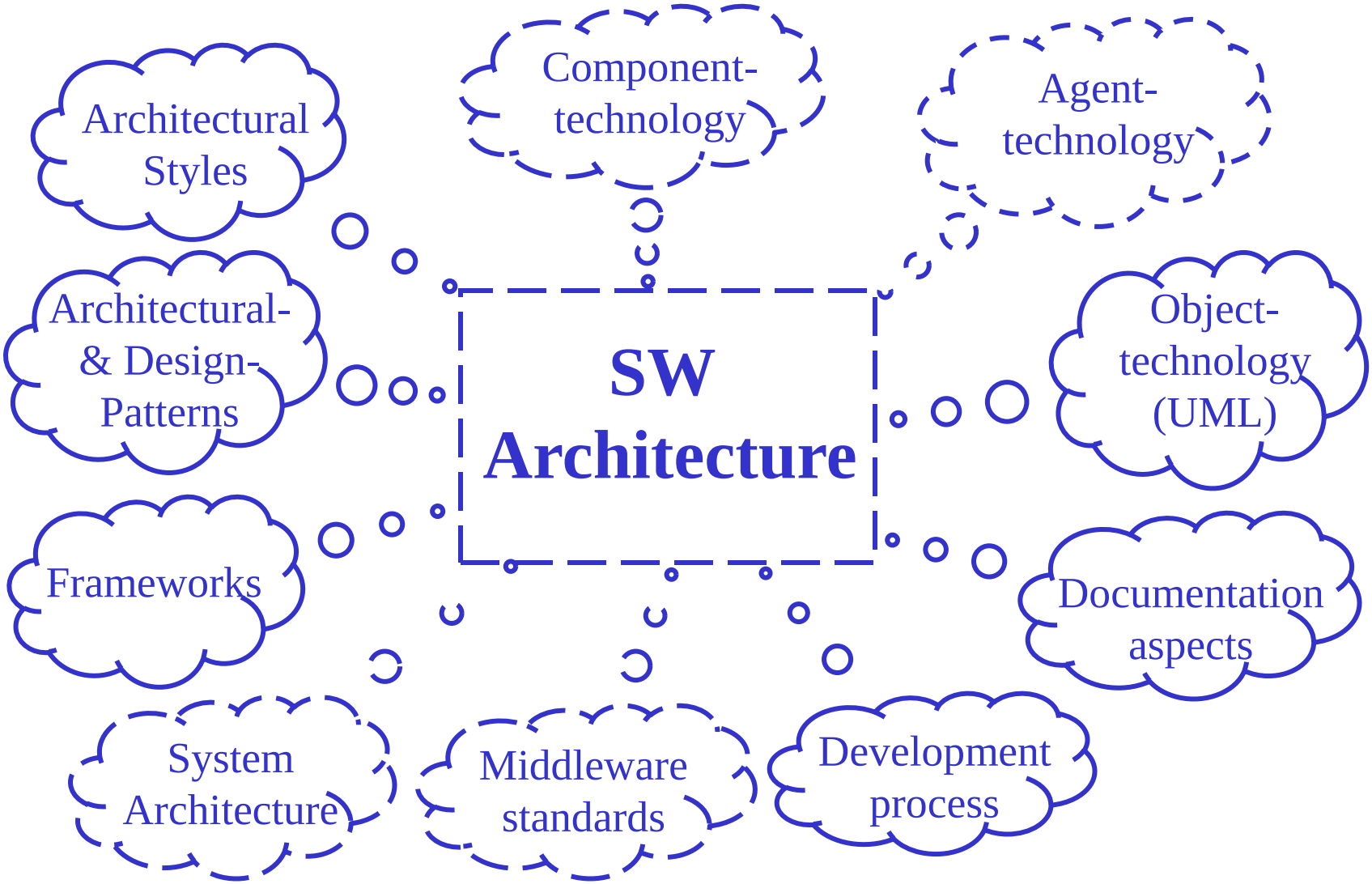
sculpture:
Joan Mirós

SW Architecture Definitions

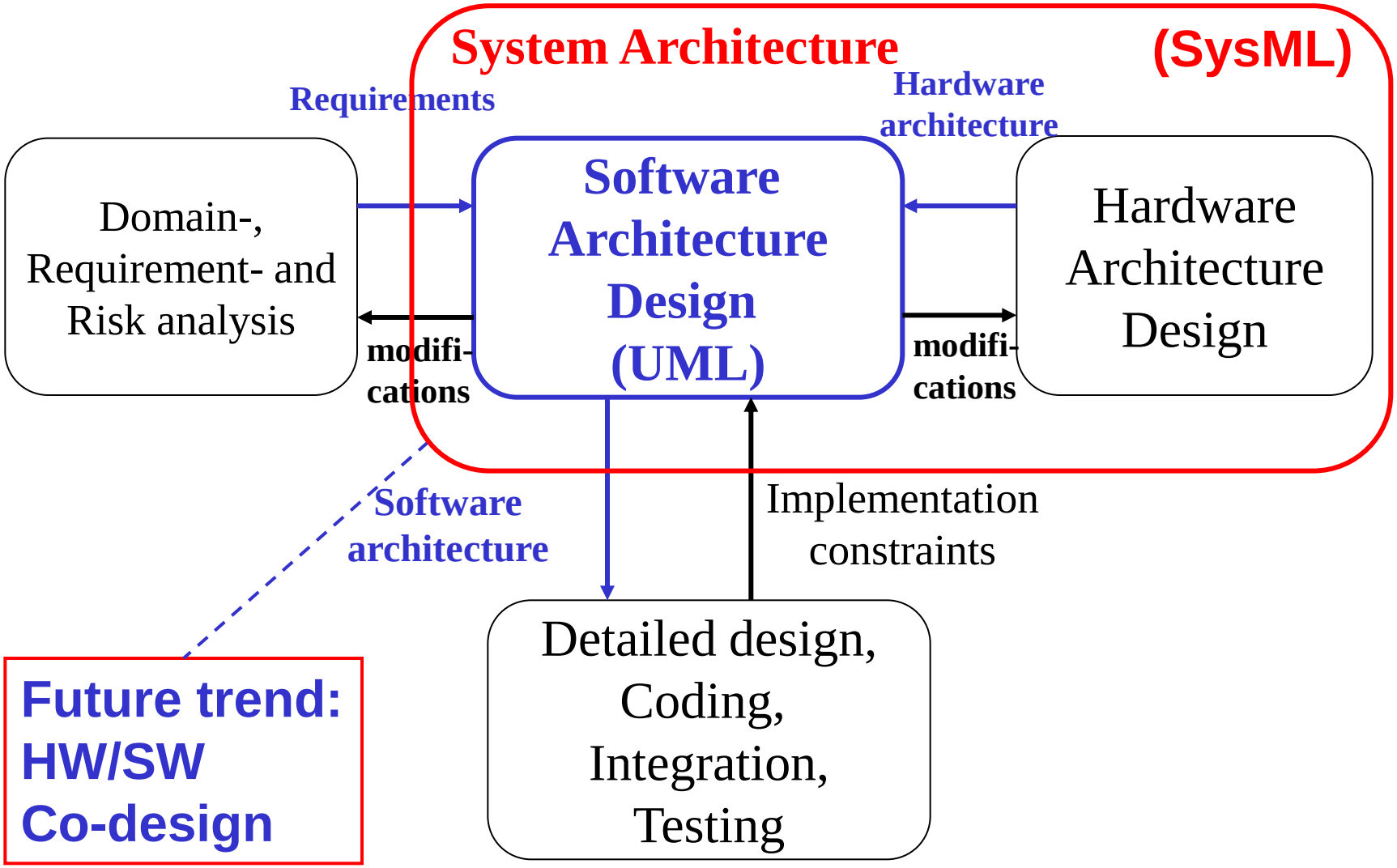
Examples on SW architecture definitions:

- *”The software architecture of a program or computing system is the **structure** or structures of the system, which comprise software **components**, the externally visible properties of those components and the **relationships** among them”*
  - *Ref: Software Architecture in Practice,* [Bass98]
- *”is a set of concepts and design decisions about the structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architectural significant explicit functional and quality requirements and implicit requirements of the product family, the problem and the solution domains”*
  - *Ref: Software Architecture for Product families*, [Jazayeri2000]

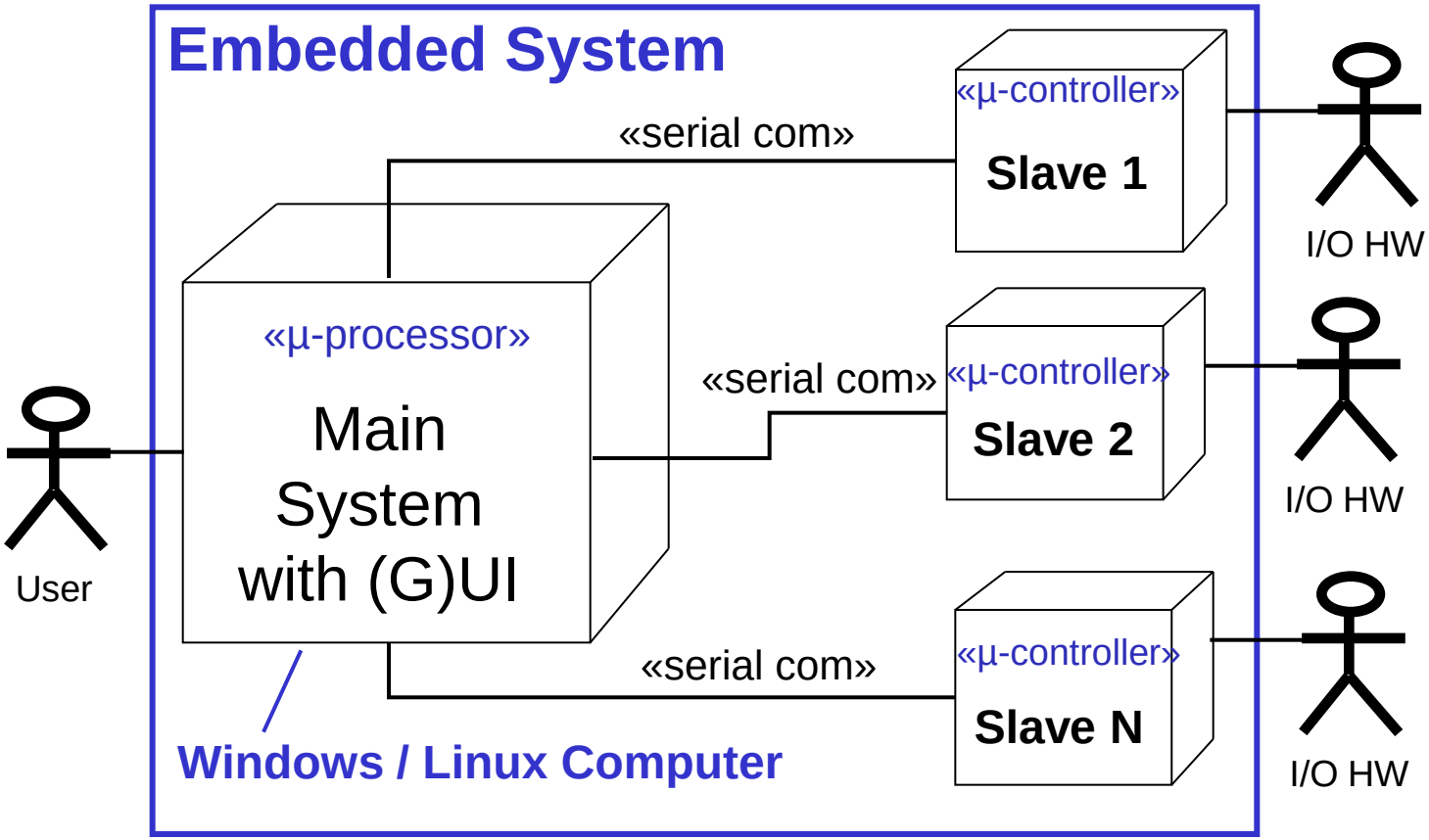
Context for SW Architecture



SW Architecture in relation to related System Development Activities

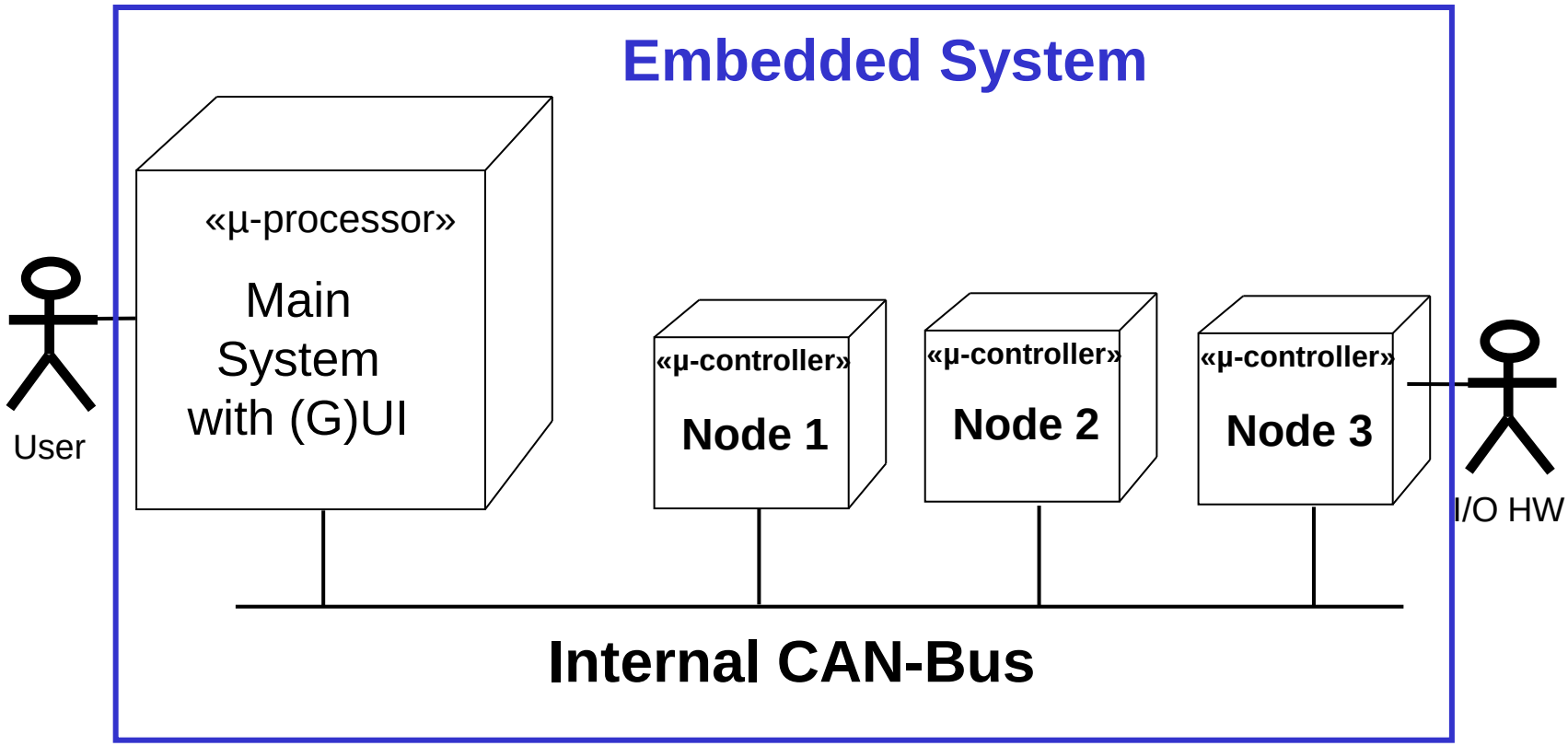


HW/SW Architecture – Example 1



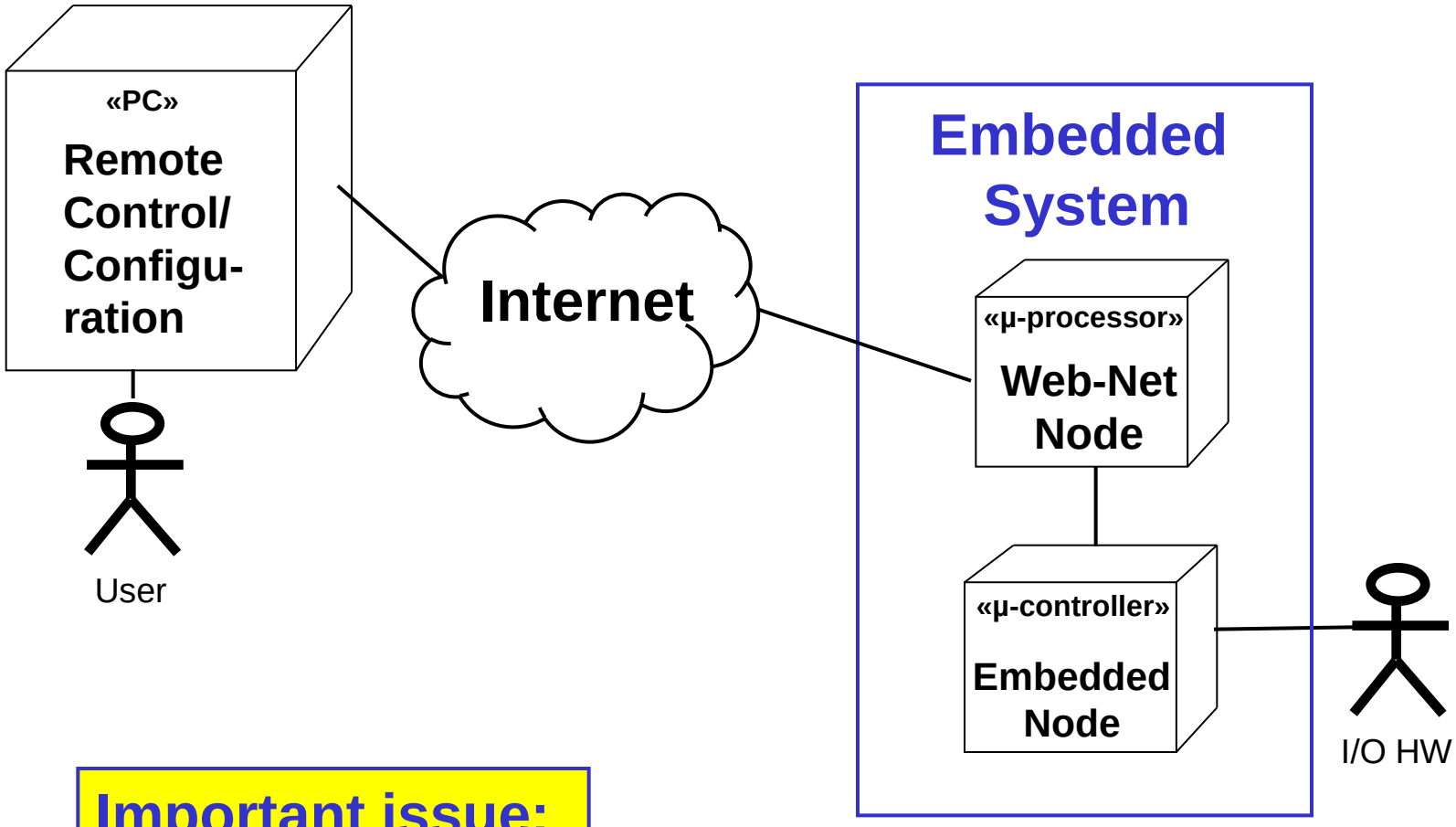
Typical HW Architecture for Larger Embedded Systems

