Telelogic

Real-Time UML

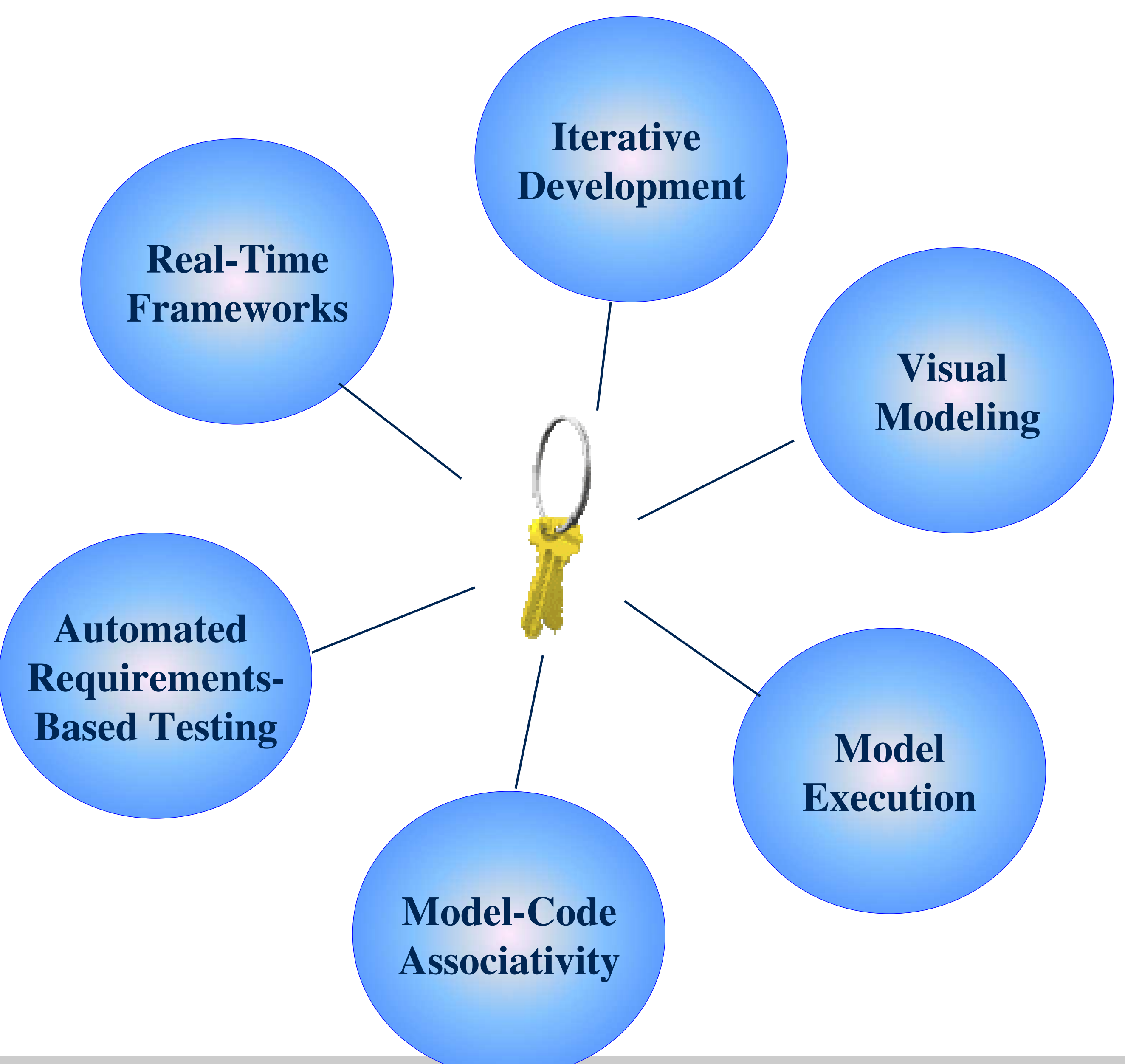
Bruce Powel Douglass, PhD
Chief Evangelist
Telelogic
www.ilogix.com
groups.yahoo.com/group/RT-UML

What is UML?

- Unified Modeling Language
- Comprehensive full life-cycle 3rd Generation modeling language
  - Standardized in 1997 by the OMG
  - Created by a consortium of 12 companies from various domains
  - I-Logix a key contributor to the UML including the definition of behavioral modeling
- Incorporates state of the art Software and Systems A&D concepts
- Matches the growing complexity of real-time systems
  - Large scale systems, Networking, Web enabling, Data management
- Extensible and configurable
- Unprecedented inter-disciplinary market penetration
  - Used for both software and systems engineering
- UML 2.0 is latest version (2.1 is in the pipeline)

© Telelogic AB

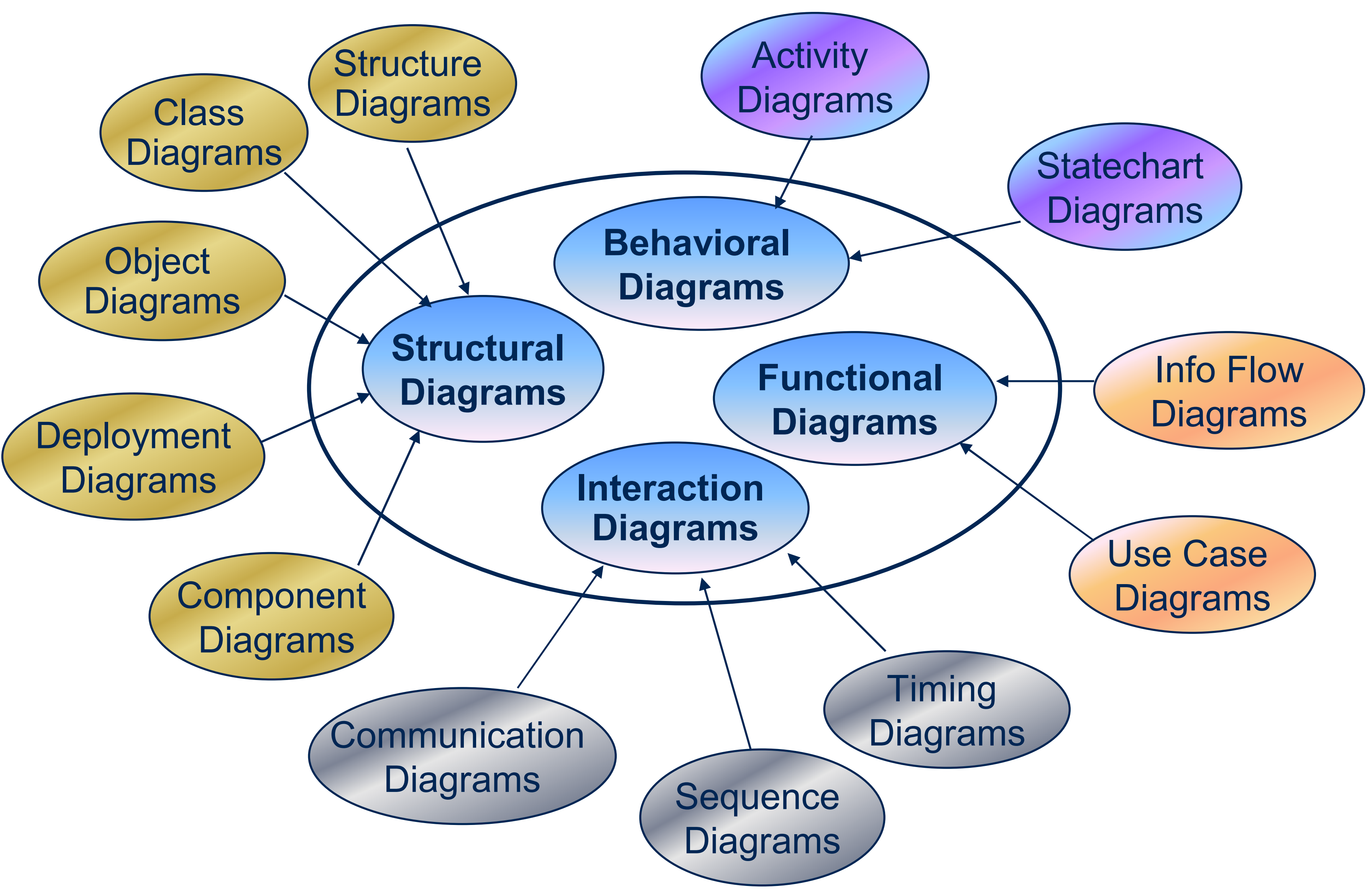
UML supports Key Technologies for Development



© Telelogic AB

Telelogic

UML 2 Diagrams



© Telelogic AB

How does UML apply to Real-Time?

- Real-Time UML is ***standard UML***
  - “UML is adequate for real-time systems” Grady Booch 1997
  - “Although there have been a number of calls to extend UML for the real-time domain … experience had proven this is not necessary.” Bran Selic, 1999 (Communications of the ACM, Oct 1999)
- Real-time and embedded applications
  - Special concerns about quality of service (QoS)
  - Special concerns about low-level programming
  - Special concerns about safety and reliability
- Real-Time UML is about applying the UML to meet the specialized concerns of the real-time and embedded domains