HW/SW Architecture – Example 2



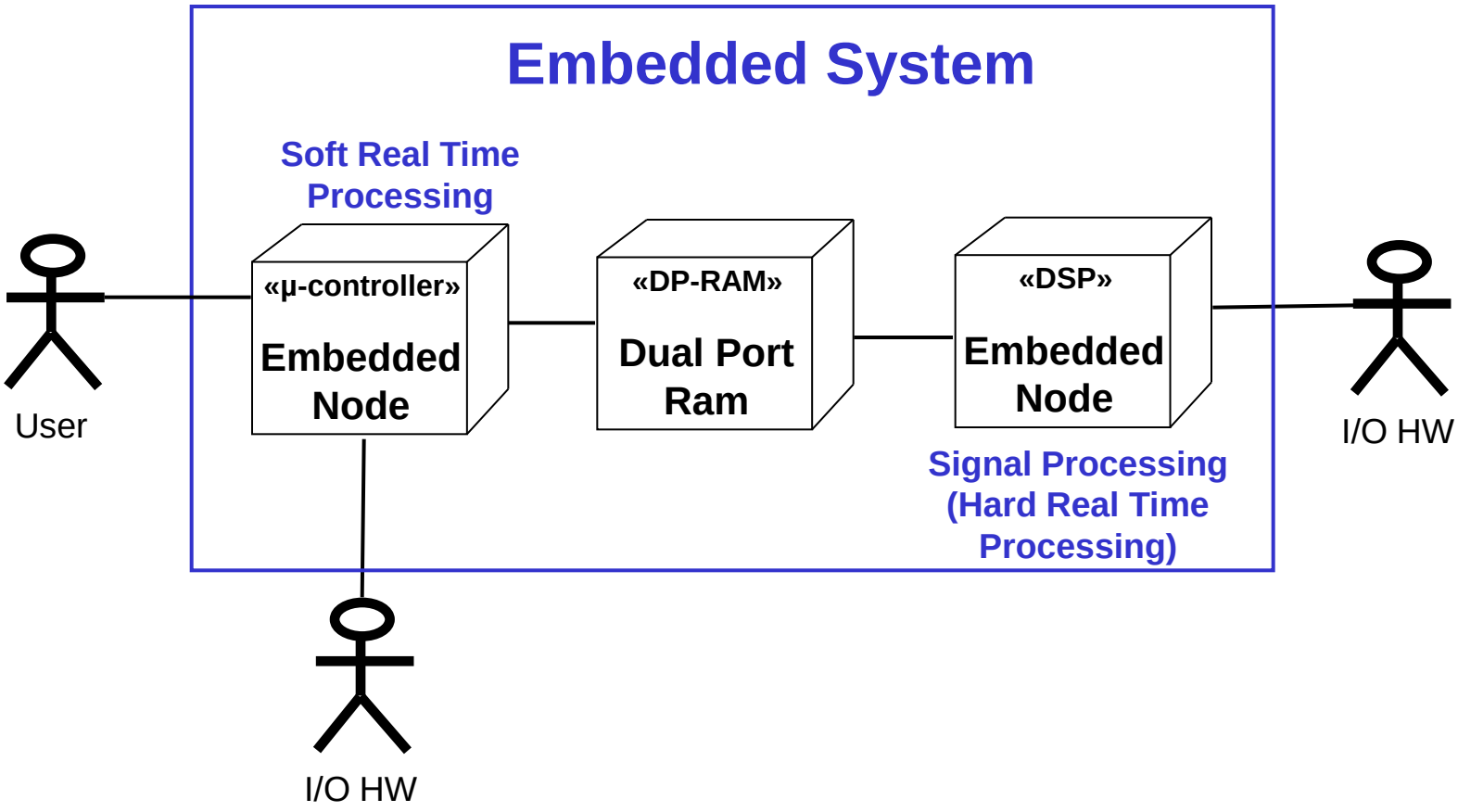
Modular system
– communication for free

HW/SW Architecture – Example 3

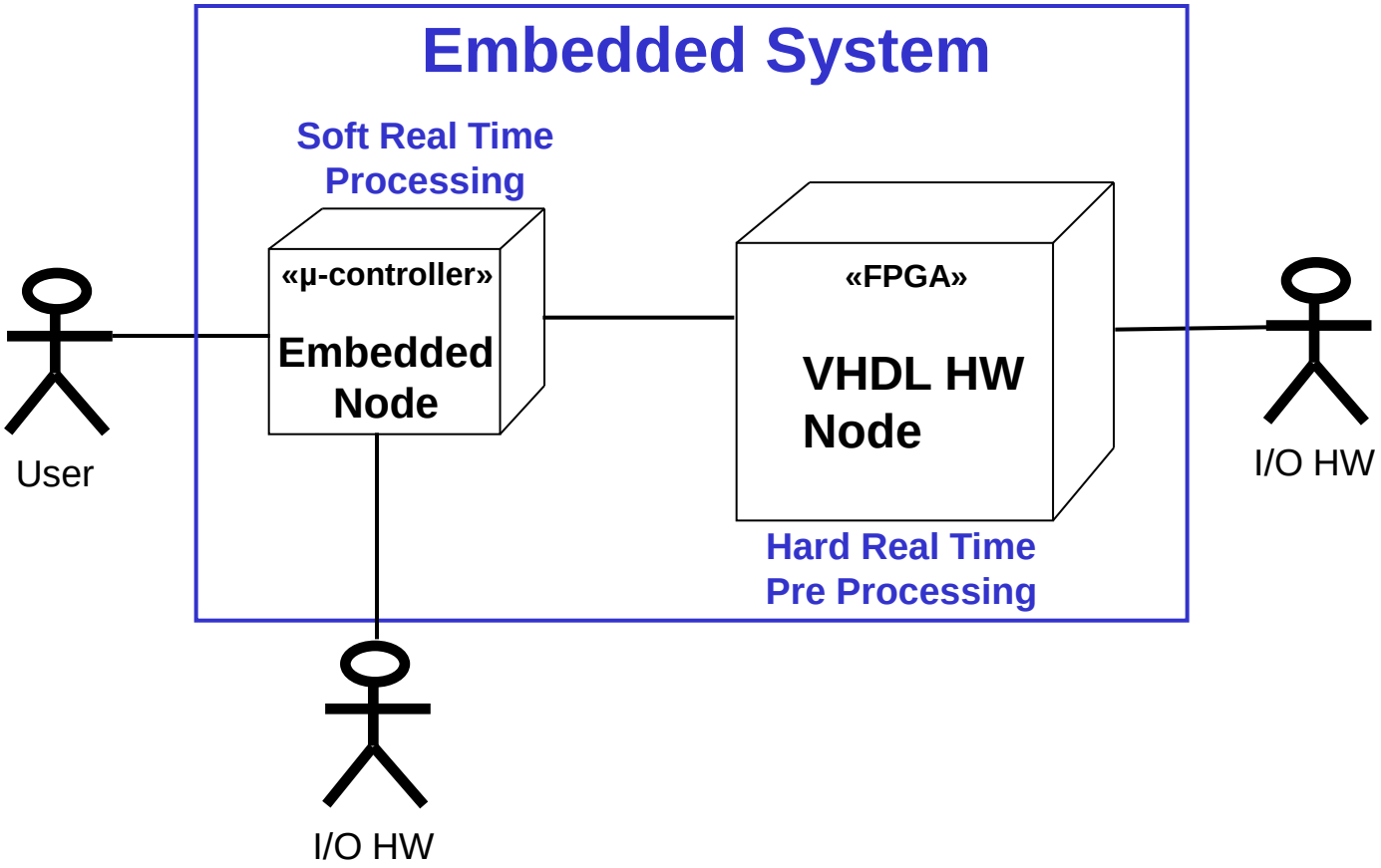


Important issue: Security

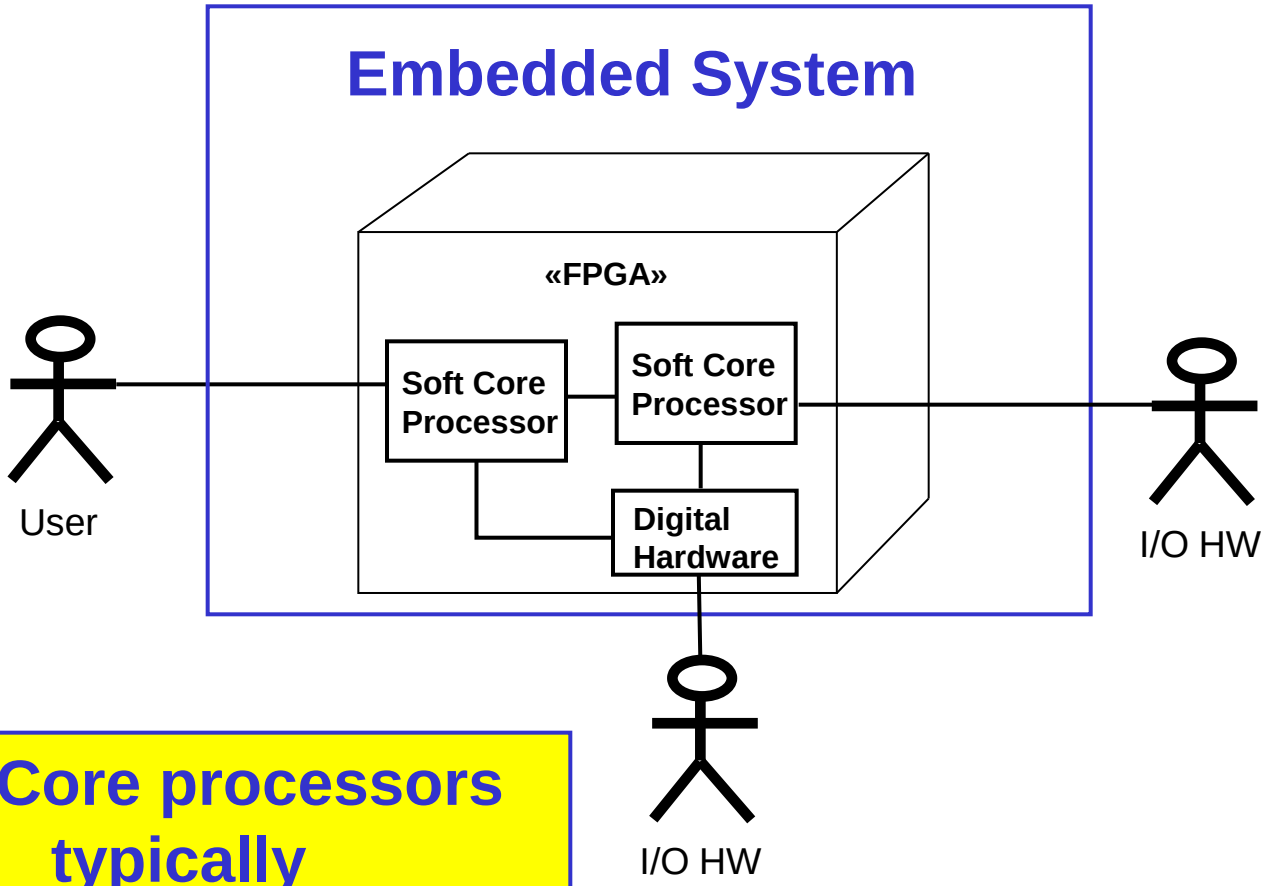
HW/SW Architecture – Example 4



HW/SW Architecture – Example 5

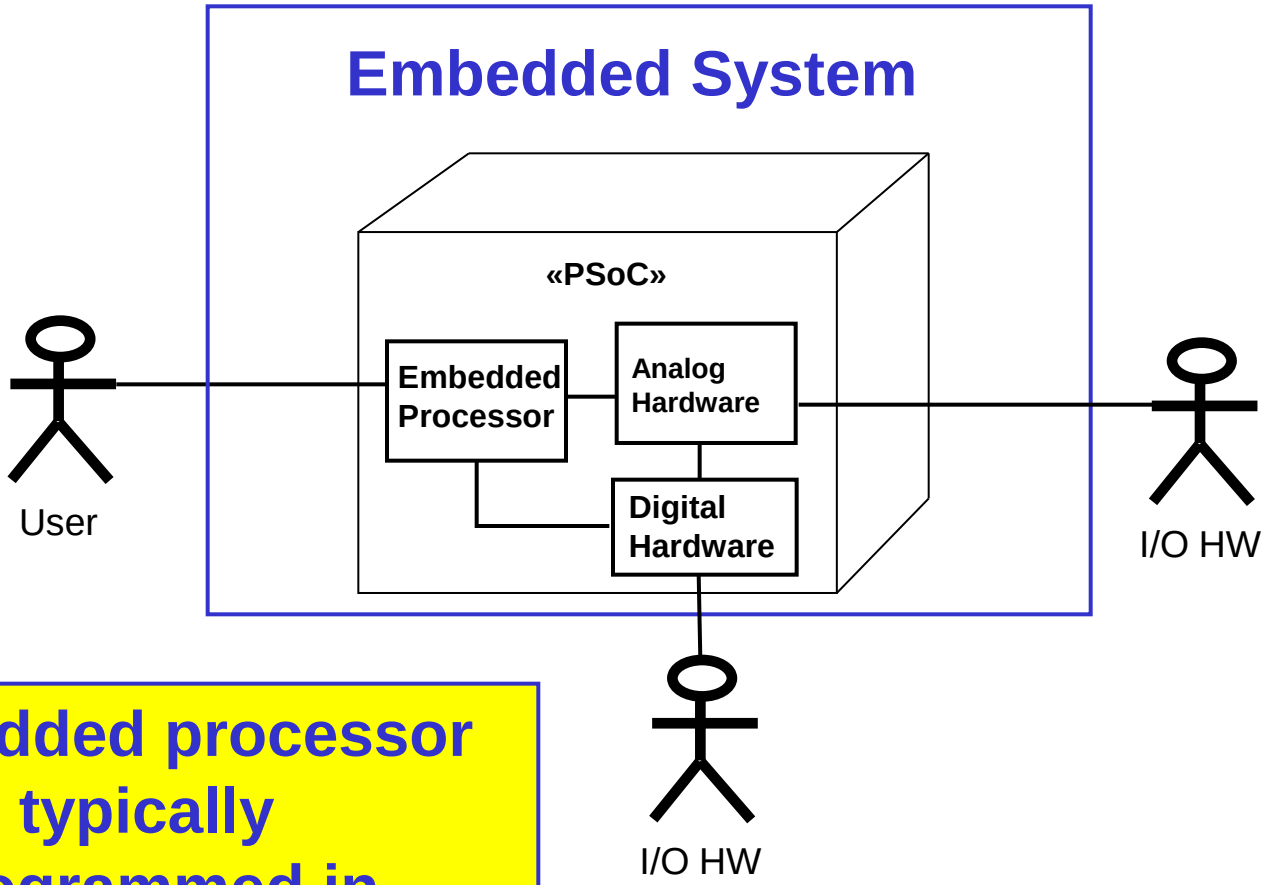


HW/SW Architecture – Example 6



Soft Core processors typically programmed in C or C++

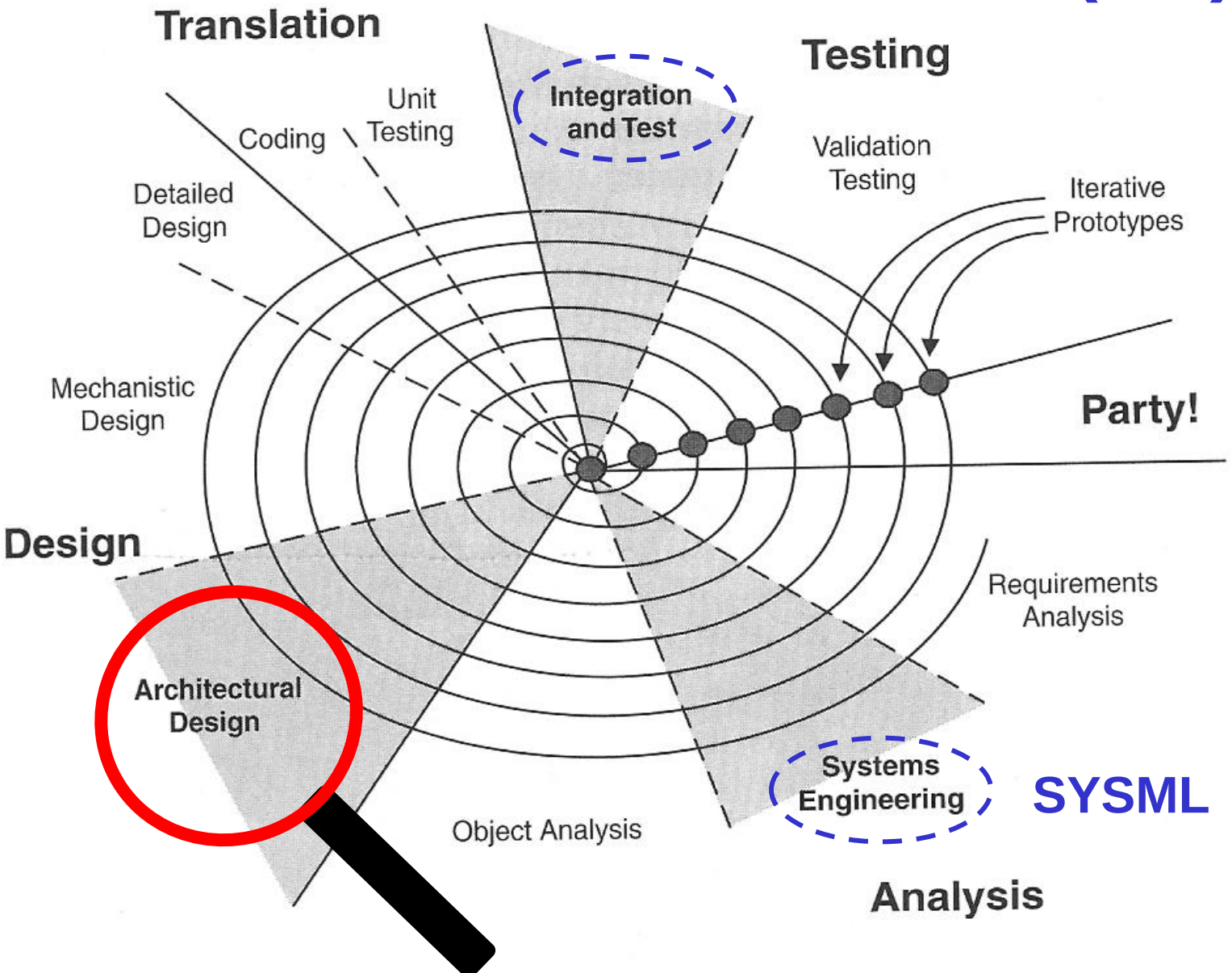
HW/SW Architecture – Example 7



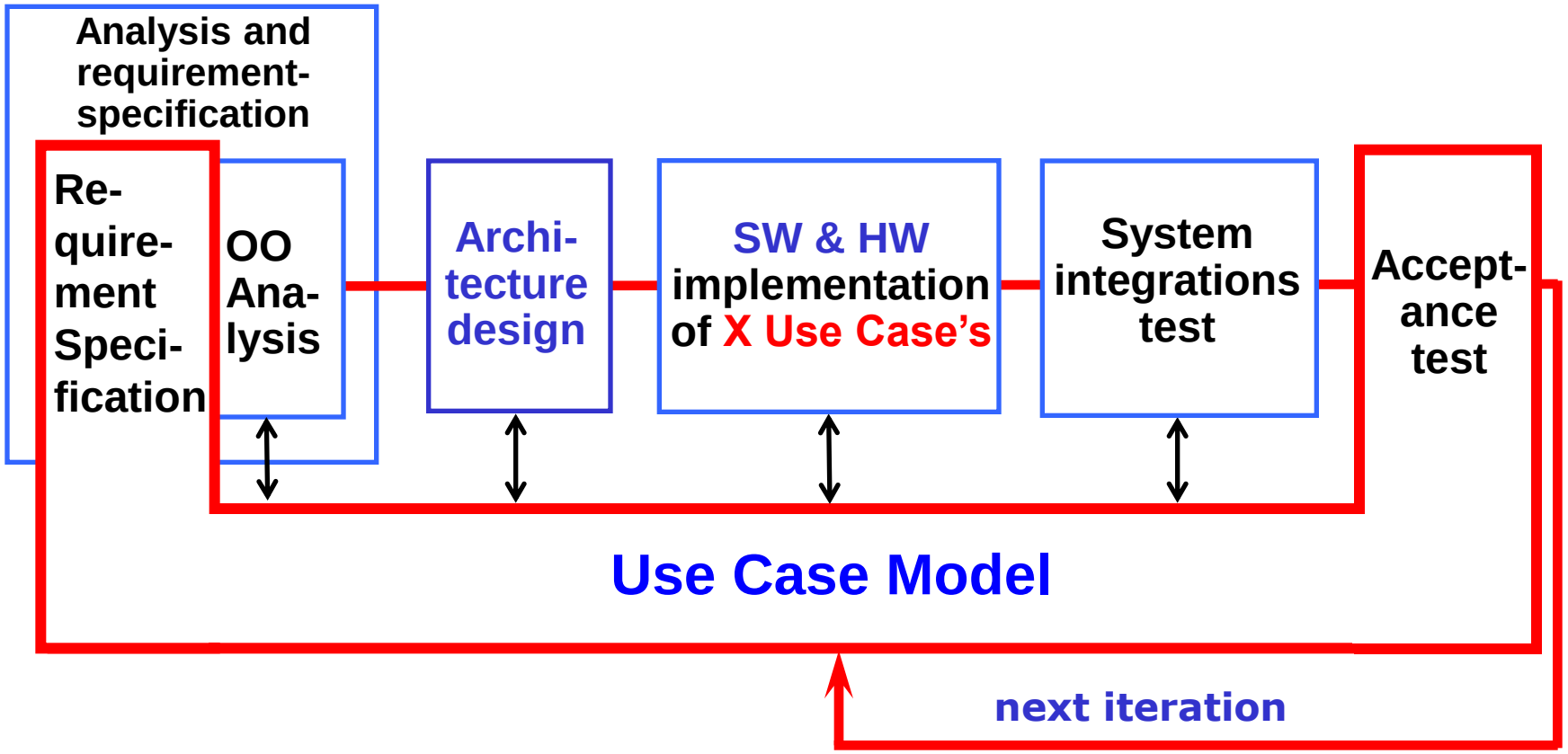
Embedded processor typically programmed in C or C++