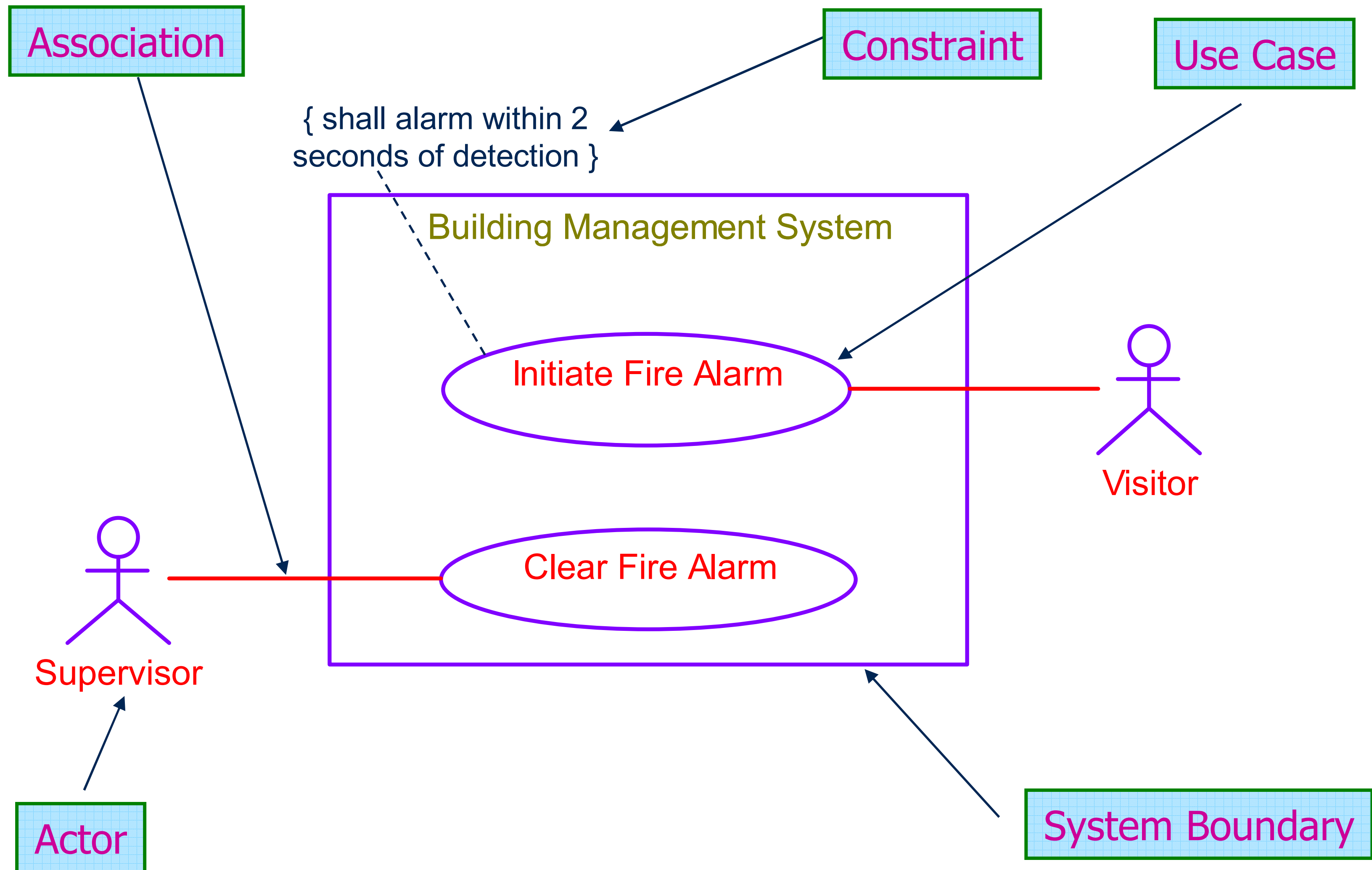
© Telelogic AB

How do we capture requirements using UML?

Use Case Modeling

© Telelogic AB

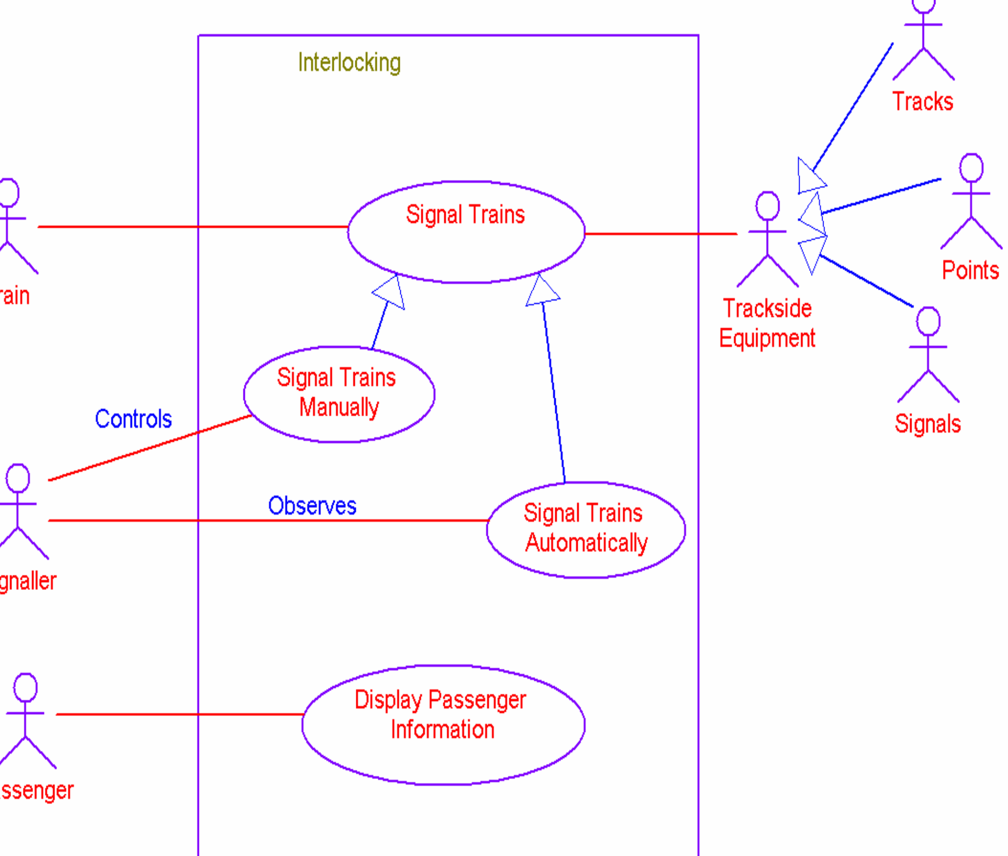
Basic Use Case Syntax



© Telelogic AB

A Use Case …

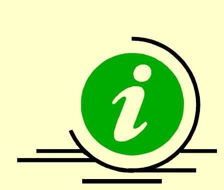
- Is a ***named operational capability of a system***
  – Why the user interacts with the system
- Returns a result visible to one or more actors
- Does not reveal or imply internal structure of the system



© Telelogic AB

A Use Case Is Used to

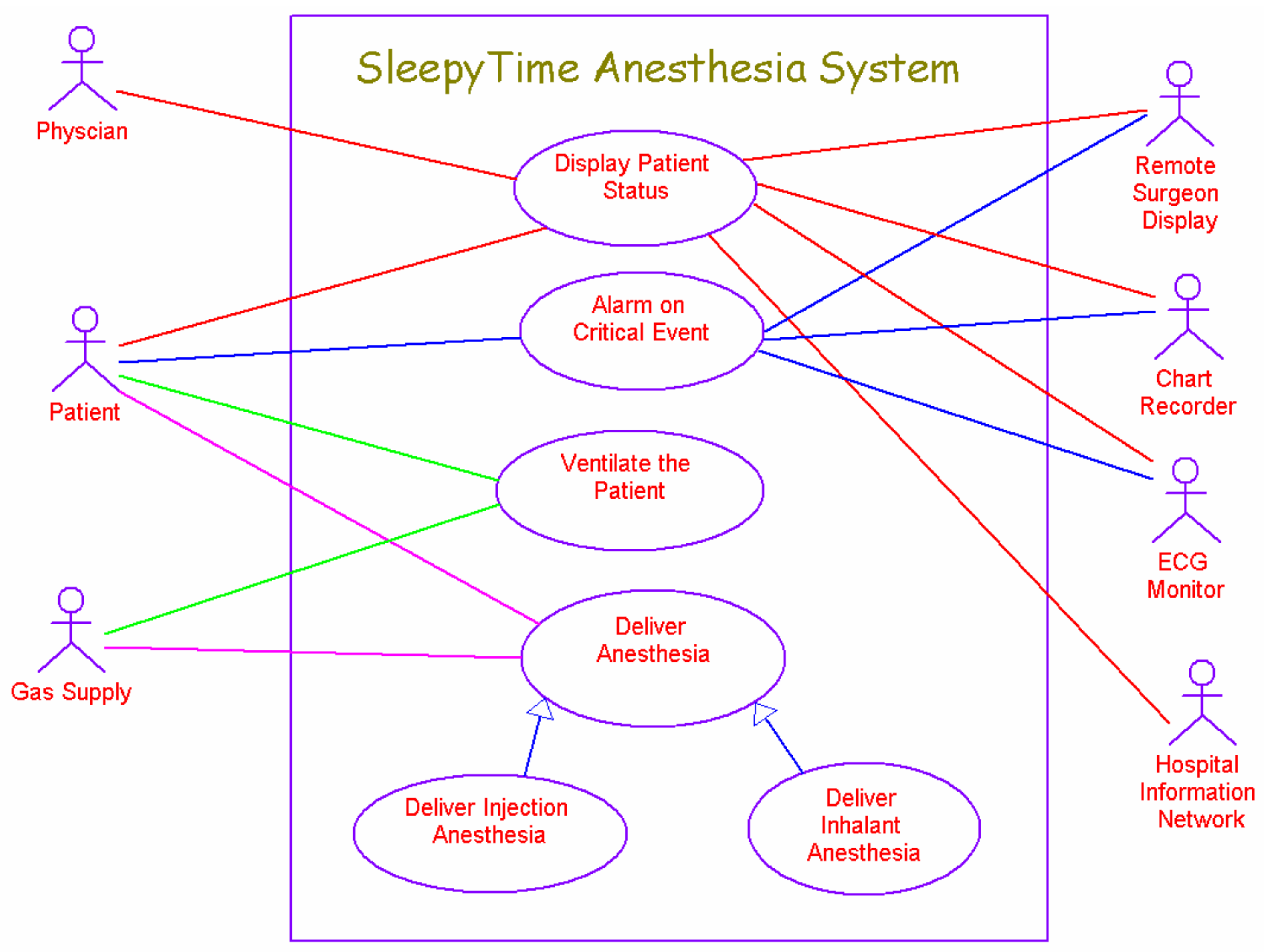
- Capture requirements of a system
  - Functional requirements
    - What the system does
  - Quality of Service (QoS)
    - *How well* the system does it:
      - Worst-case execution time
      - Average execution time
      - Throughput
      - Predictability
      - Capacity
      - Safety
      - Reliability



Functional requirements are modeled as use cases, sequence diagrams, activity diagrams or statecharts