PSoC a Cypress product

2. Architecture and UML (2-1)

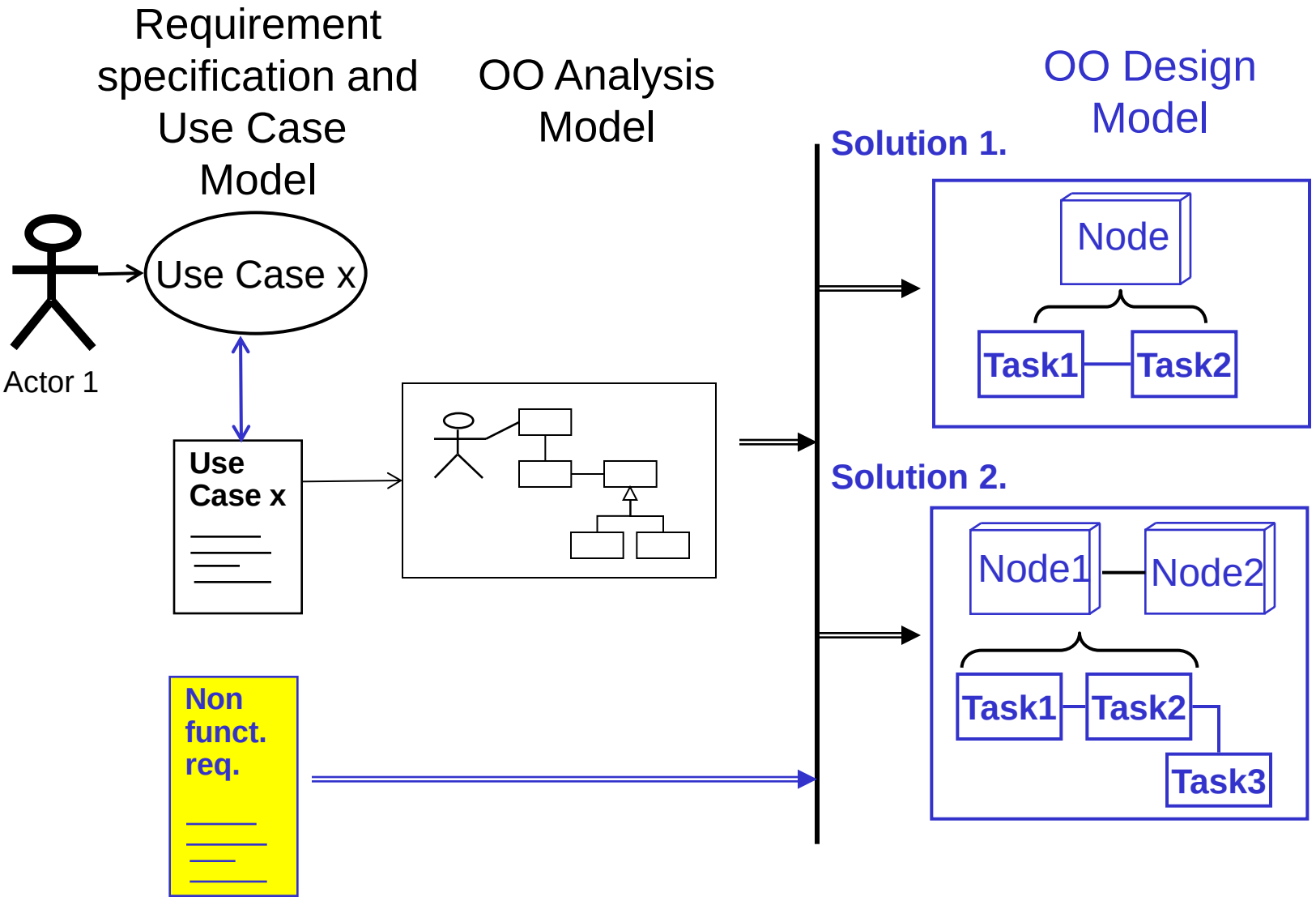


A Use Case driven and Iterative Development Model



This view focus on the importance of the **Use Case Model**

OO Analysis – OO Design

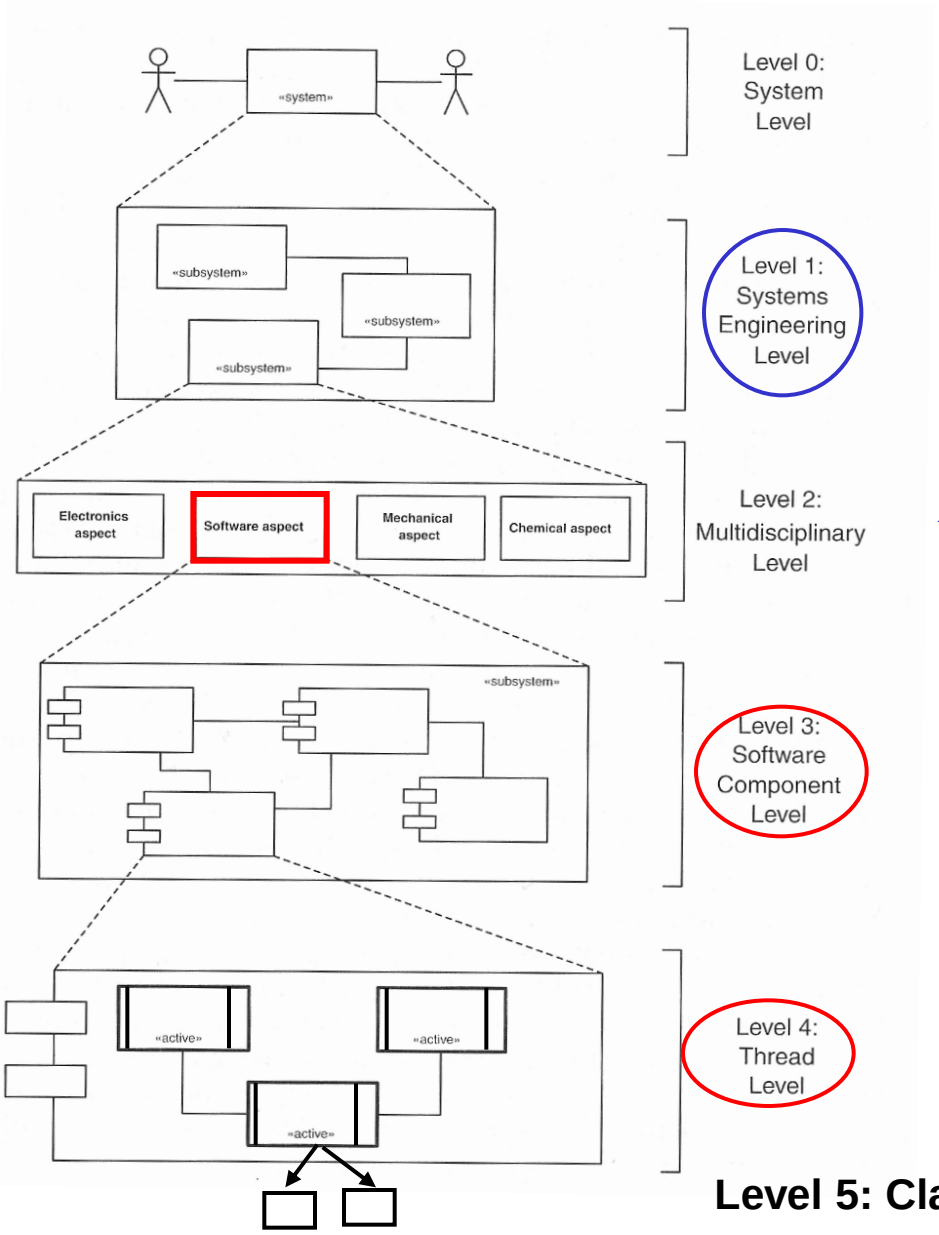


Design Characteristics

- An analysis model can be implemented by a number of **different design solutions**
- It is a normal **engineering discipline** to suggest and investigate a number of different solutions and select the best compromise for the given situation

Remember also to document the investigated design alternatives !

Levels of Abstraction in Architecture (2-6)



Level 5: Class/object level

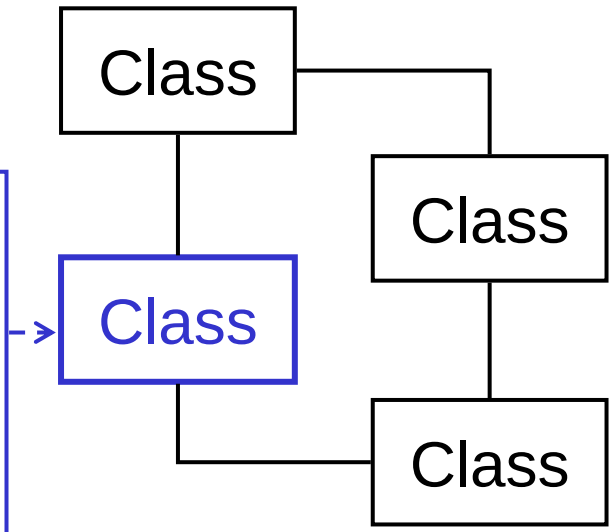
ROPES three Design Levels

1. Architectural Design

Scope: Processors, Packages, Components, Tasks

2. Mechanistic Design

Scope: Groups of collaborating objects



3. Detailed design

Scope: Class

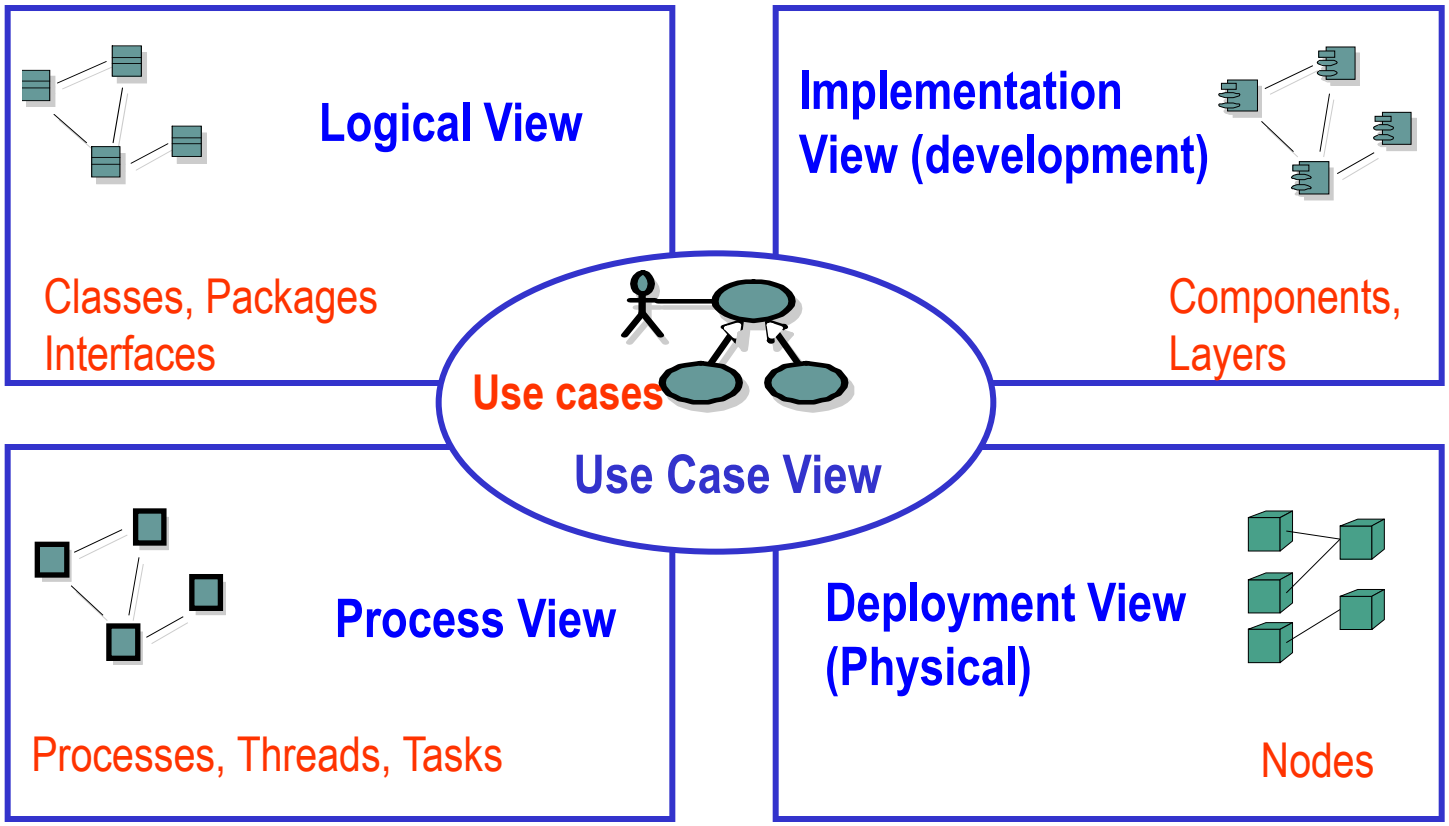
ROPES Design Activities

Design Phase	Scope	What is Specified
Architectural Design	**System-wide** **Processor-wide**	• Number and type of processors • Packages of objects running on each processor • Inter-processor communication media and protocols • Concurrency model and inter-thread communication strategies • Software layering and vertical slices • Global error handling policies
Mechanistic Design	**Inter-object**	• Instances of design patterns (GoF) of multiple objects collaborating together • Containers and design-level classes and objects • Medium-level error handling policies
Detailed Design	**Intra-object**	• Algorithmic details within an object • Details of data members (types, ranges) • Details of function members (arguments, internal structure)

3. Architectural Views

- Different view models are described in literature:
  - **Kruchten, 4+1 View Model**
  - **Nokia model, 3+1 View Model**
  - **Hofmeister, 4 View Model**
  - **Douglass, 5 View Model**
- All authors agrees on the main view concept
  - different views are needed
  - each view shows **a specific characteristic** of the system and **has different target groups**
  - but names and contents varies
- It is useful to supplements with a ”+1” view, describing scenarios or Use Cases
- There can be more views
  - e.g. RUP (Rational Unified Process) operates with both a **Data view** and a **Security view** as optional views

"4+1" View Model
for SW Architecture

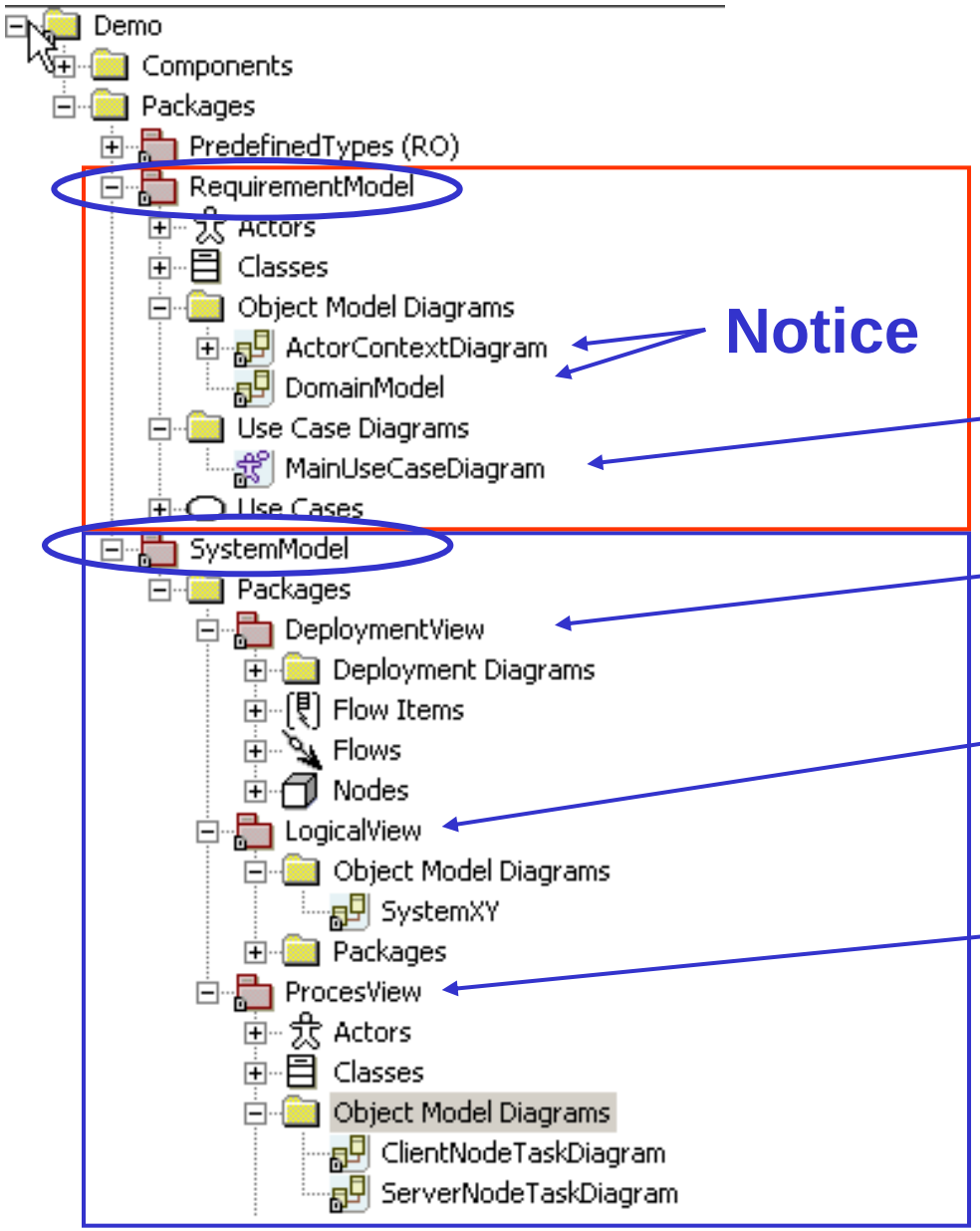


Ref.: Philippe Kruchten, "The 4+1 View of Architecture, "IEEE Software, 12(6) Nov. 1995

Architecture Views in ”4+1” Model

- **Logical View**
  - Describes the logical design by class diagrams with the overall organization described by packages
- **Implementation View** (also called development view)
  - Describes the static software organization in modules, physical layering and grouping of source code, data files, components etc.
- **Process or Task View (important for Real-Time Systems)**
  - Describes concurrency e.g. processes, tasks or threads and their communication and synchronization
- **Deployment View**
  - Describes computers and connected hardware devices and optional also allocation of run-time components on computers
- **Use Case / scenario View**
  - Describes a number of **selected scenarios**.
    The working of these scenarios are explained by the other views

Example: Views in Rhapsody



Notice

Use Case View

Deployment View

Logical View

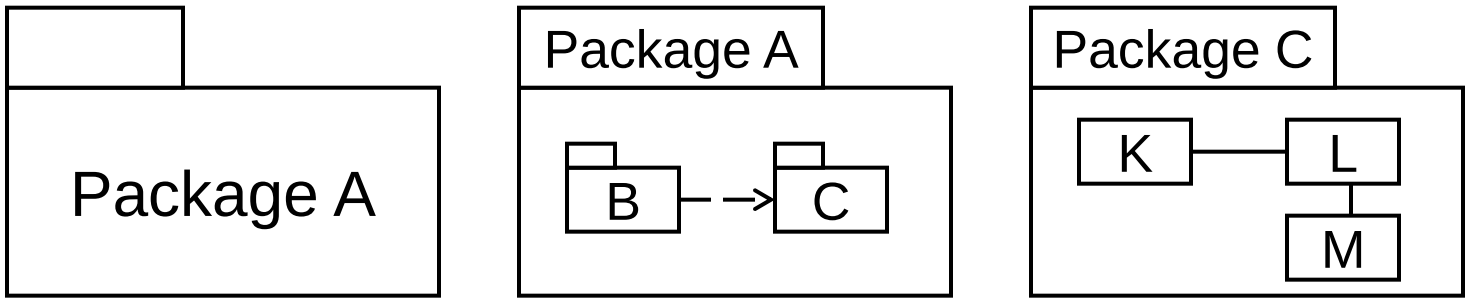
Process View

UML Design Views and Diagrams

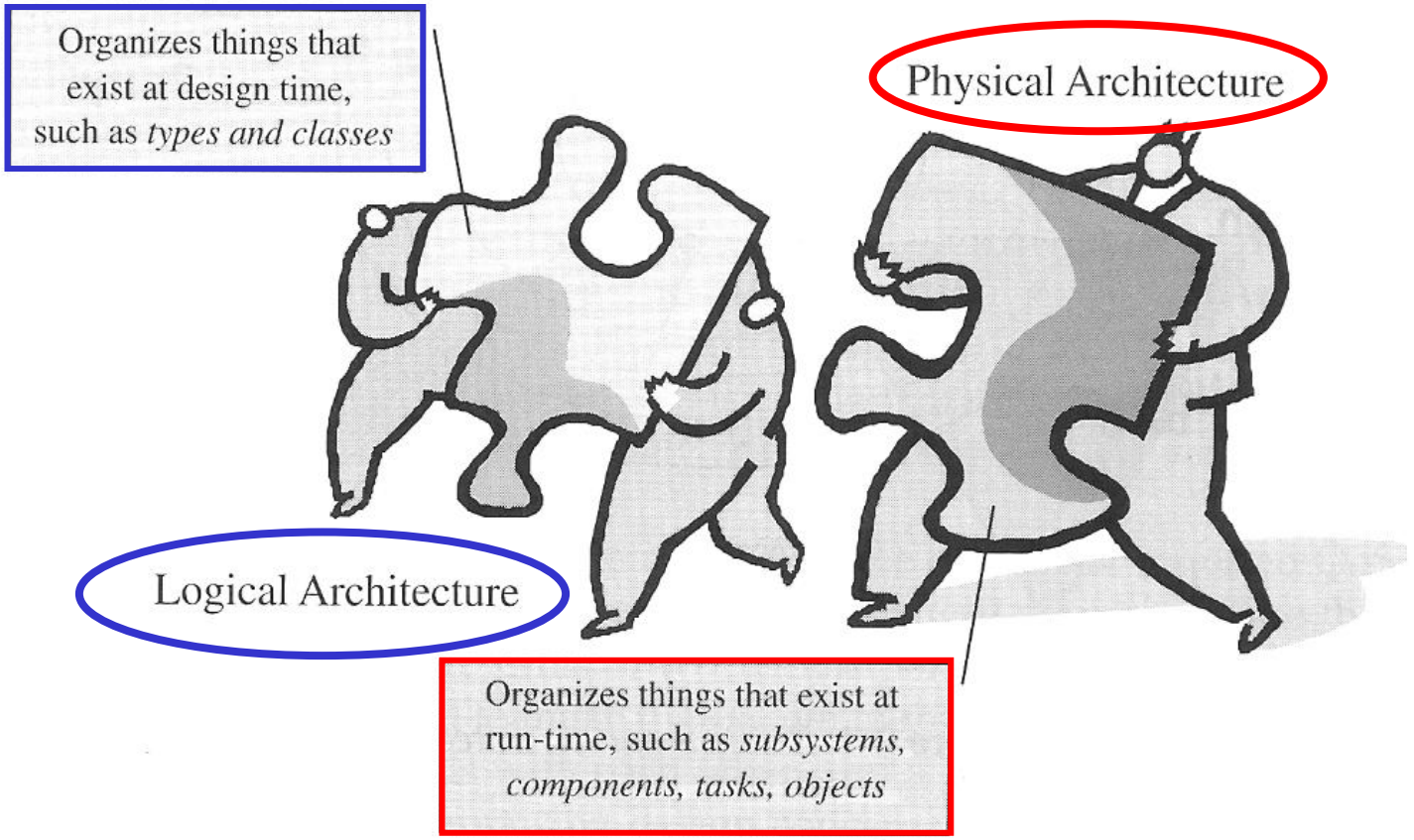
- The following diagram types are used in the **architectural design activities**
  - **Logical View: Class diagrams**
    - Showing logical packages or classes
  - **Deployment View: Deployment diagrams**
    - showing physical devices and connections
  - **Process View: Class diagrams**
    - showing primarily active classes (process/task/thread) and their communication mechanism
  - **Implementation View: Component diagrams**
    - showing implementation artifacts

Logical View: UML Packages

- In Requirement specification
  - to group use cases
- In OO analysis
  - to show a logical and hierarchical partition of the object model
- In OO design and implementation
  - to show the partition in physical subsystems



Logical and Physical Architecture (2-2)



One of the logical architectural patterns is to mirror the physical architecture structure