© Telelogic AB

System Use Cases



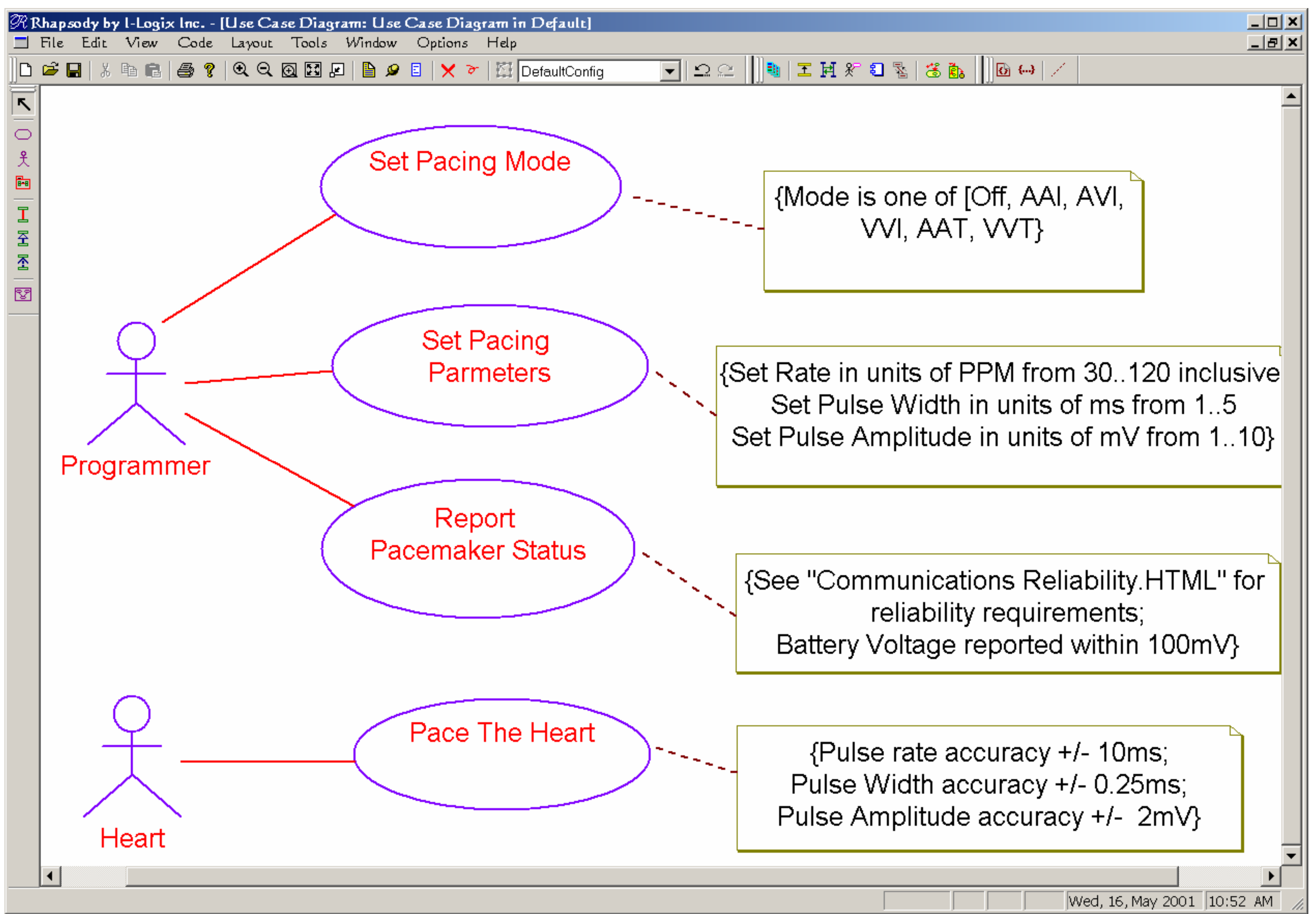
© Telelogic AB Telelogic

Use Cases Are Not ..

- A functional decomposition model of the system internals – that's HOW
  - They do not capture HOW in any way
  - They do not capture anything the Actor does outside of the System
  - How do I capture HOW?
    - Use Object Model Diagrams to capture Static Structure
    - Use Sequence Diagrams and State Diagrams (or Activity Diagrams) to capture Dynamic Behavior

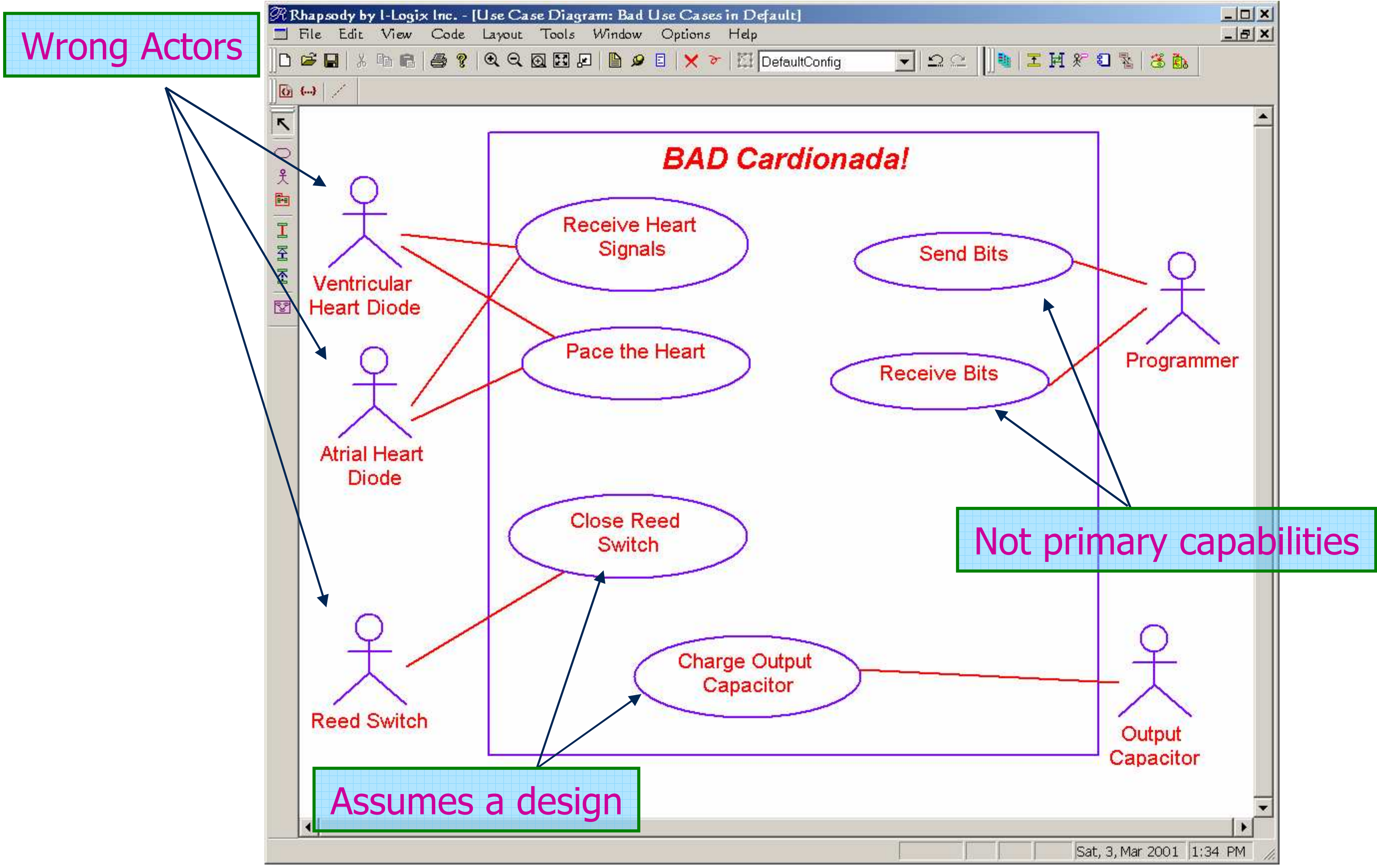
© Telelogic AB

Good Example: Cardionada



 © Telelogic AB Telelogic

Bad Example: Cardionada-NADA



© Telelogic AB Telelogic

Things to avoid

- Single messages are not use cases!
  - Use cases represent a large number of different scenarios
  - Each scenario has many (possibly hundreds) of messages
- Low-level interfaces are not use cases
  - Low-level interfaces are *means* to realize use cases
  - LLI’s are subsumed within actual use cases
  - A use case should map to a *reason* for the user to interact with the system, *not the means by which it occurs*

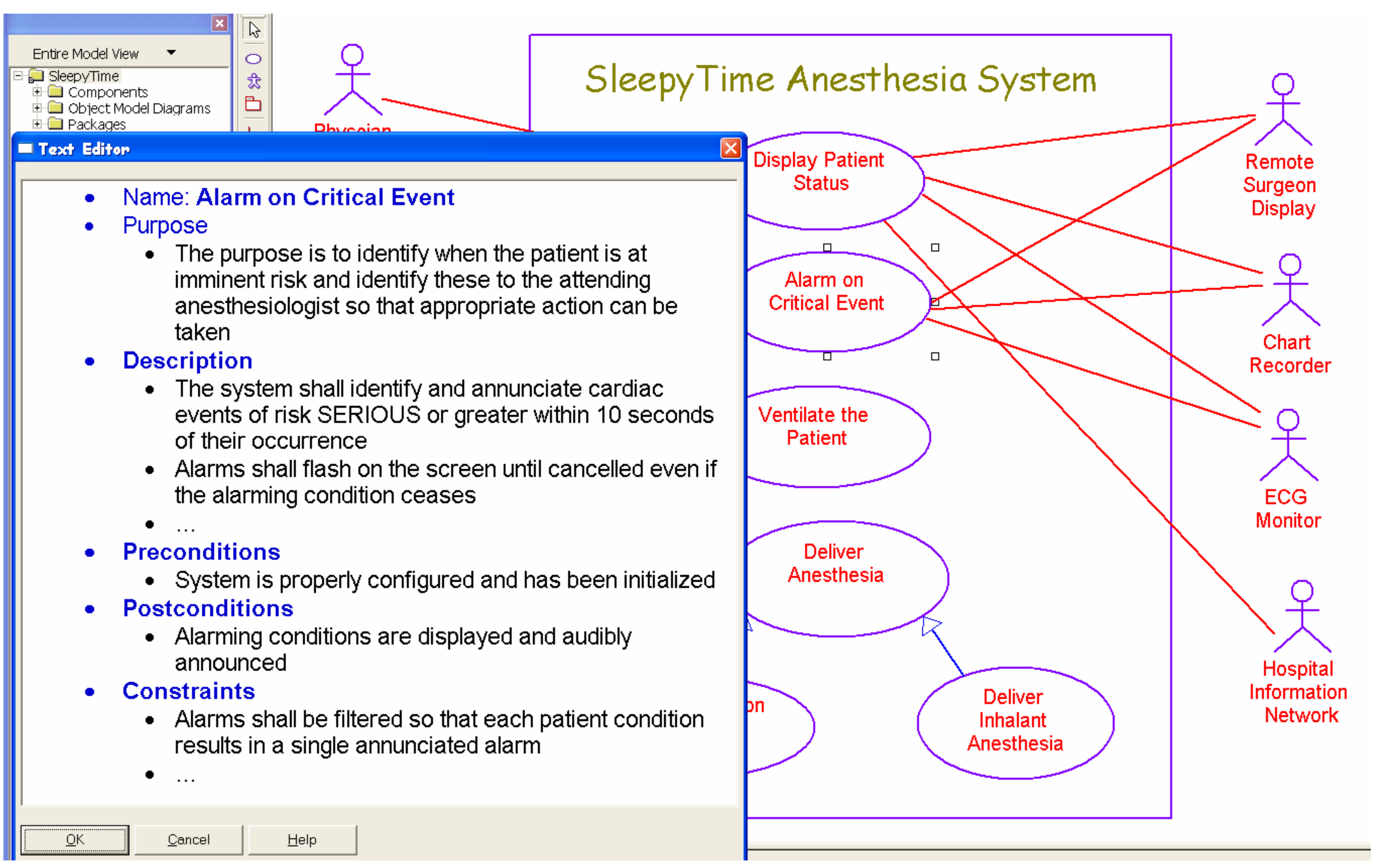
© Telelogic AB Telelogic

Detailing Use Cases

- By operational example
  - Use scenarios captured via sequence diagrams
    - Each scenario capture a specific path through the use case
  - Partially constructive
  - Infinite set, so you must select which are interestingly different
  - Non-technical users can understand scenarios
- By specification
  - Use an informal (e.g. English) and/or formal (e.g. statechart) to represent *all possible* paths
  - Fully constructive
  - Non-technical readers might not understand formal specs