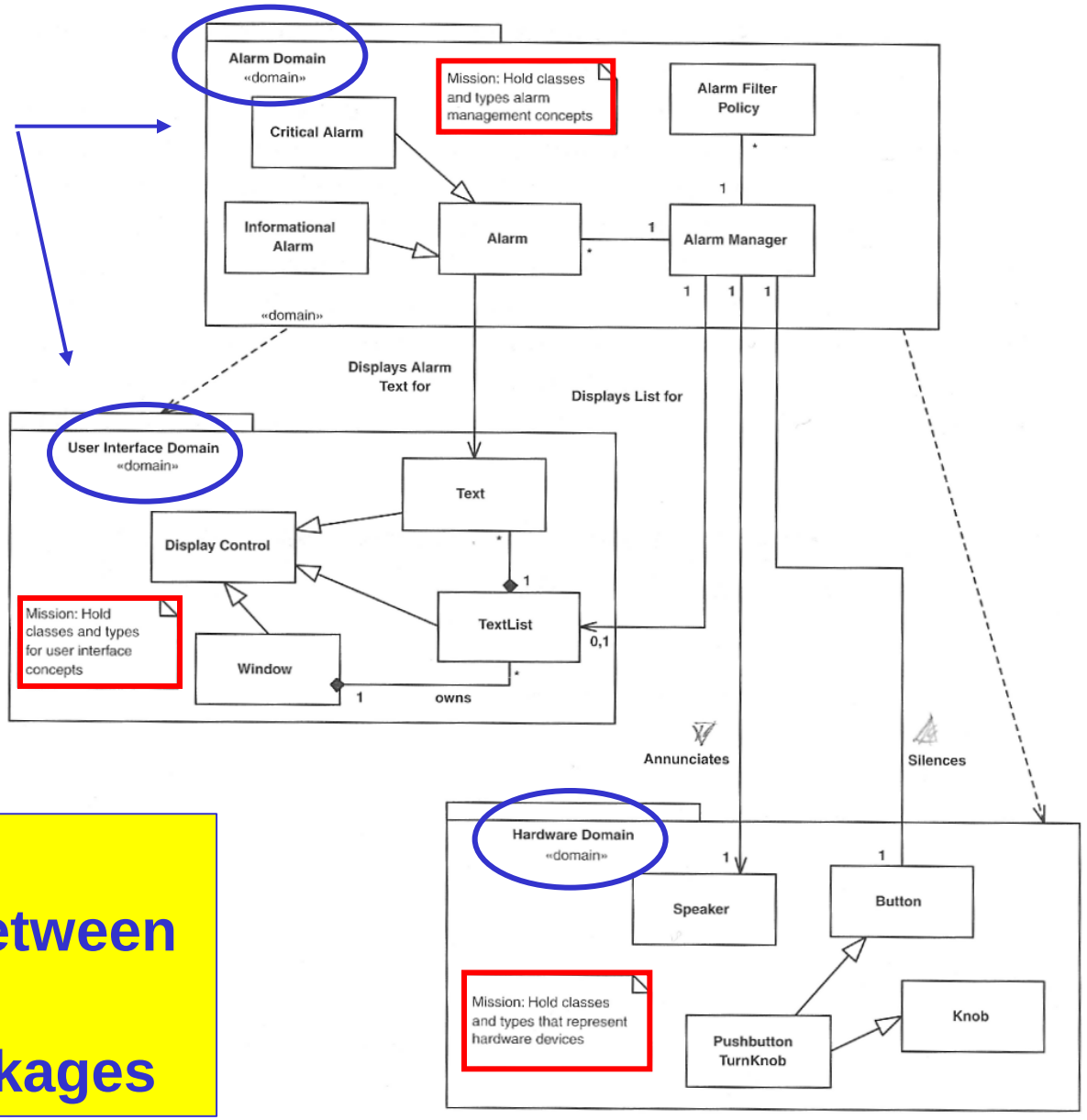
Logical Architecture

- **ROPES recommends** a logical architecture based on the concepts of **domains**
- A domain is an independent subject area
- Examples:
  - User Interfaces, Hardware
  - Alarm management, Communications
  - Operating Systems, Data Management,
  - Medical Diagnostics, Guidance and navigations,
  - Avionics, Image Reconstruction, Task planning

Logical Domain Architecture (2-3)

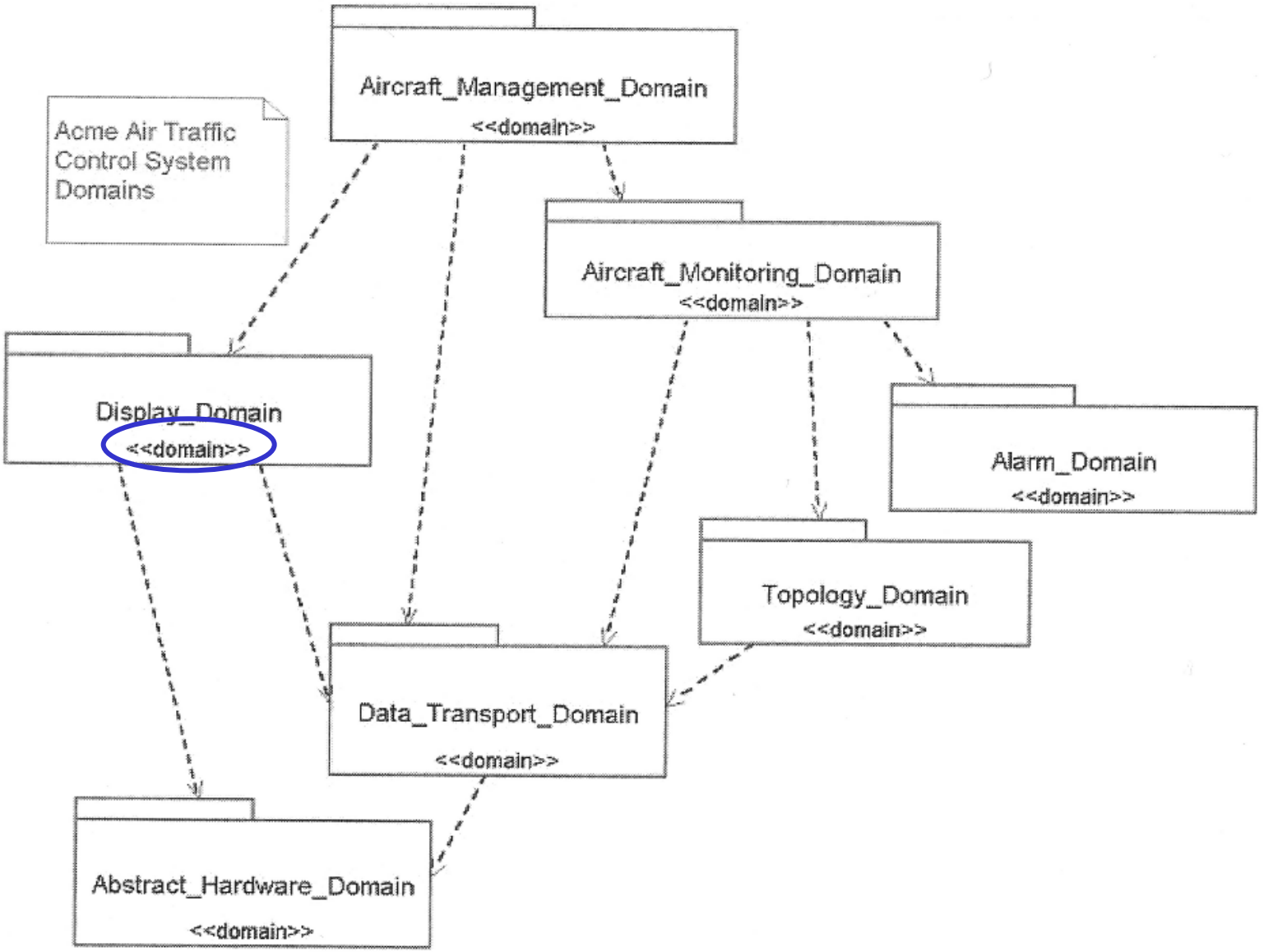
UML Packages

Notice

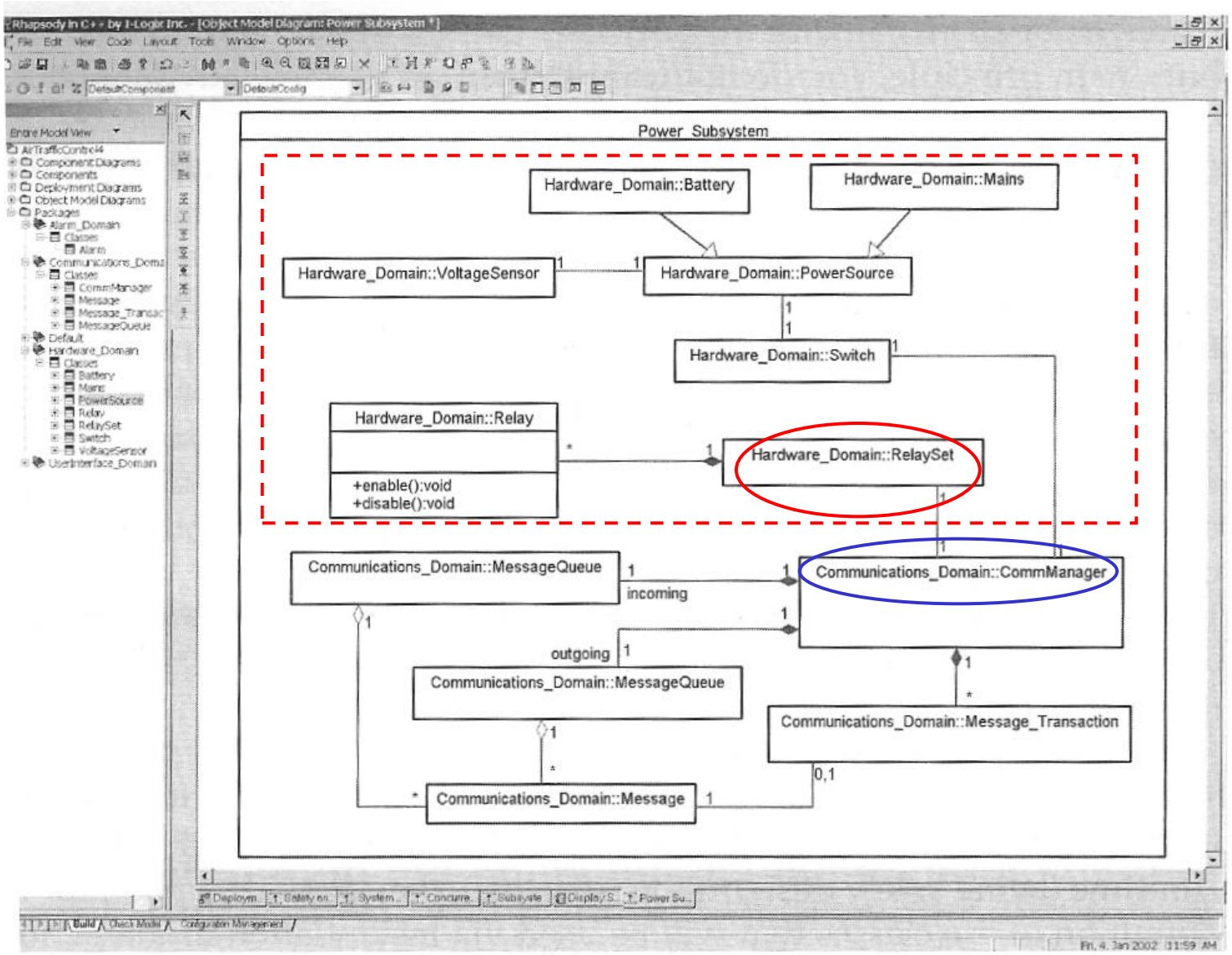


Notice:
associations between classes
in different packages

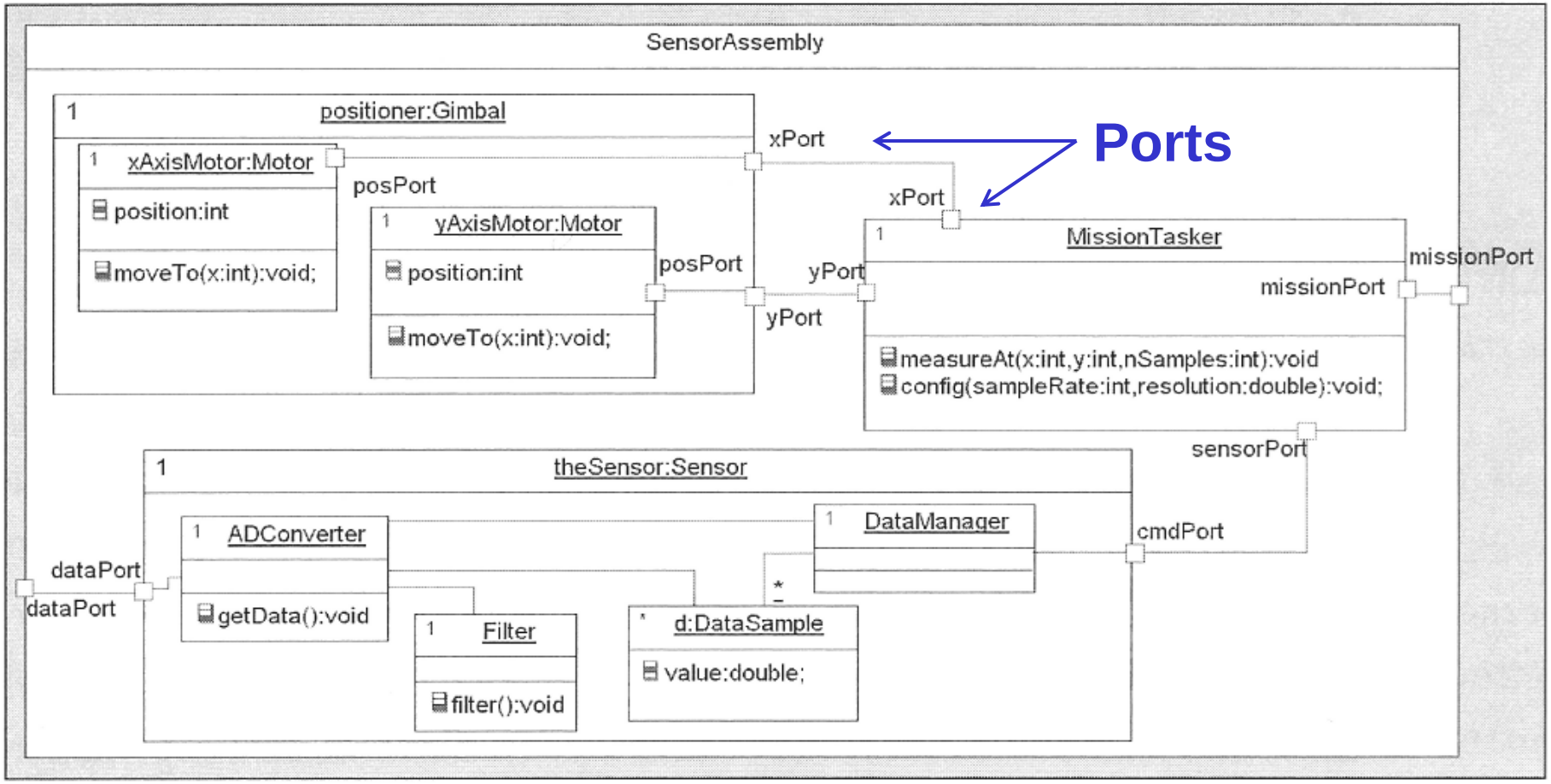
Domain Hierarchy (2-4)



Relating Logical and Physical Architecture (2-5)



Structured Class (a UML 2.x concept)

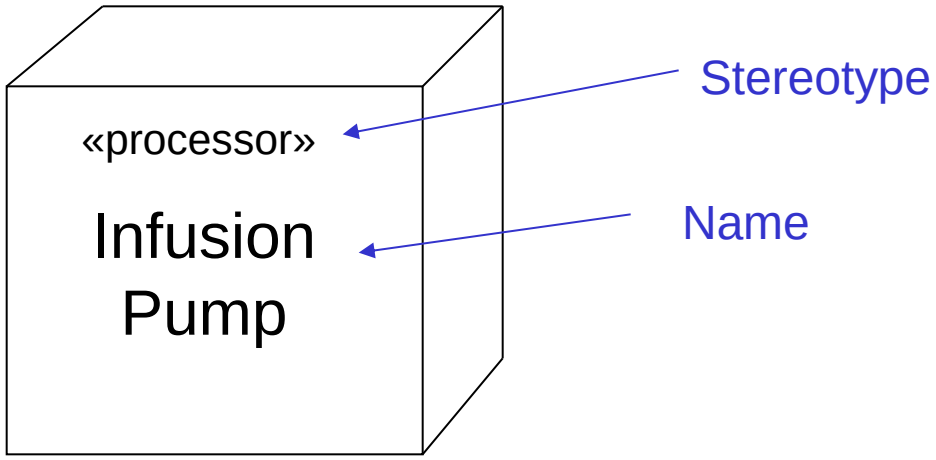


A huge advancement in the UML standard

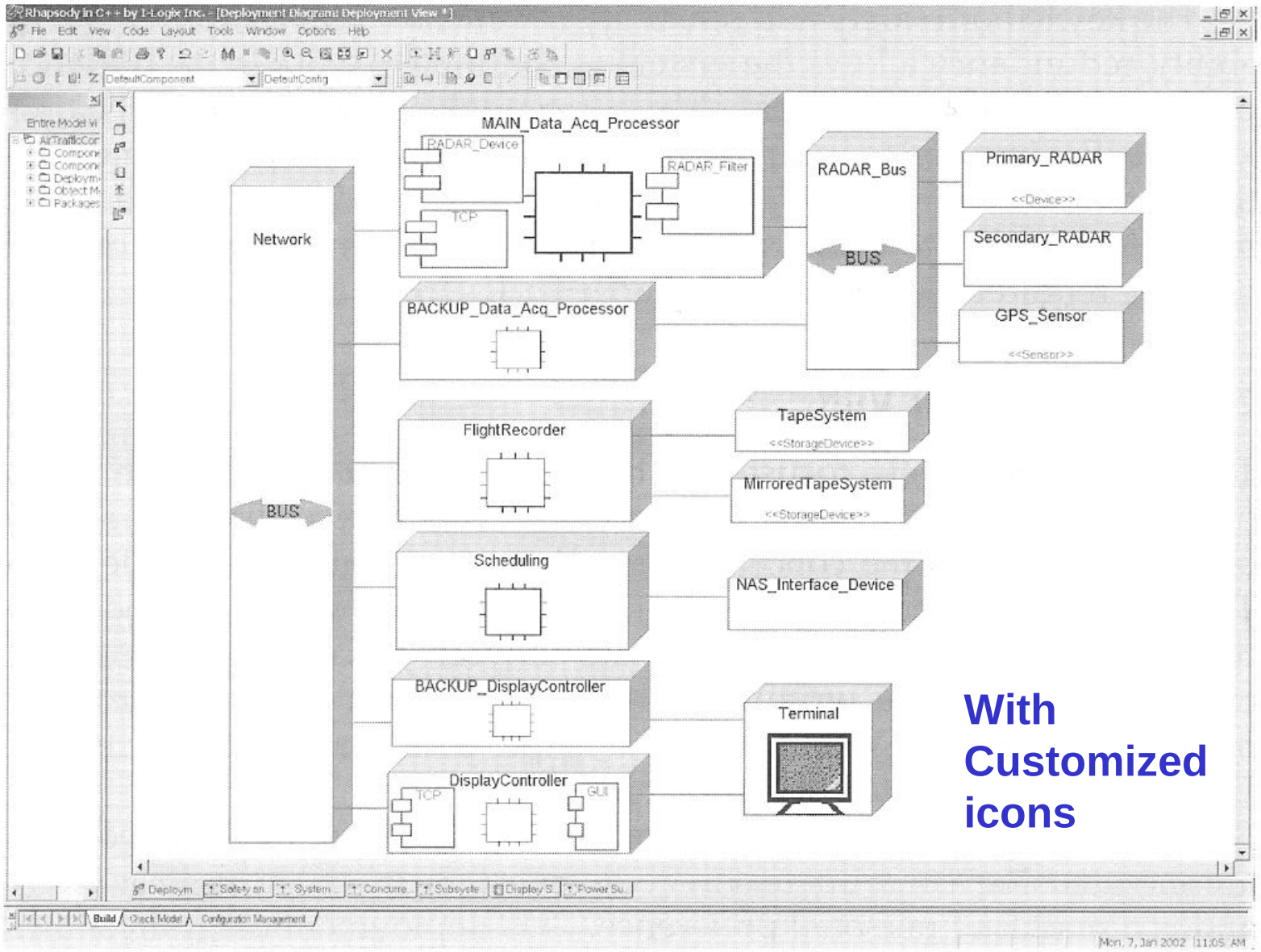
UML Deployment View: UML Nodes

Nodes represents

- either physical units (devices):
  - processors, sensors, actuators, routers, displays, input devices, memory, custom PLAs
  - it is very useful to mark the unit type with a stereotype
- or execution environments
- UML symbol