© Telelogic AB

Use Case Description



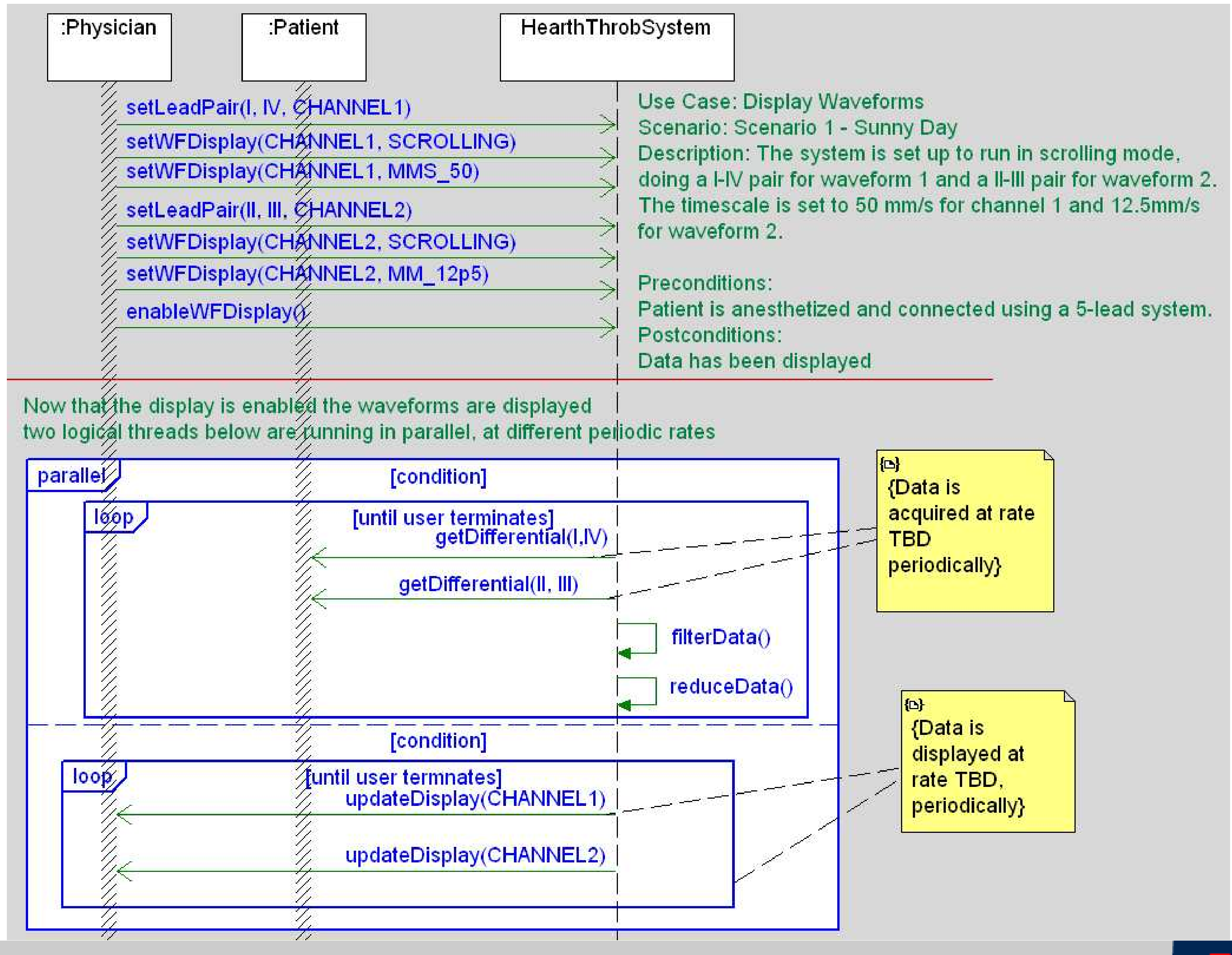
© Telelogic AB
Telelogic

Detailing Use Cases: Scenarios

- A typical system has one dozen to a few dozen use case
- Each use case typically has a few to several dozen scenarios of interest
- Scenarios capture a specific actor-system interaction
  - protocols of interaction
  - permitted sequences of message flow
  - collaboration of structural elements
  - show typical or exceptional paths through the use case

© Telelogic AB

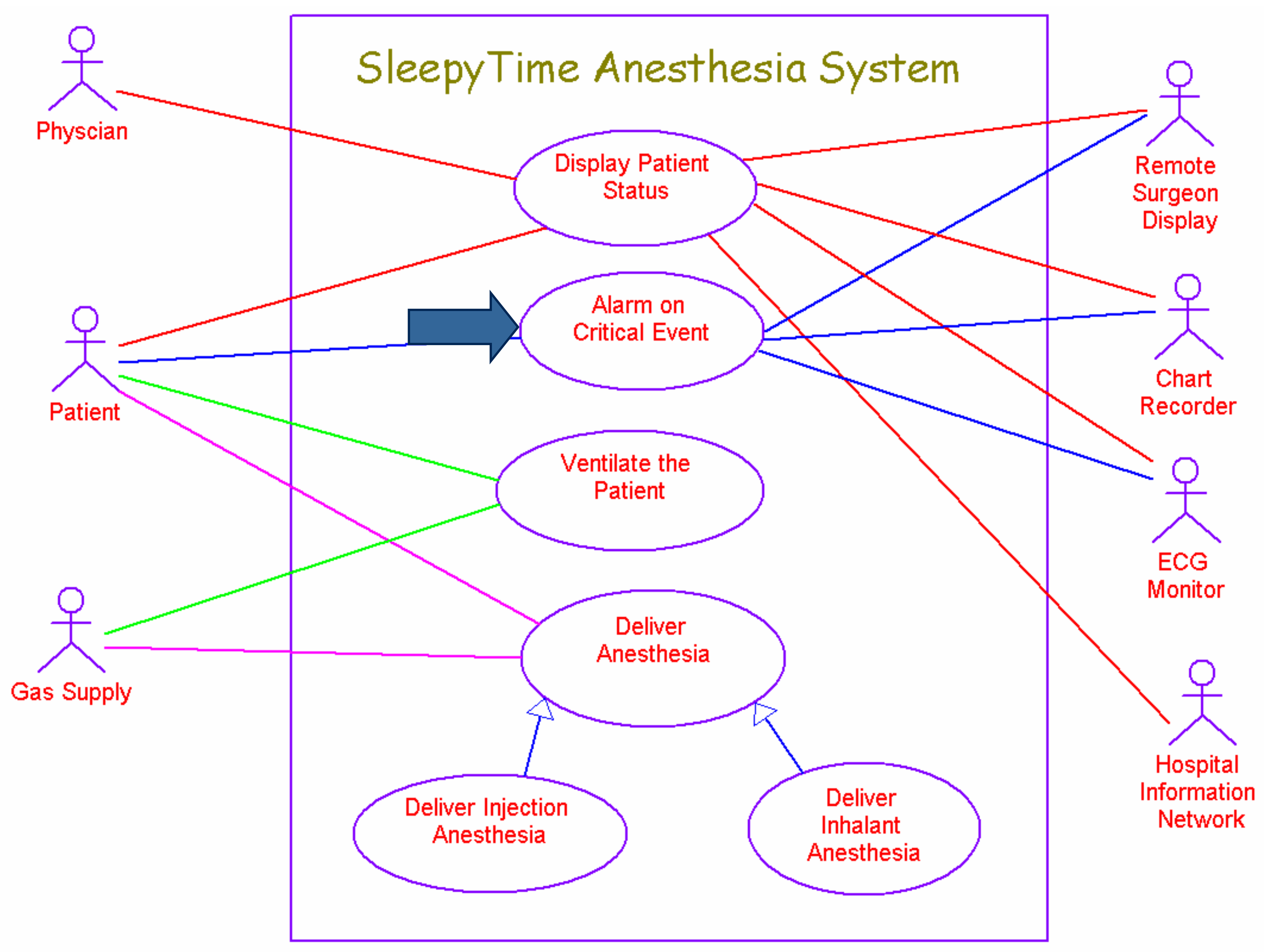
Example: Use Case Sequence Diagram



© Telelogic AB

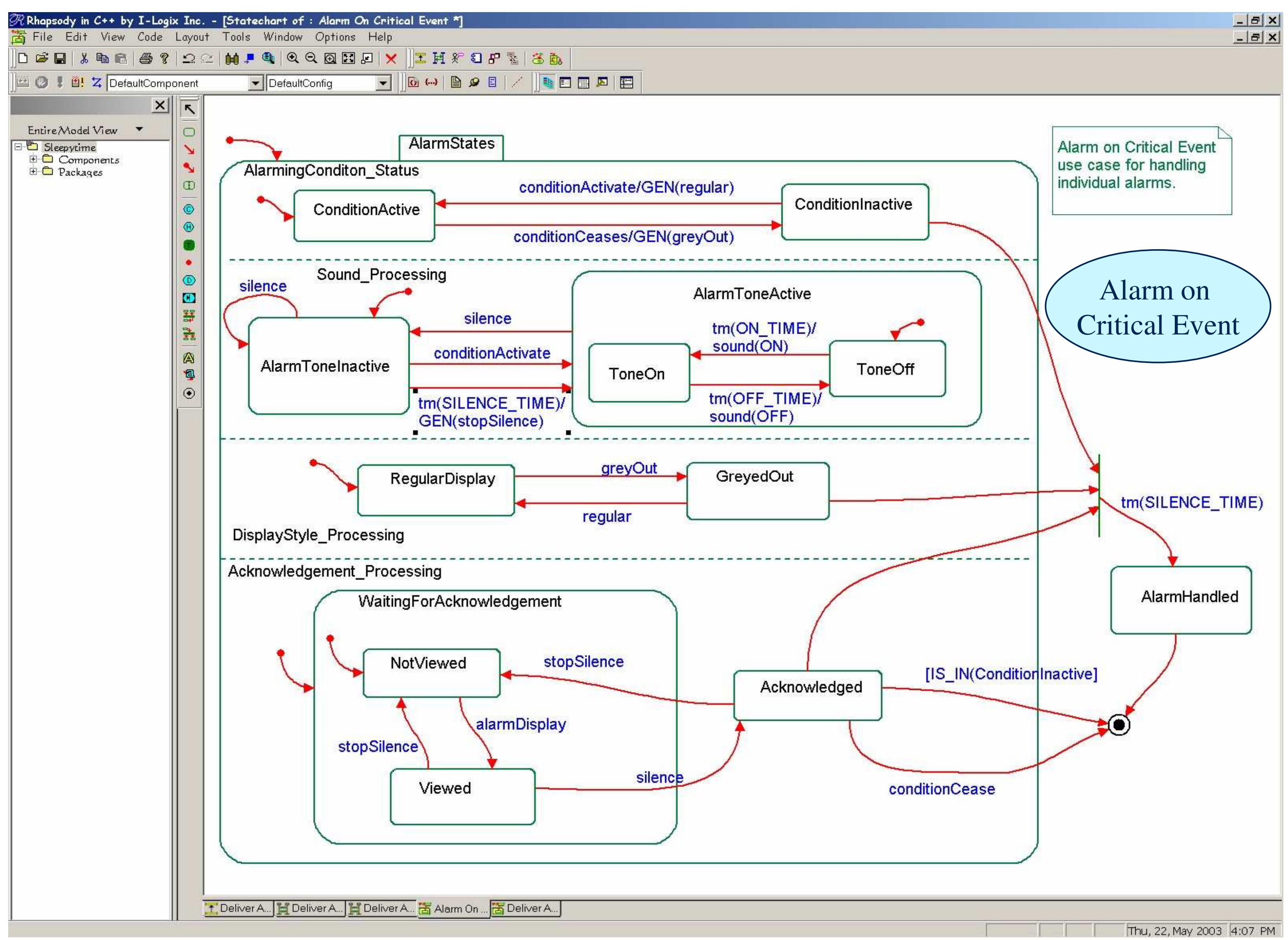
Telelogic

Detailing Use Cases: Statecharts



© Telelogic AB

Detailing Use Cases: Statecharts



© Telelogic AB
Telelogic

How do we describe structure using UML?

© Telelogic AB

Objects, Classes and Interfaces

- An **object** is a run-time entity that occupies memory at some specific point in time
  - Has behavior (methods)
  - Has data (attributes)
- A **class** is a design-time specification that defines the structure and behavior for a set of objects to be created at run-time.
  - Specifies behavior implementation (methods)
  - Specifies data (attributes)
- An **interface** is a design time concept that specifies the messages a class receives (“provided”) or uses (“required”)
  - Specifies behavior only (operation implementation)
  - May have *virtual* attributes (no implementation)
  - May have a *protocol* state machine (no actions)

© Telelogic AB

What is an *Object* ?

- An object is one of the common building blocks in a UML model.
  - Software: It can represent a system, a subsystem or a specific software element in a concrete programming language.
  - Systems: It can represent a real-world system, subsystem, or element
- Several definitions are available:
  - An object is a real-world or conceptual thing that has autonomy
  - An object is a cohesive entity consisting of data and the operations that act on those data
  - An object is a thing that has an interface that enforces protection of the encapsulation of its internal structure

© Telelogic AB

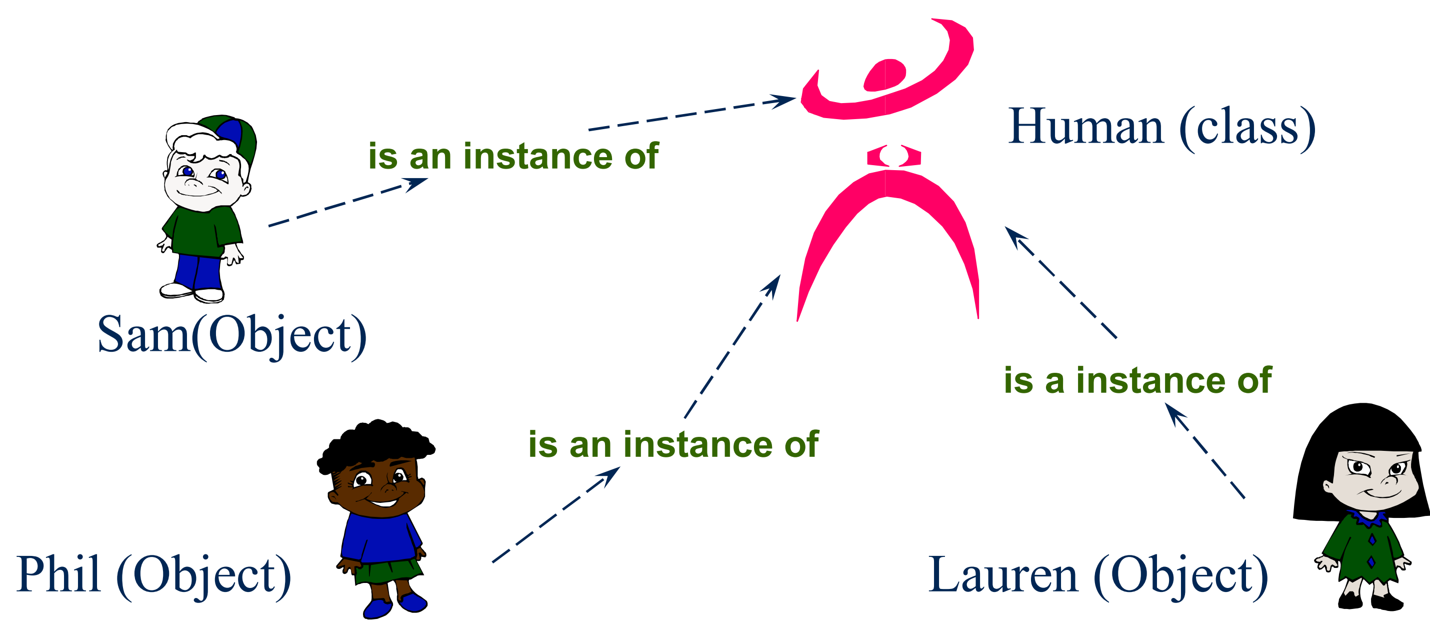
Objects can be …

- Software things
  - Occupy memory at some point in time
  - E.g. CustomerRecord, ECGSample, TextDisplayControl
- Electronic things
  - Occupy physical space at some point in time
  - E.g. Thermometer, LCDDisplay, MotionSensor, DCMotor
- Mechanical things
  - Occupy physical space at some point in time
  - E.g. WingSurface, Gear, Door, HydraulicPress
- Chemical things
  - Occupy physical space at some point in time
  - E.g. Battery, GasMixture, Halothane
- System things
  - Occupy space at some point in time
  - E.g. PowerSubsystem, RobotArm, Space Shuttle

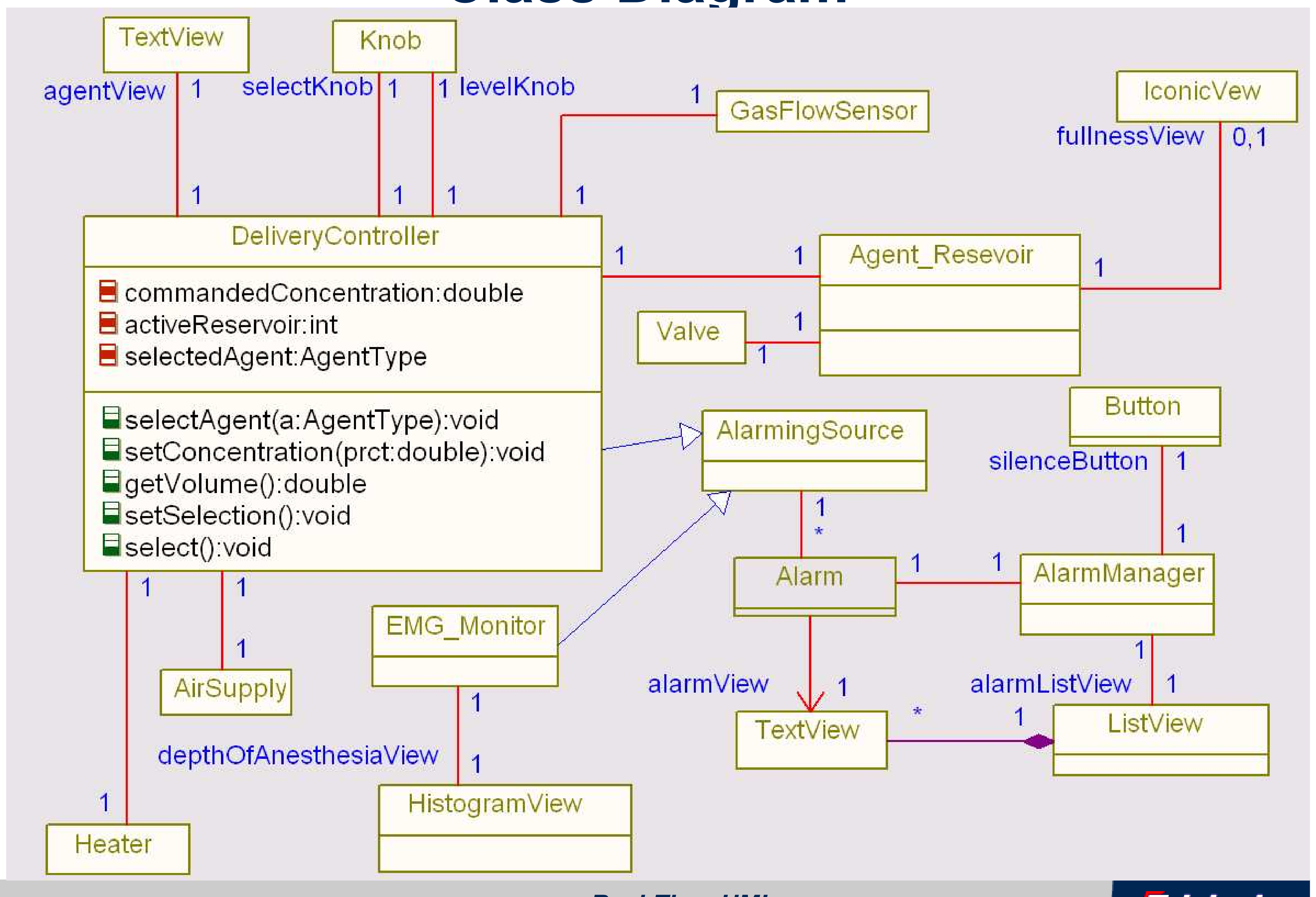
© Telelogic AB

Classes

- A Class is the ***definition*** or ***specification*** of an object
- An object is an ***instance*** of a class
- An object has the attributes and behaviours defined by its class
- It is common to have many instances of a class at the same time



Class Diagram



© Telelogic AB

Structural Diagrams

- Diagrams serve many purposes
  - System model-capture & specification
  - View aspects of system design
  - Provide basis for communication and review
- Diagrams bring 2 things to the design process
  - Represent different aspects of design, e.g.
    - Functional
    - Structural
    - Behavioral
    - Quality of Service
  - Show aspects at different levels of abstraction
    - System
    - Subsystem
    - Component
    - “Primitive” class