Deployment View (2-14)

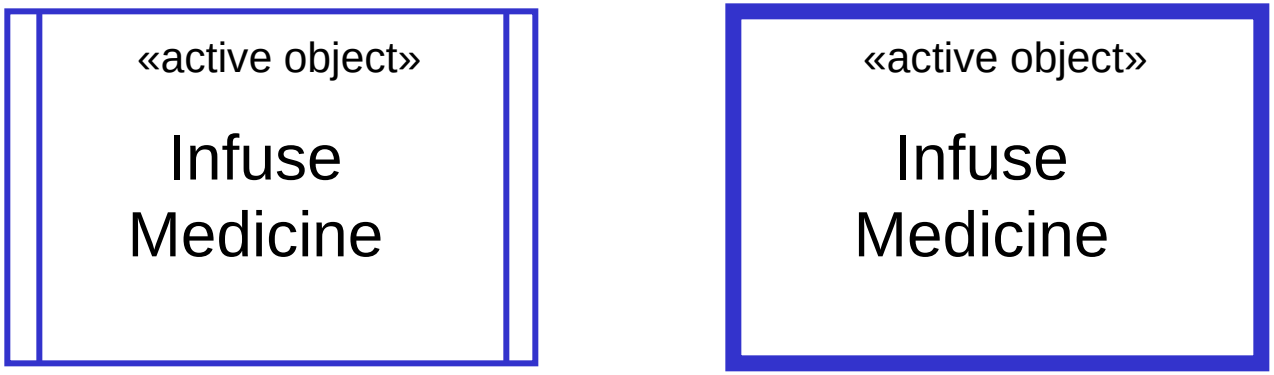


Process or Tasking View

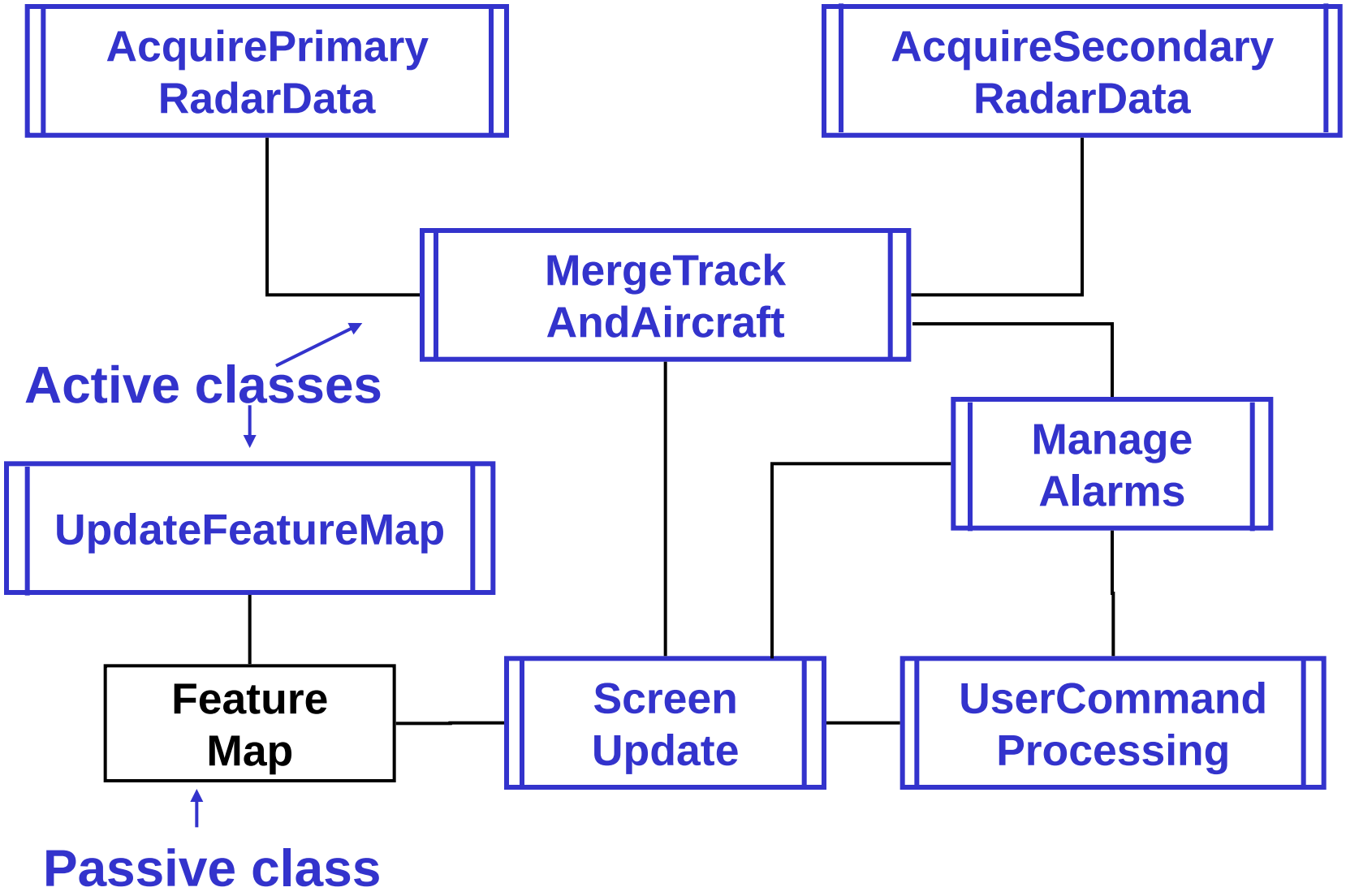
- A **thread** can be defined as a set of actions that executes sequentially
- Multiple objects typically participate within a single thread.
- **Heavyweight** and **lightweight** threads
  - heavyweight threads have its own data space, strong encapsulation, expensive message passing
    - often also called a **process** or a **task**
  - lightweight threads shares the same data space
    - often called a **thread**
  - Distinction between these two types can be shown with a UML stereotype («thread»)
- **Interrupt-threads/drivers**
  - an asynchronous activated sw-routine

Active Objects (Tasks)

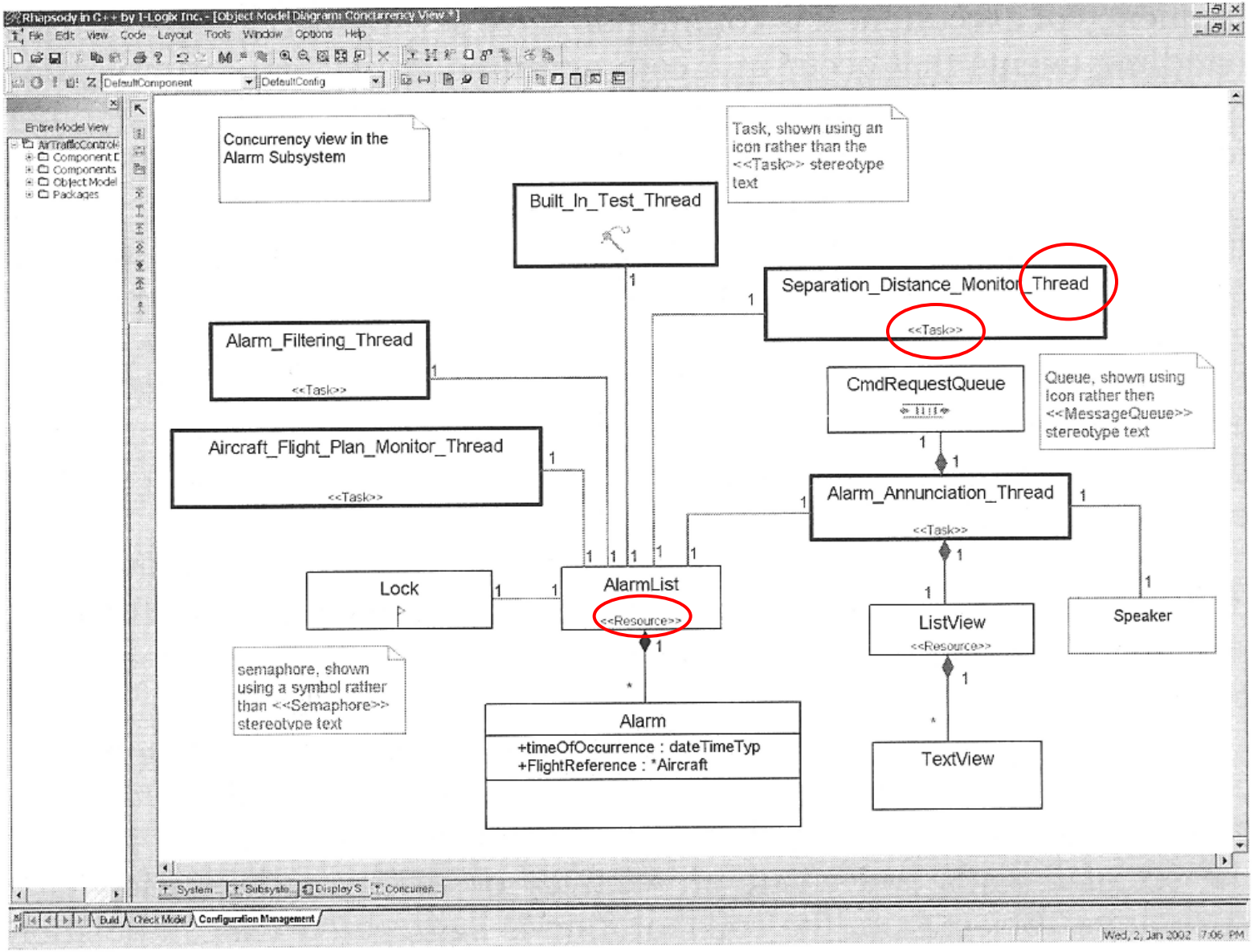
- An ***active*** object is an object who execute in it own thread
- In Real Time UML called a task or thread
- UML 2.0 symbol UML 1.x symbol



AATCS Task Diagram



Concurrency and Resource View (2-11)

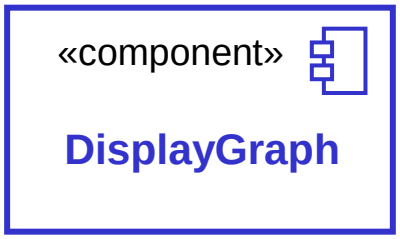


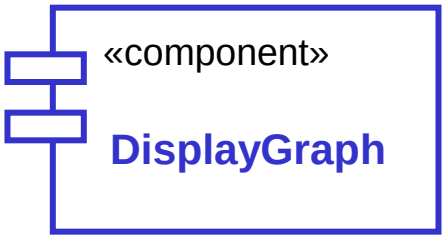
Concurrency in UML diagrams

- The following UML diagram types can be used to show/design for concurrency:
  - Task diagrams (class or object diagrams with active and passive classes/objects)
  - Concurrent State Diagrams
  - Concurrent Activity Diagrams
  - Concurrent Sequence Diagrams
  - Concurrent Communication Diagrams

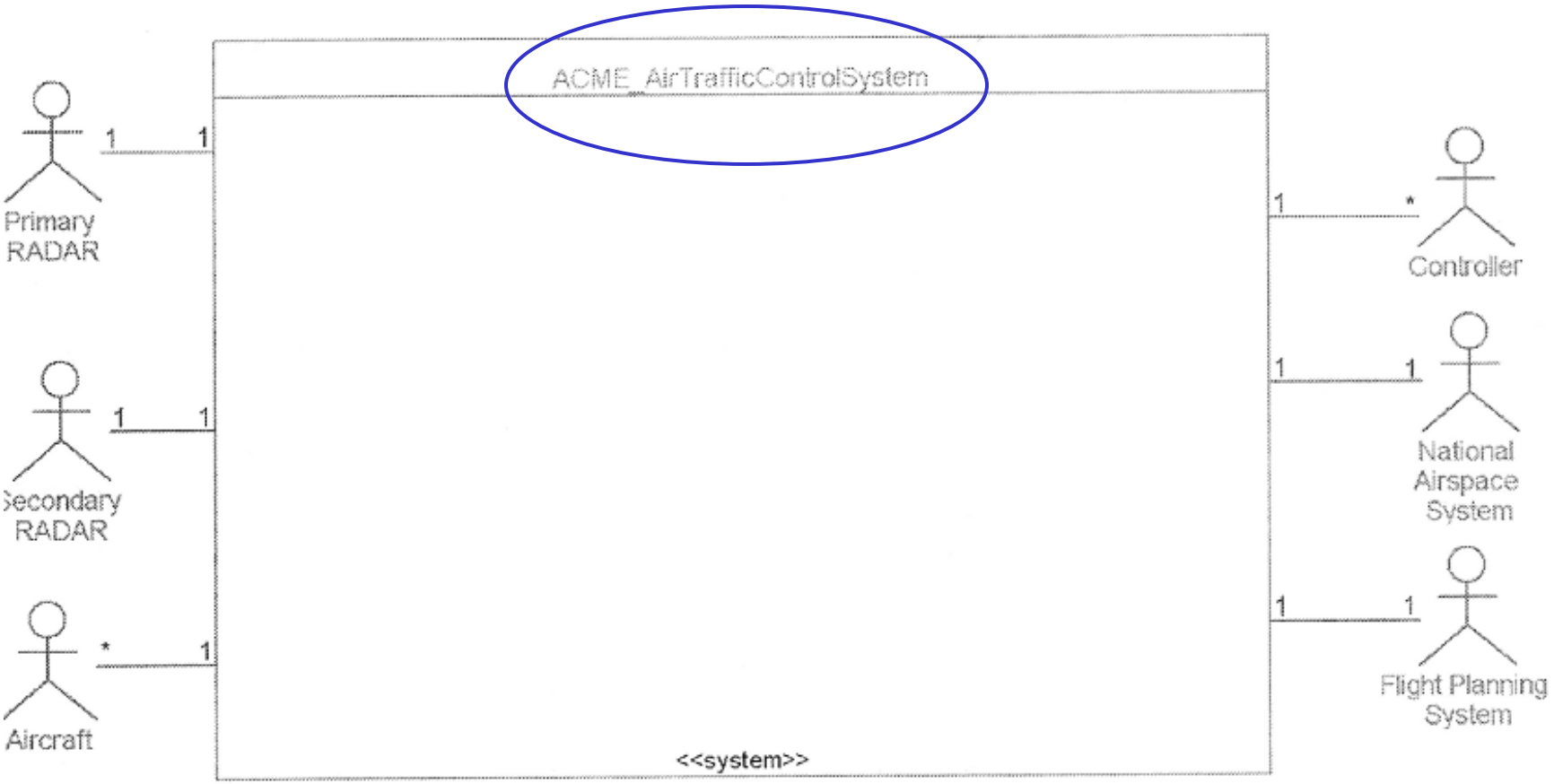
Implementation View

- UML Component diagram
- A ***Component*** describes a software unit, existing at ***run-time***, examples:
  - An executable program file (exe)
  - A library (dll)
  - A binary component (ocx/dll)
  - A table (database, data in ROM)
  - A file (e.g. a configuration file, a html/xml file)
- **UML 2.0 symbol UML 1.x symbol**