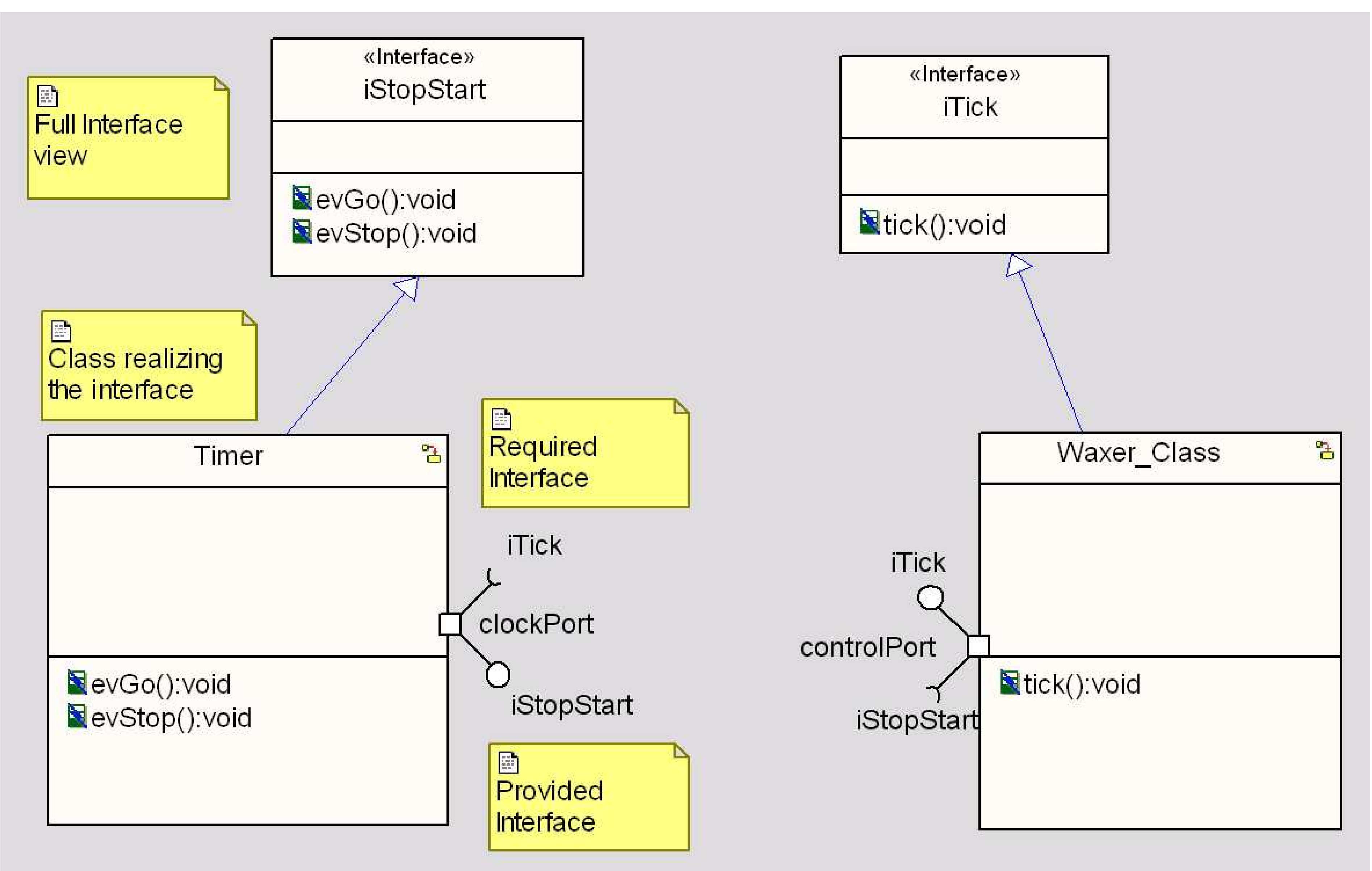
© Telelogic AB

Interfaces

- UML interfaces specify operations or event receptions
- UML Interfaces have no implementation
  - No methods
  - No attributes
- Classes realize an interface by
  - providing a method (implementation) of an operation or
  - specifying an event reception on a state machine
- A class that realizes an interface is said to be compliant with that interface
  - Classes may realize any number of interfaces
  - Interfaces may be realized by any number of classes

© Telelogic AB

Interfaces



© Telelogic AB

Relationships

- Relationships allow objects to communicate at run-time
- Objects may use the facilities of other objects with an ***association***
- There are two specialized forms of association:
  - Objects may contain other objects with an ***aggregation***
  - Objects may *strongly* aggregate others via ***composition***
- Classes may derive attributes and behaviours from other classes with a ***generalization***
- Classes may depend on others via a ***dependency***

© Telelogic AB

Associations

- Associations allow instances of classes to communicate at run-time
  - Instances of associations are called ***links***
  - Links may come and go during execution
- Denotes one object using the facilities of another
- Lifecycles of the objects are independent
- Allows objects to provide services to many others

© Telelogic AB

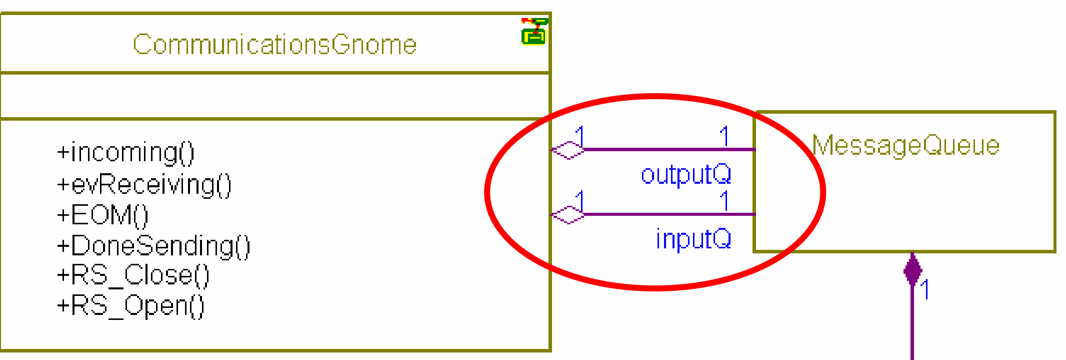
Associations

- Associations may have labels
  - This is the “name” of the association
- Associations may have role names
  - Identifies the role of the object in the association
- Associations may indicate multiplicity
  - Identifies the number of instances of the class that participate in the association
    - N
    - *
    - 1..[n | *]
- Associations may indicate *navigation* with an open arrowhead
  - Unadorned associations are assumed to be bi-directional
  - Most associations are unidirectional

© Telelogic AB

Aggregation

- Indicated by a hollow diamond
- “Whole-part” relationship
  - Denotes one object logically or physically contains another
- “Weaker” form of aggregation. Nothing is implied about
  - Navigation
  - Ownership
  - Lifetimes of participating objects

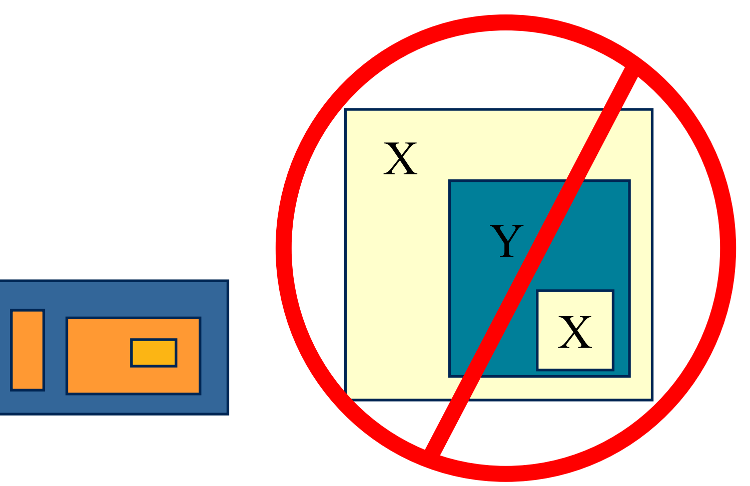


© Telelogic AB

Telelogic

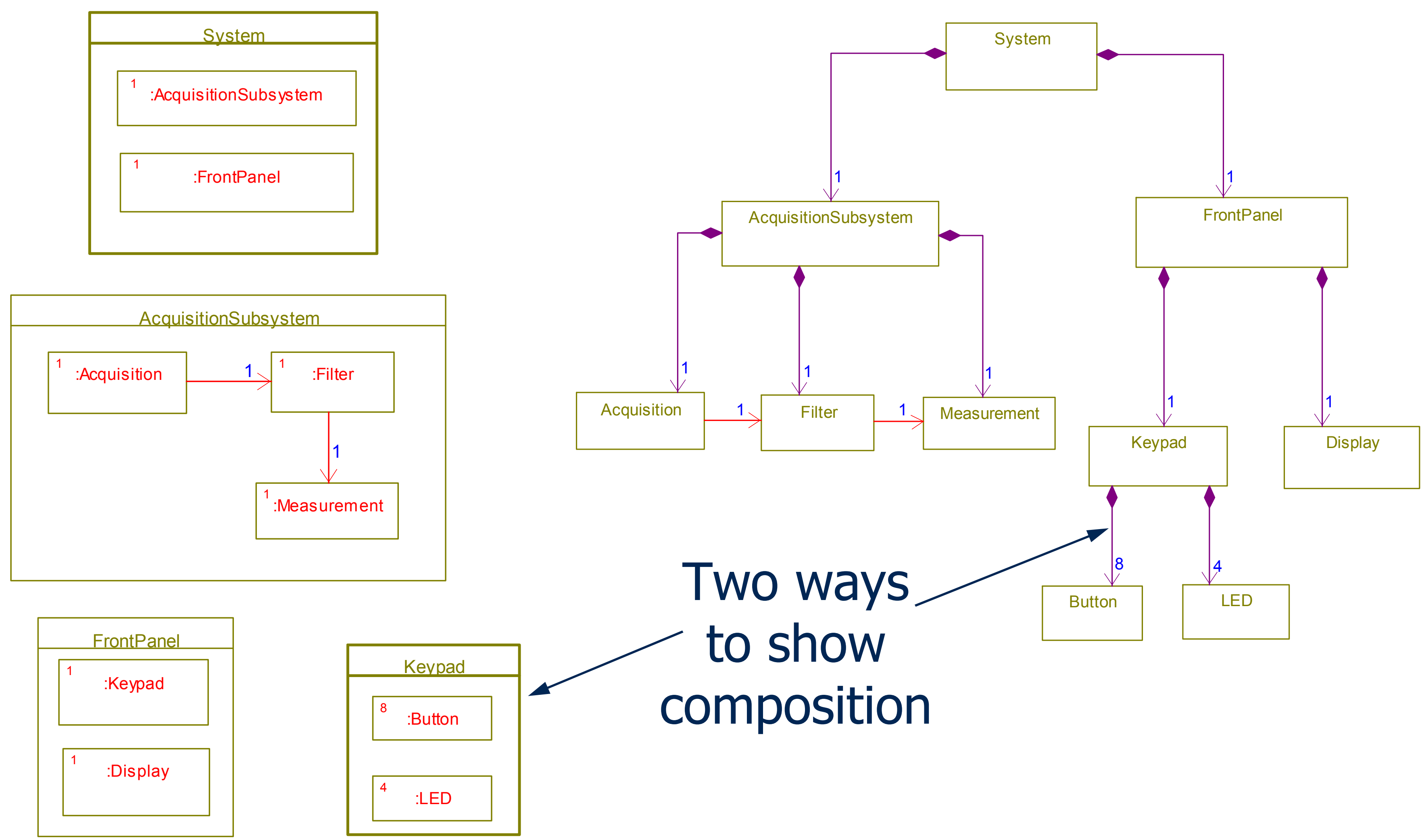
Composition

- Indicated by containment or a filled aggregation diamond
- Whole both creates and destroys part objects
- Composite is a higher level abstraction than the parts
  - Allows the class model to be viewed and manipulated at many levels of abstraction
- “Stronger” form of aggregation
  - Implies a multiplicity of no more than one with the whole
- Forms a tree with its parts