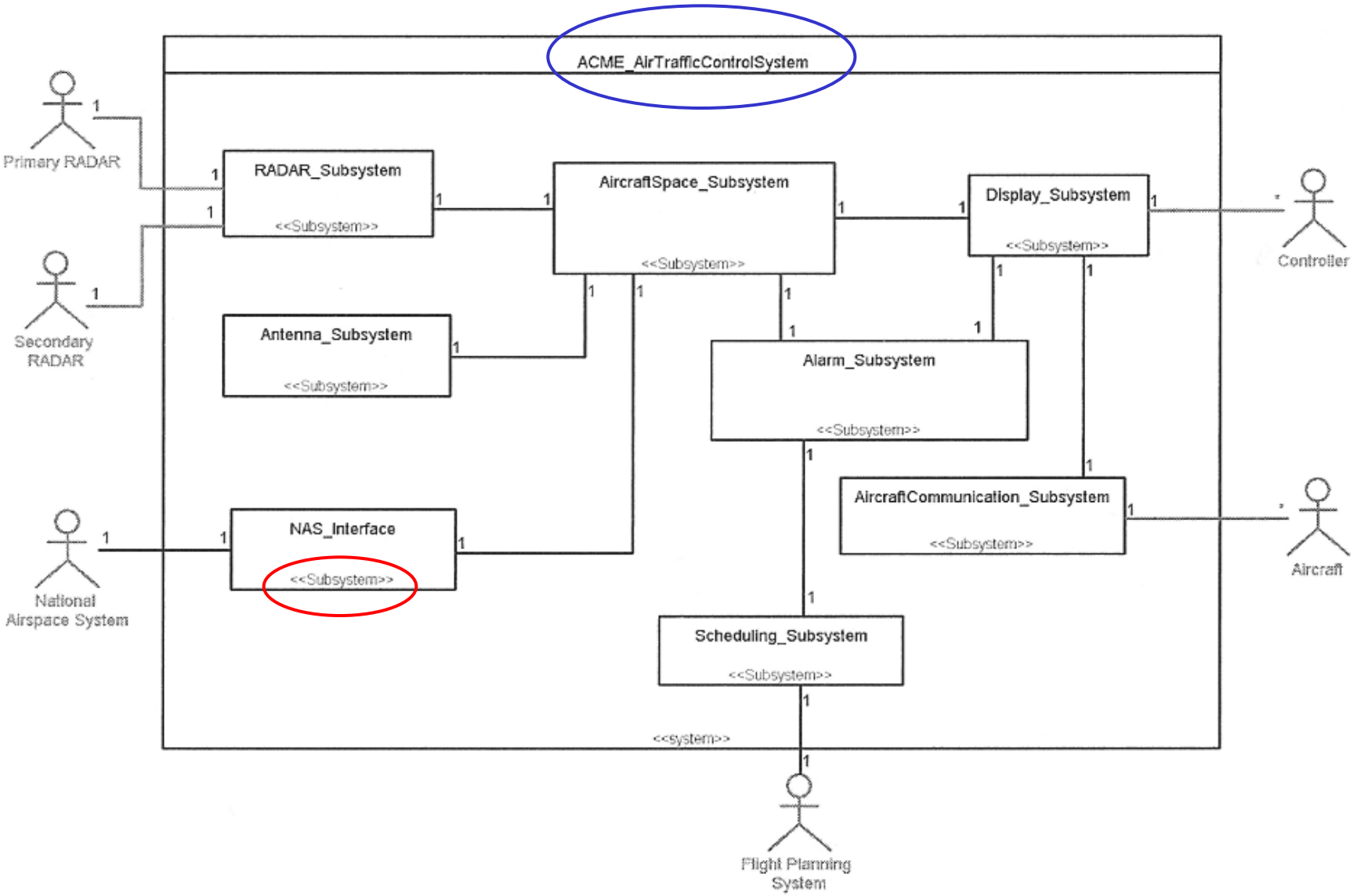


System View (2-8)

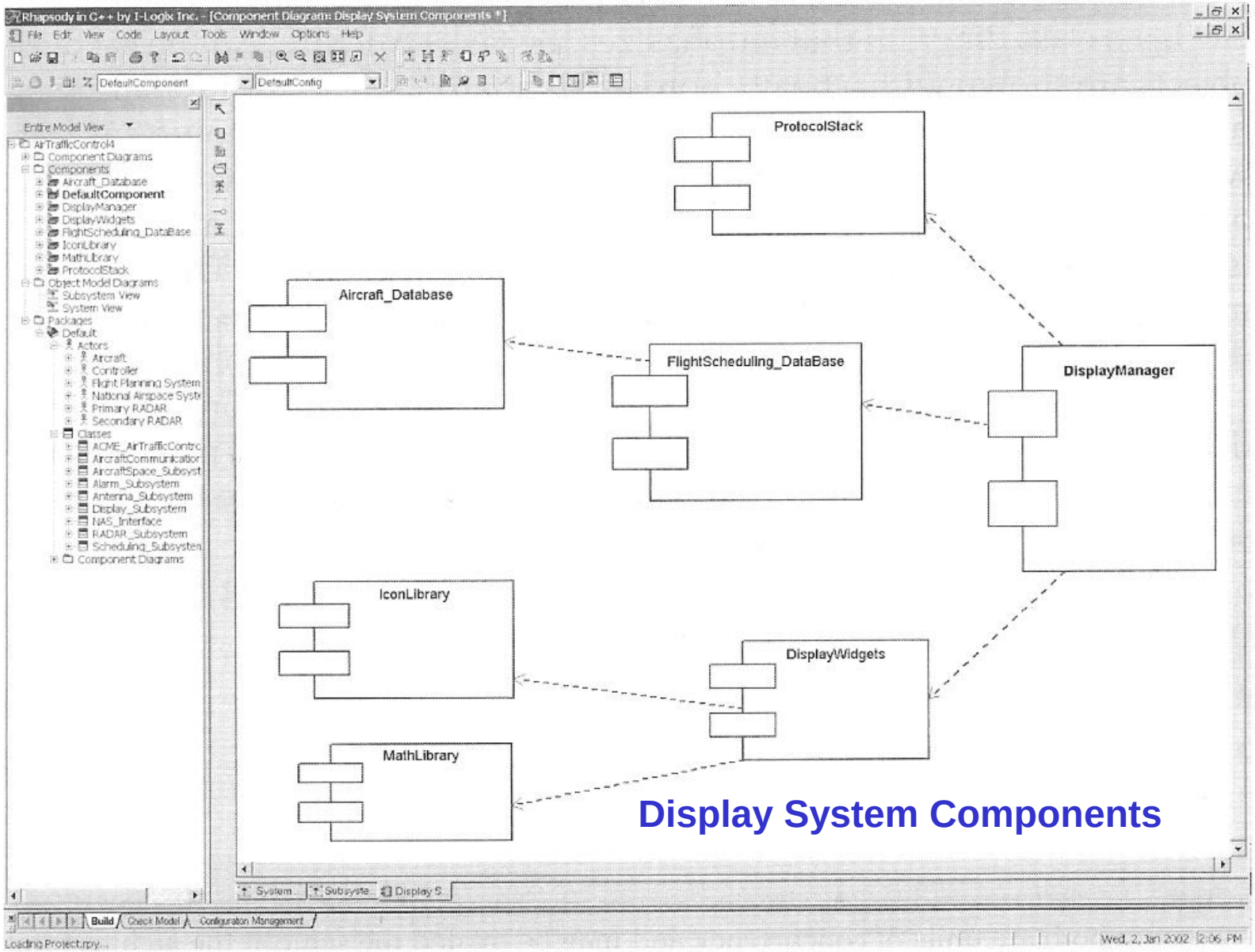


An Actor-context Diagram

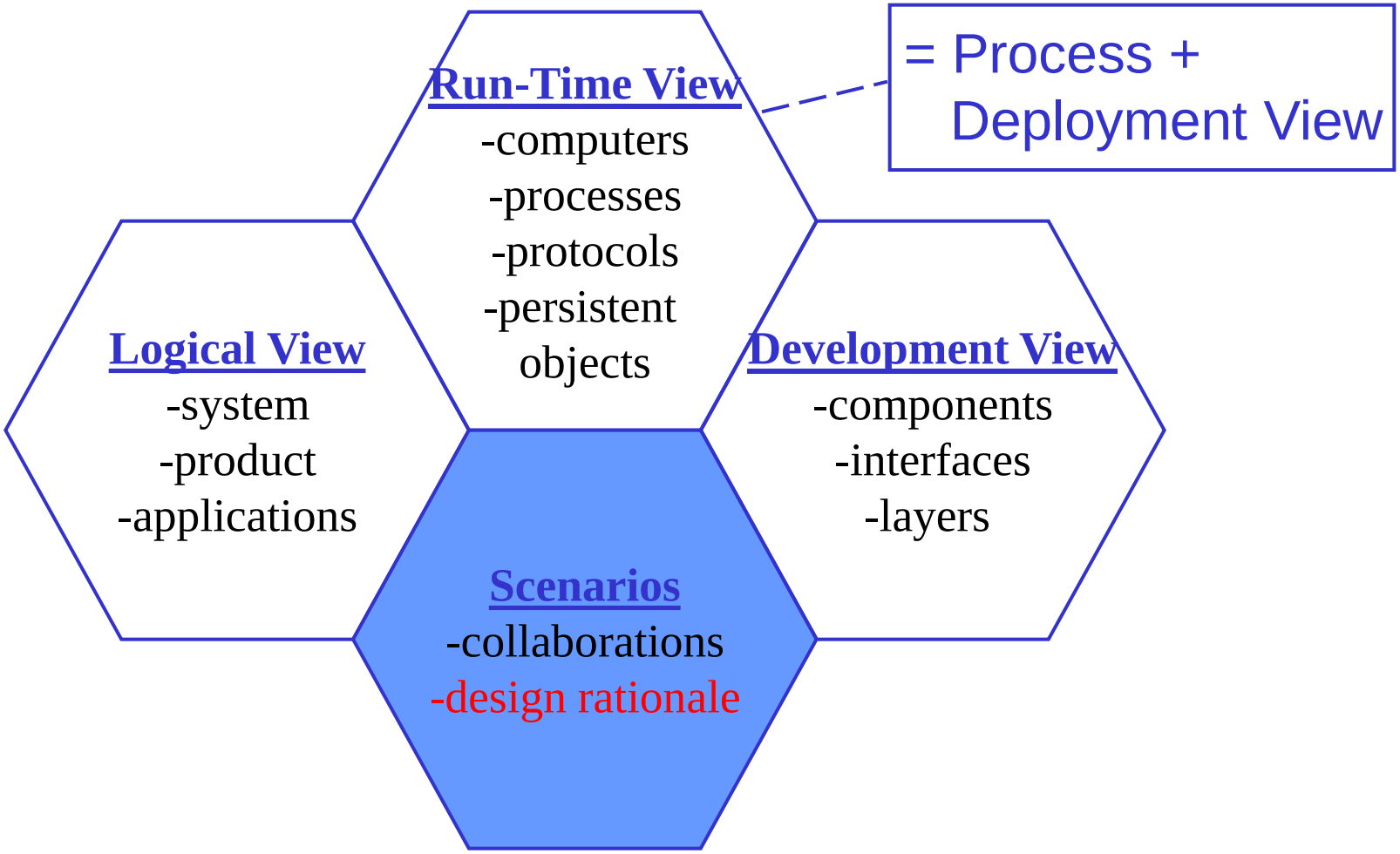
Subsystem View (2-9)



Component View (2-10)

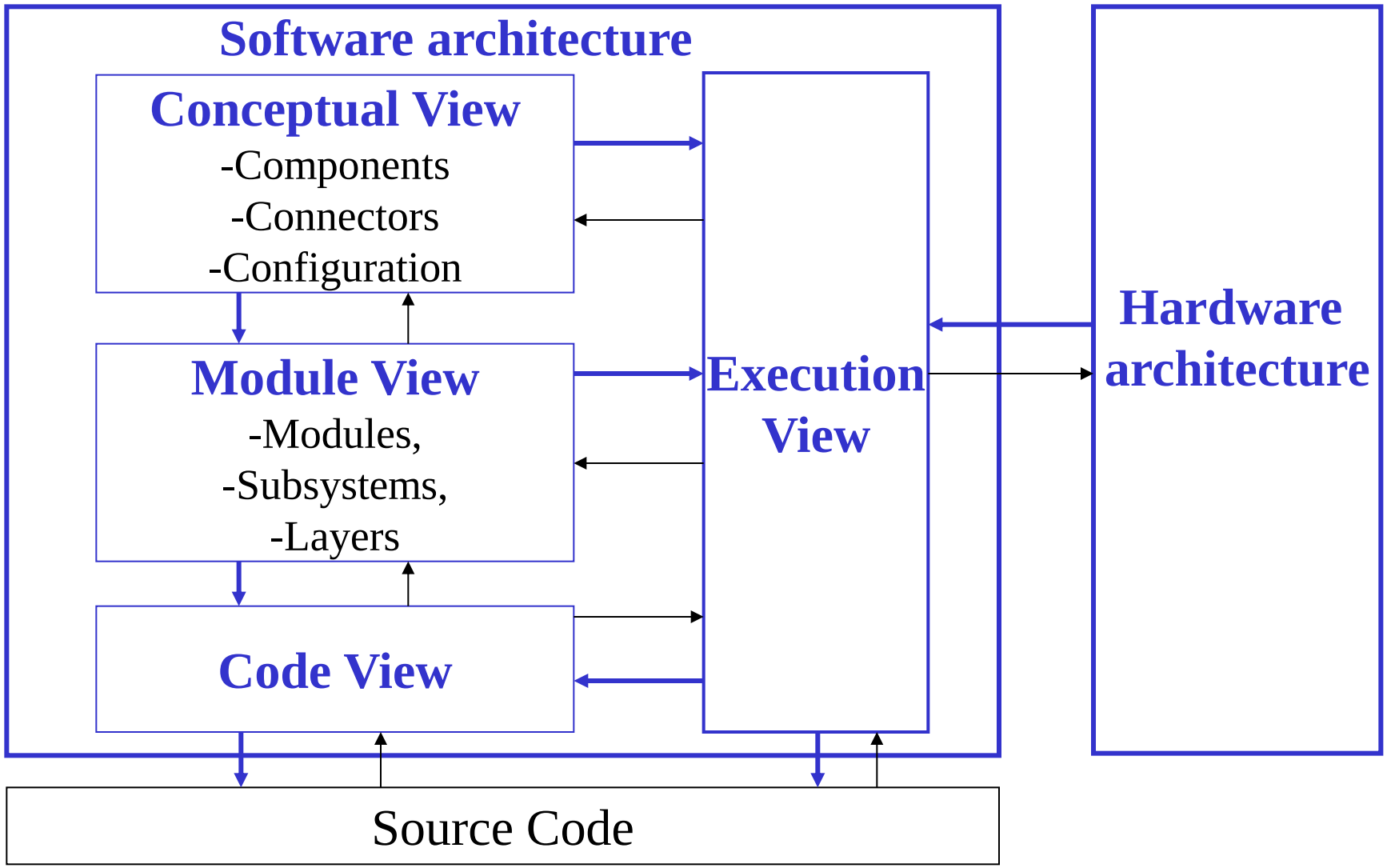


Nokias ”3+1” View Model



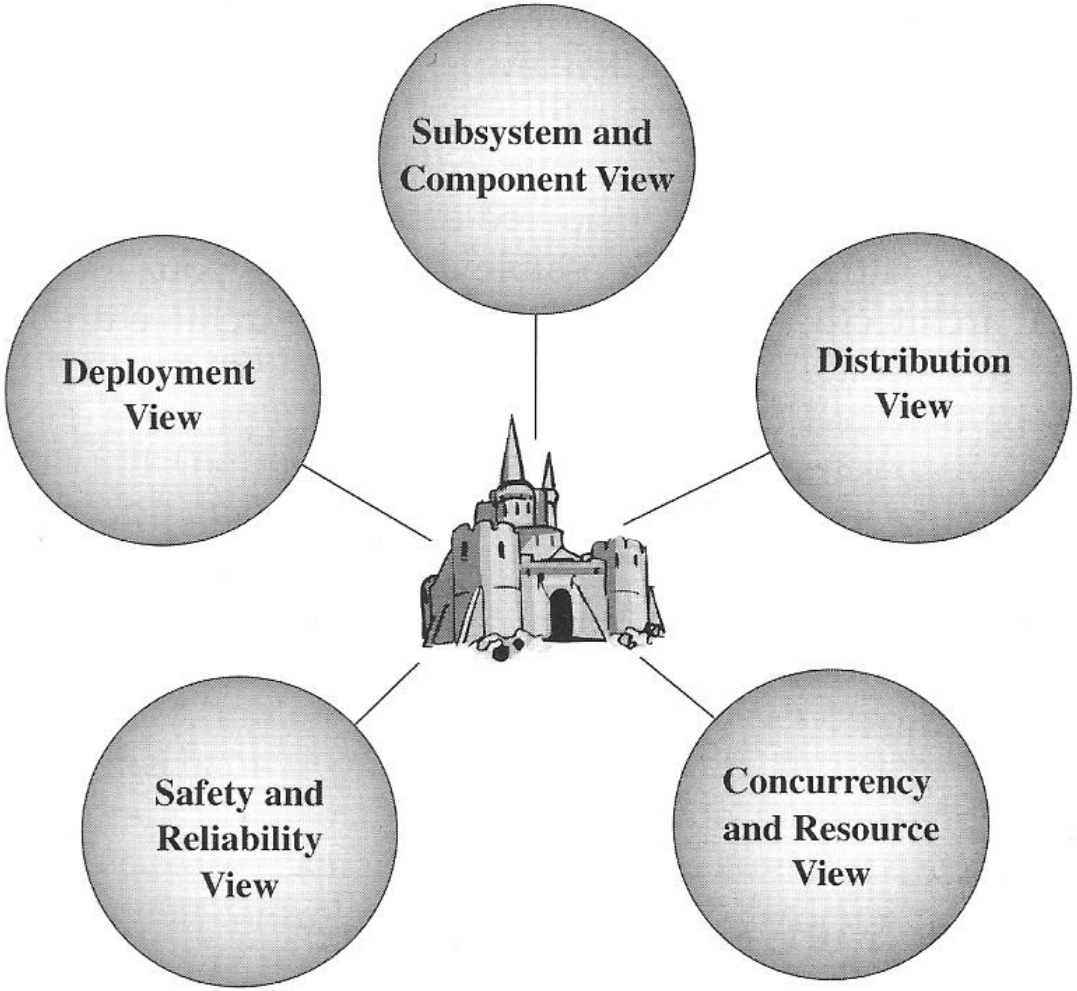
Ref: [Jaaksi99]