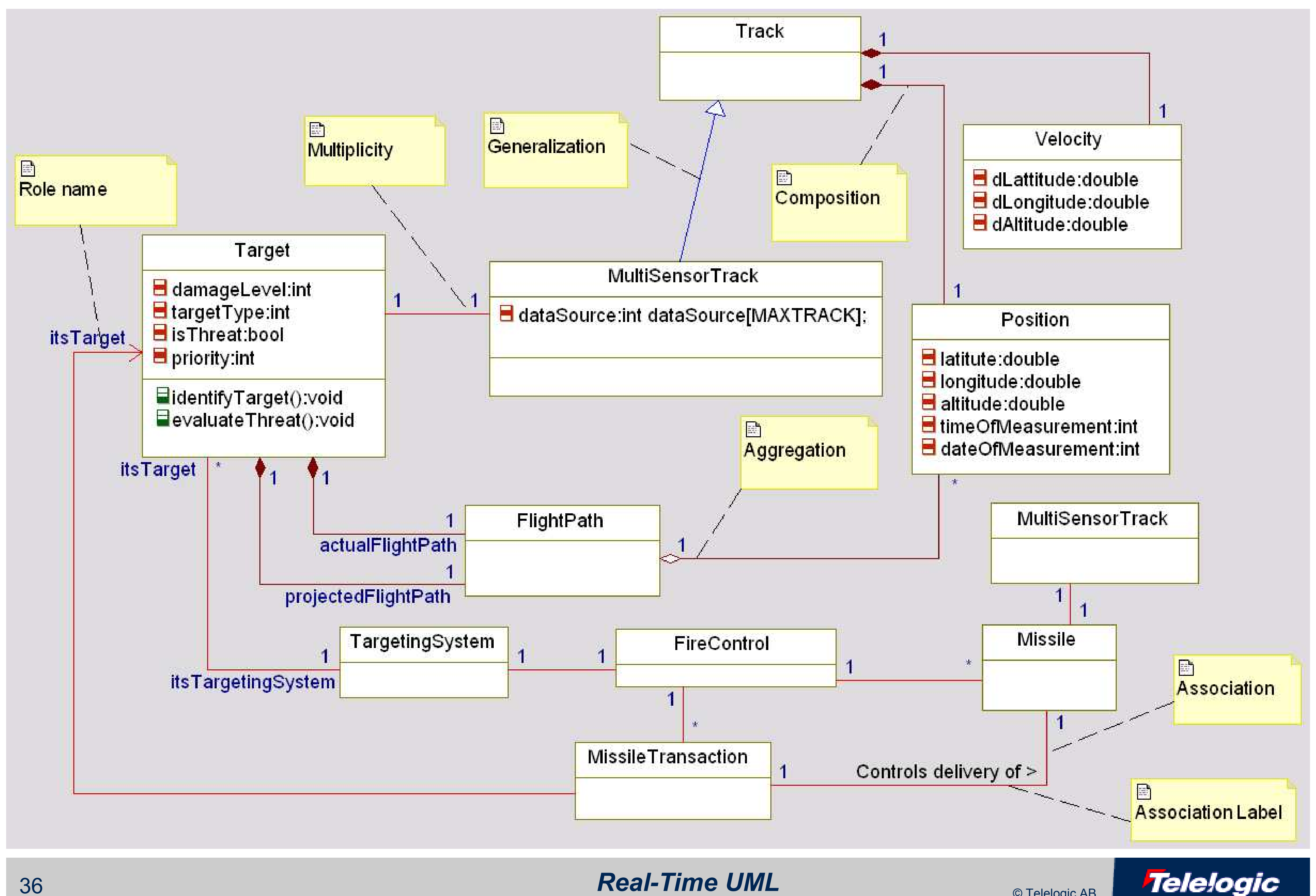
© Telelogic AB

Composition Example



© Telelogic AB
Telelogic

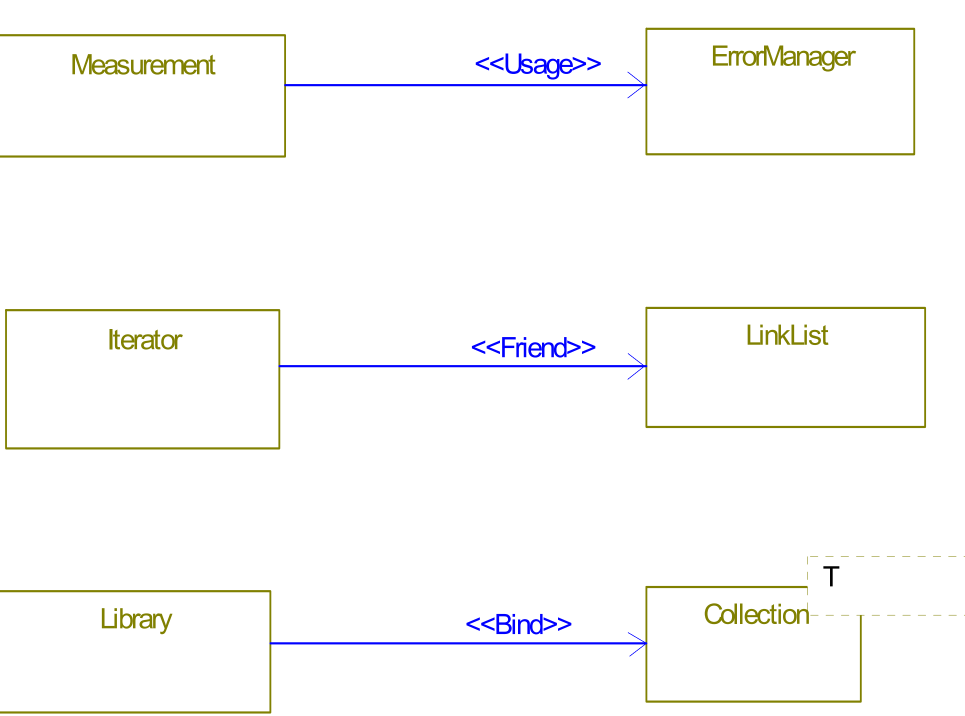
Relationships



© Telelogic AB
Telelogic

Dependencies

- A dependency can be used when a class has no direct relation to another class, but depends on it in some way.
  - A Dependency stereotype (<<>>) details how one item is dependent on the other
  - One class may depend on another to build an executable from code with a <<usage>> dependency
  - A Use Case may depend on a class that realizes its required functionality



© Telelogic AB Telelogic

Advanced Structural Concepts

 © Telelogic AB

Advanced Structural Concepts

- Flows
- Structure Classes
- Ports

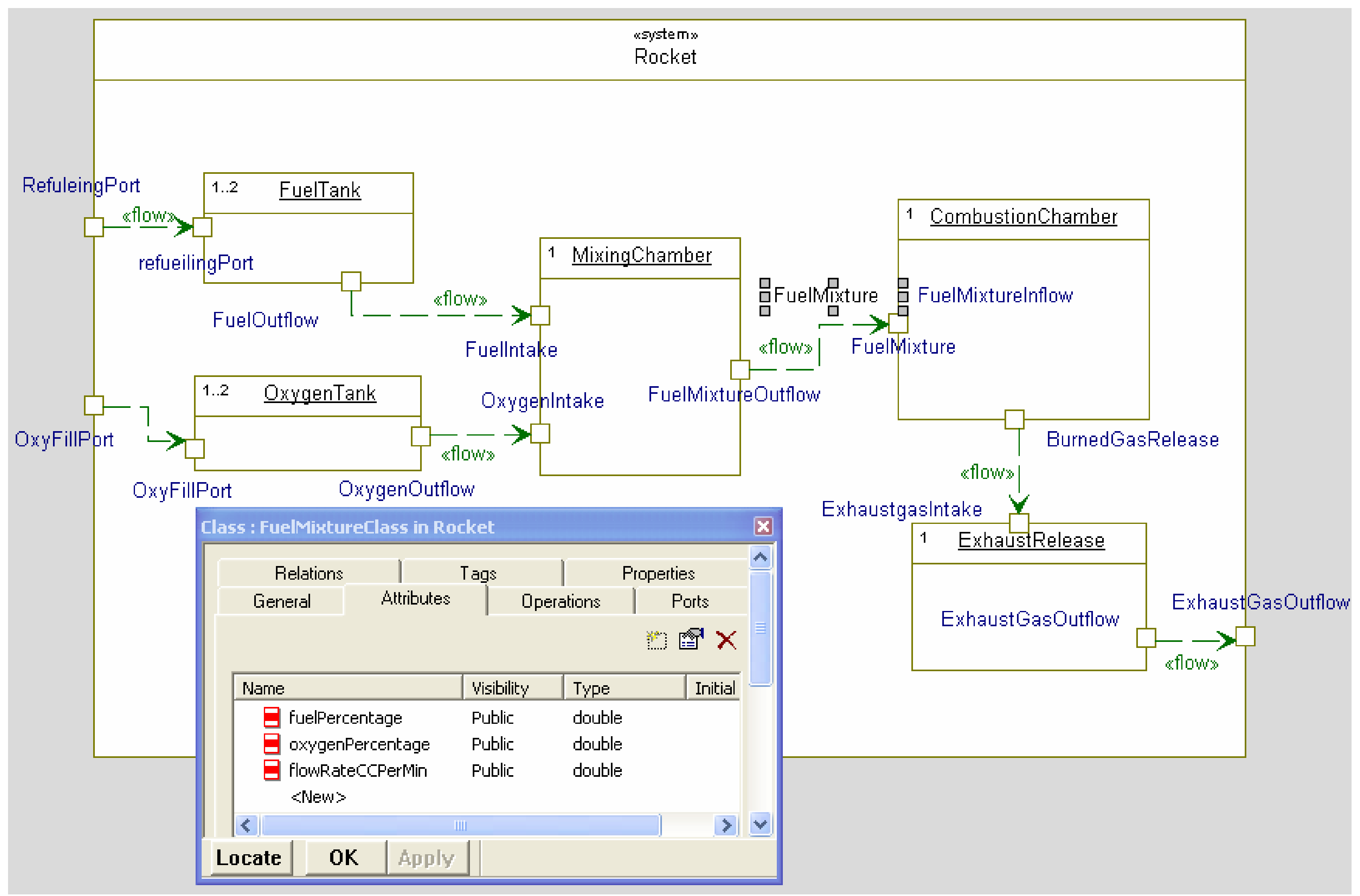
 © Telelogic AB

Information Flows

- *Information flows* are an abstract view of collaboration
  - Information items represent data
- Similar to data flow diagrams
  - Non-constructive
  - No sequence is shown
- Information Flows are details with data elements known as *Flow Items*
  - Flow Items can be primitive types, records containing other items, or even classes
- Operations and event receptions on the target class will realize information flow
  - Information flow items will be parameters or return values of the operations

© Telelogic AB

Item Flows



© Telelogic AB

Structured Classes

- A *structured class* is a class that is composed of parts
  - The structured class is responsible for the creation and destruction of its parts
  - The structured class “owns” the parts via composition
- A *part* is an *object role* that executes in the context of the structured class
  - Parts are *typed* by classes
- Parts are connected via *connections*
  - Connections are contextualized links
  - The structured class links together the parts as necessary