Hofmeister´s View Model



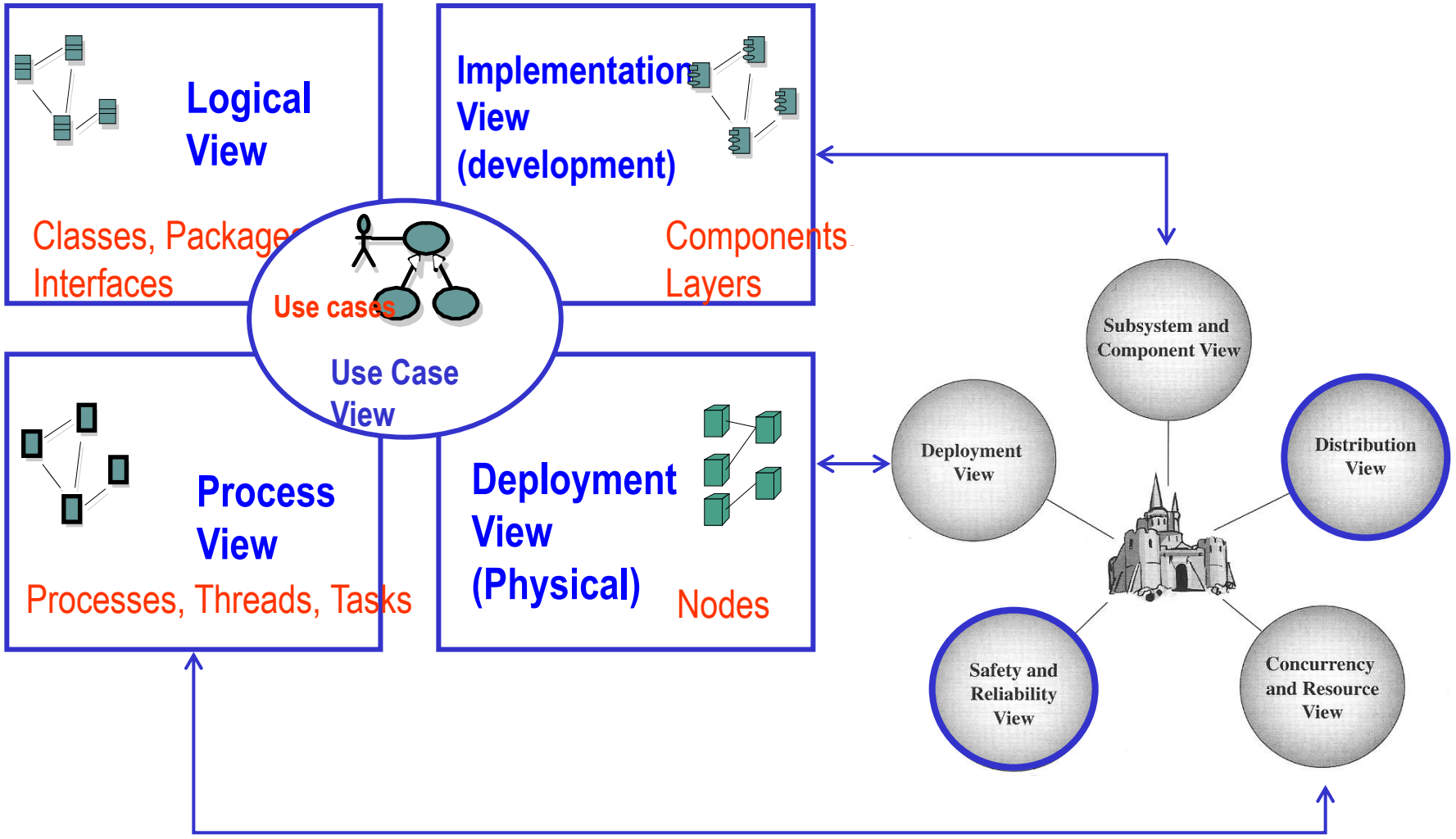
Ref: [Hofmeister2000]

BPDs **Five Views** of **Physical Architecture** (2-7)

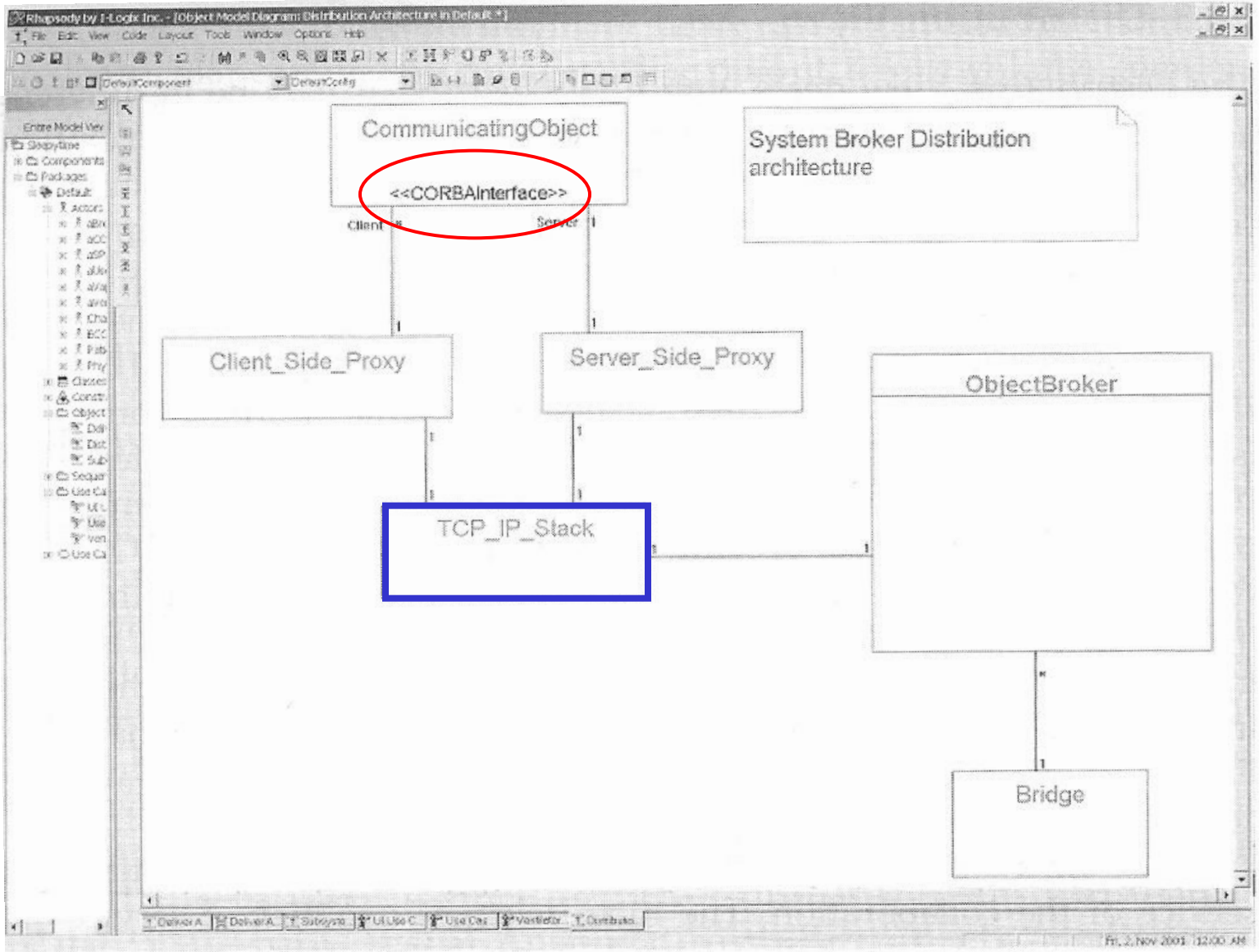


Ref: [Douglass2003]

Comparison between View Models



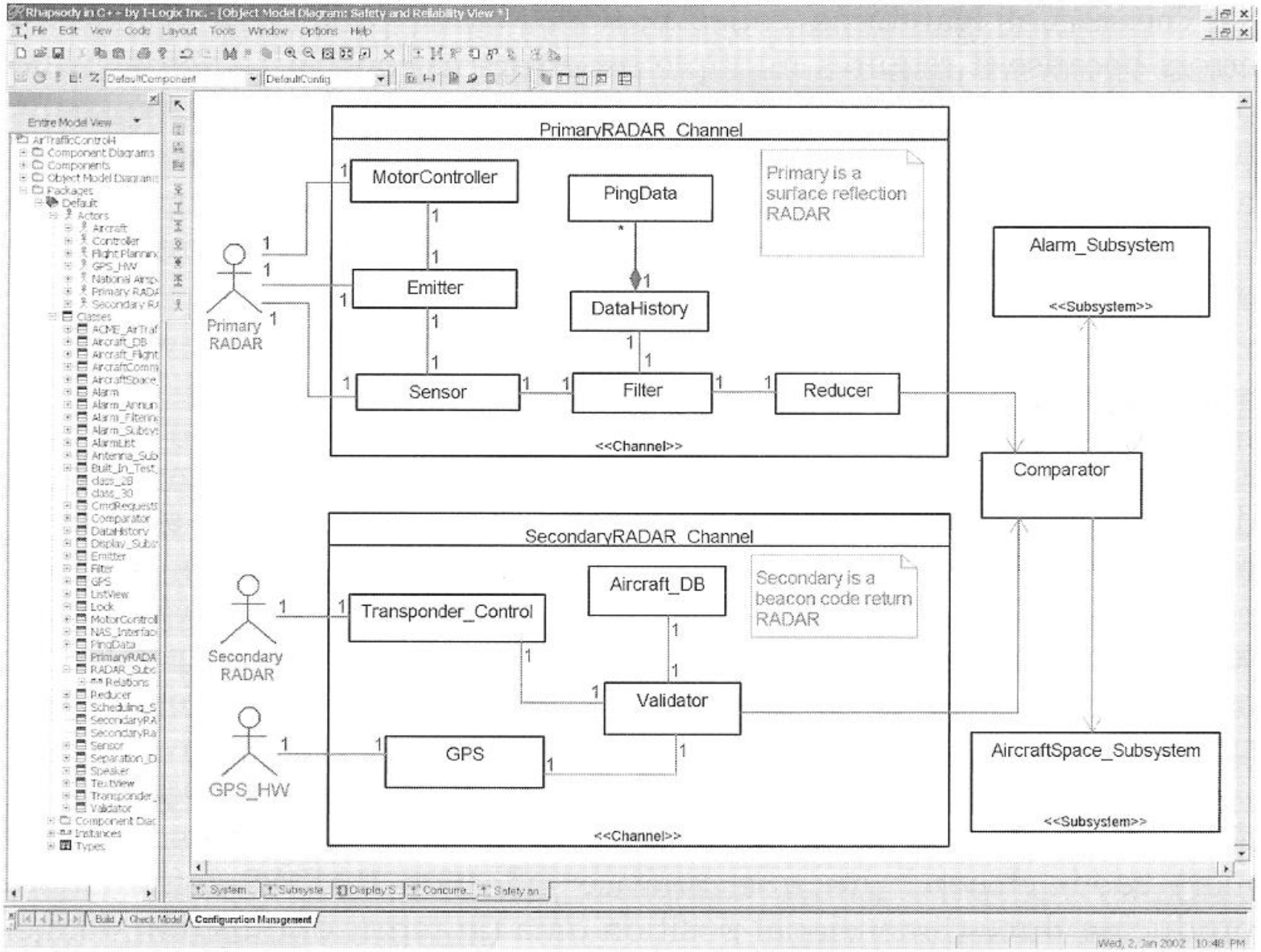
Distribution View (2-12)



Safety and Reliability View

- A **safe** system is a system that does not create accidents leading to injury, the loss of life, or damage to property
- A **reliable** system is a system that continues to function for long periods of time
- In **safety-critical** environments, applications must continue to do the right thing **in the presence of single-point failures**

Safety and Reliability View (2-13)



Development Process and Architecture

- Use Case driven development process
  - Select the **architectural significant use cases**
  - Develop an **executable prototype** for these use cases (in one or more iterations)
- Use the “4+1” view model to develop an architecture solution (iterate modeling for each use case)
- Use Concurrent Engineering (concurrent SW + HW development)
  - Architecture defines the hole system both HW and SW
- Focus early on development of an **infrastructure**
- Wrap existing (procedural) code into classes
- Investigate the possibility to develop **frameworks** for **product families** (Examples: HP Printers, Nokia Mobile Telephones)

Coupling and Cohesion

- The following **fundamental design guidelines** can be used on all three design levels:
  - **Minimize Coupling** between units
    - well defined and minimum interfaces
    - low traffic between units
  - **Maximize cohesion** in a unit
    - high internal cohesion
    - each unit has a well defined responsibility
    - high internal traffic
  - [Unit = sub system, task, component, class]

Architectural Design Method Steps

1. Select an **architectural significant** Use Case
2. Start with the OO analysis model for this use case
3. Investigate one or more design solutions – use the ”4+1” model (Deployment -, Process- and Implementation view)
4. Try to use an architectural style or pattern
5. Incorporate the new design classes and update the design models (diagrams + model information)
6. Add the information to the System Architecture Document (see template on course web site)
7. If there are more architecture significant use cases in the current iteration then continue with step. 1

Summary

- Examples of HW/SW architectures
- Different architecture view models
- Architecture design possibilities with UML
- Architecture & Process

References

- [Gamma95] Design Patterns, Elements of Reusable Object-Oriented Software
  - Eric Gamma, R. Helm, Ralph Johnson, J. Vlissides, Addison-Wesley, 1995
- [Kruchten95] The 4+1 View Model of Architecture
  - Philippe Kruchten, IEEE Software, 12 (6), November 1995, IEEE
  - Kan downloades fra www.rational.com
- [Shaw&Garlan96] Software Architecture, Perspectives on an Emerging Discipline
  - Mary Shaw, David Garlan, Prentice-Hall, 1996
- [Clements96] Software Architecture: An Executive Overview
  - Paul C. Clements, Linda M. Northrop, Technical Report CMU/SEI-96-TR-003, feb. 1996
- [Bushmann 96] Pattern-Oriented Software Architecture - A System of Patterns (POSA1)
  - Frank Bushmann et. al., John Wiley & Sons, 1996
- [Bass98] Software Architecture in Practice
  - L. Bass, P. Clements and R. Kazman, Addison-Wesley, 1998
- [Jaaksi99] Tried & True Object Development, Industry-Proven Approaches with UML,
  - Ari Jaaksi, J-M. Aalto, A.Aalto, K.Vättö, Cambridge University press, 1999
- [Hofmeister2000] Applied Software Architecture
  - Christine Hofmeister, Robert Nord, Dilip Soni, Addison-Wesley, 2000
- [Bosch2000] Design and Use of Software Architecture, Adapting and evolving a product line approach
  - Jan Bosch, Addison-Wesley, 2000
- [Jazayeri] Software Architecture for Product Families, Principles and Practice
  - Mehdi Jazayeri, Alexander Ran, Frank van der Linden, Addison-Wesley, 2000
- [Douglass2003] Real-Time Design Patterns
  - Bruce Powel Douglass, Addison Wesley, 2003
- [COT] Center for Object Technology (COT),
  - a Danish research project financed by Center for IT Research.
    The COT project has a number of reports on SW architecture,
  - http://www.cit.dk/COT/ - see under Report Series