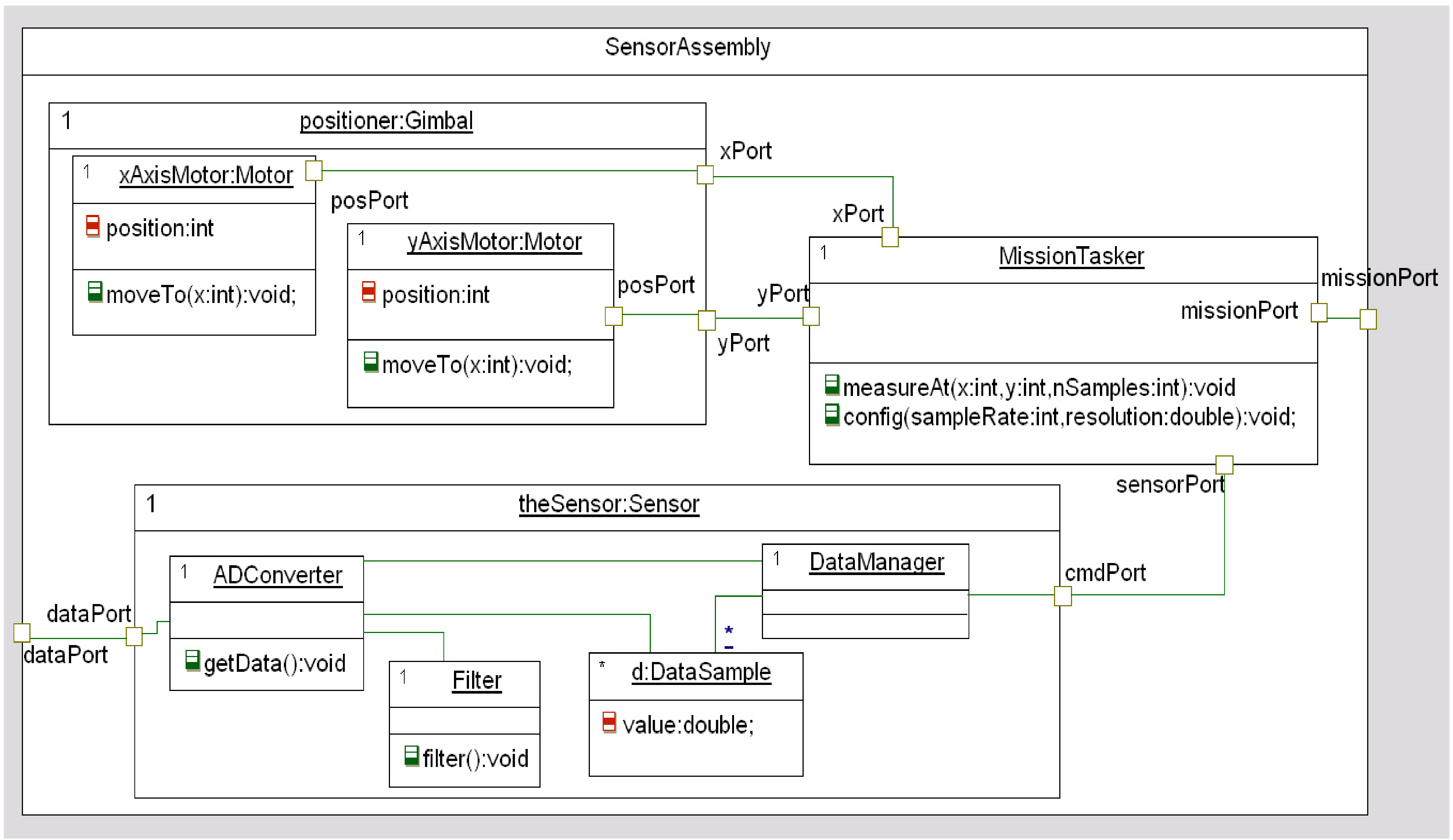
© Telelogic AB

Structured Classes

- PRIMARY USE: show systems at different levels of abstraction
  - Composite is at a higher level of abstraction than the part (e.g. system and subsystem)
  - Composite achieves its behavioral goals primarily through delegation
- Additional use: Populate simple classes into «active» classes for concurrency
- Additional use: Use composites to aid in system boot and object construction

© Telelogic AB

Structured Class



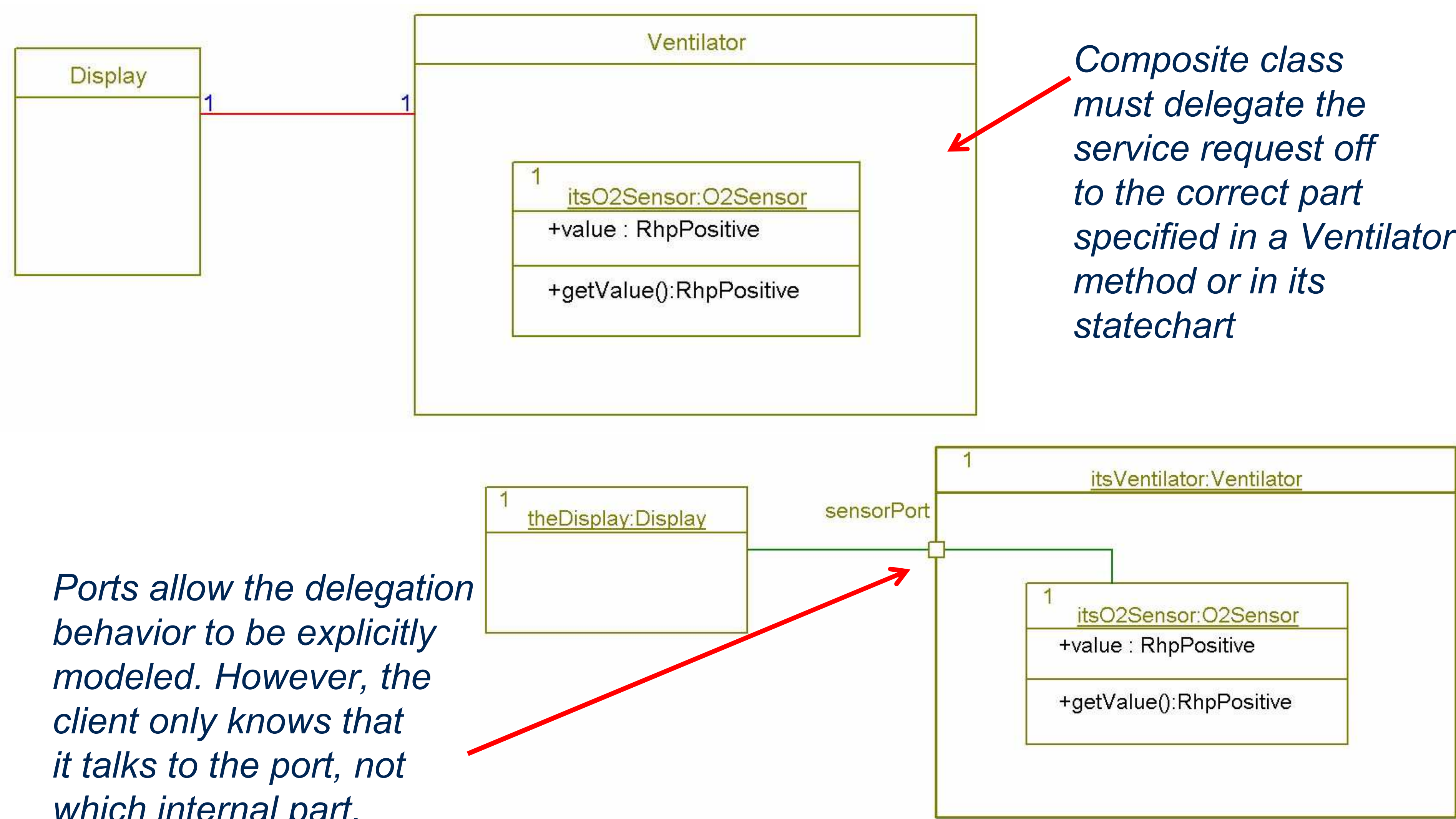
 © Telelogic AB Telelogic

Parts and Ports

- Ports are “named connection points”
- Ports allow a part to provide an interface across the boundary of its enclosing structured class without revealing to the client of the service the internal structure of the structured class
  - Client “talks to” the port without internal knowledge
- Ports are “typed by” their interfaces
  - Provided interfaces
  - Required Interfaces

© Telelogic AB

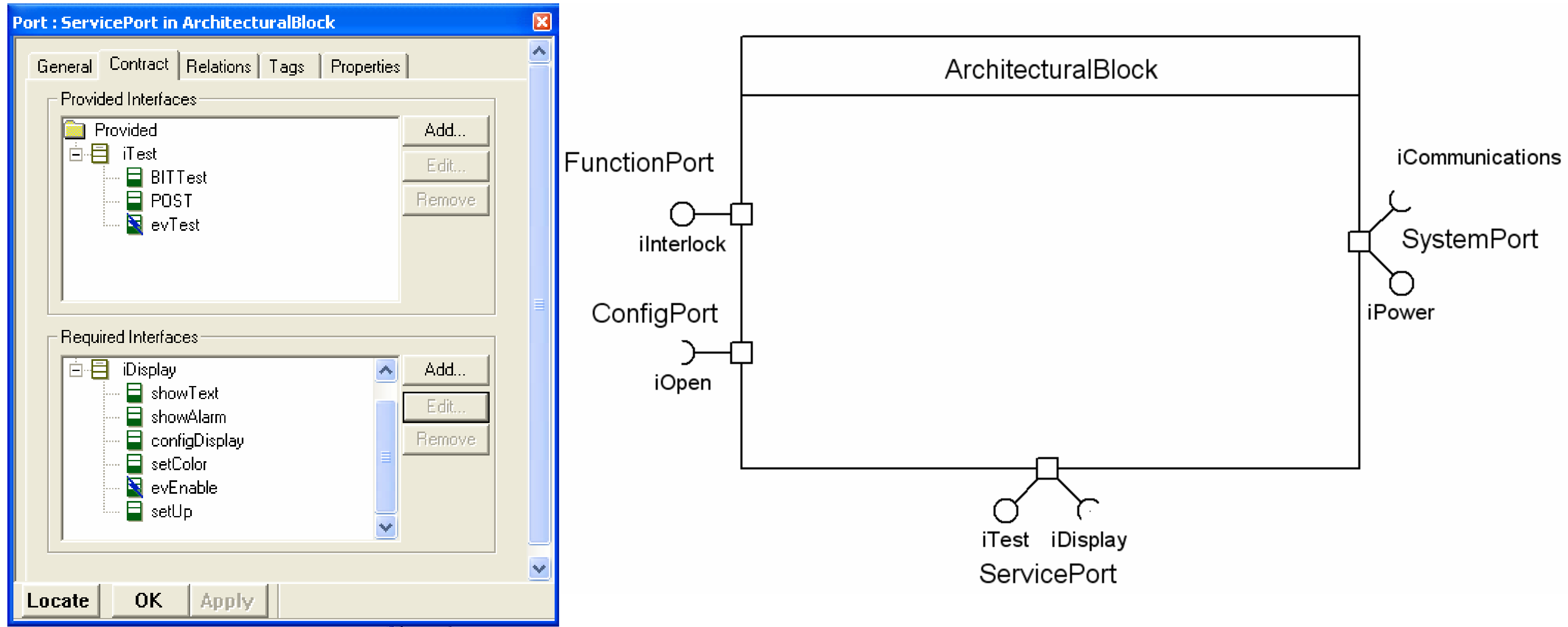
Parts and Ports



© Telelogic AB

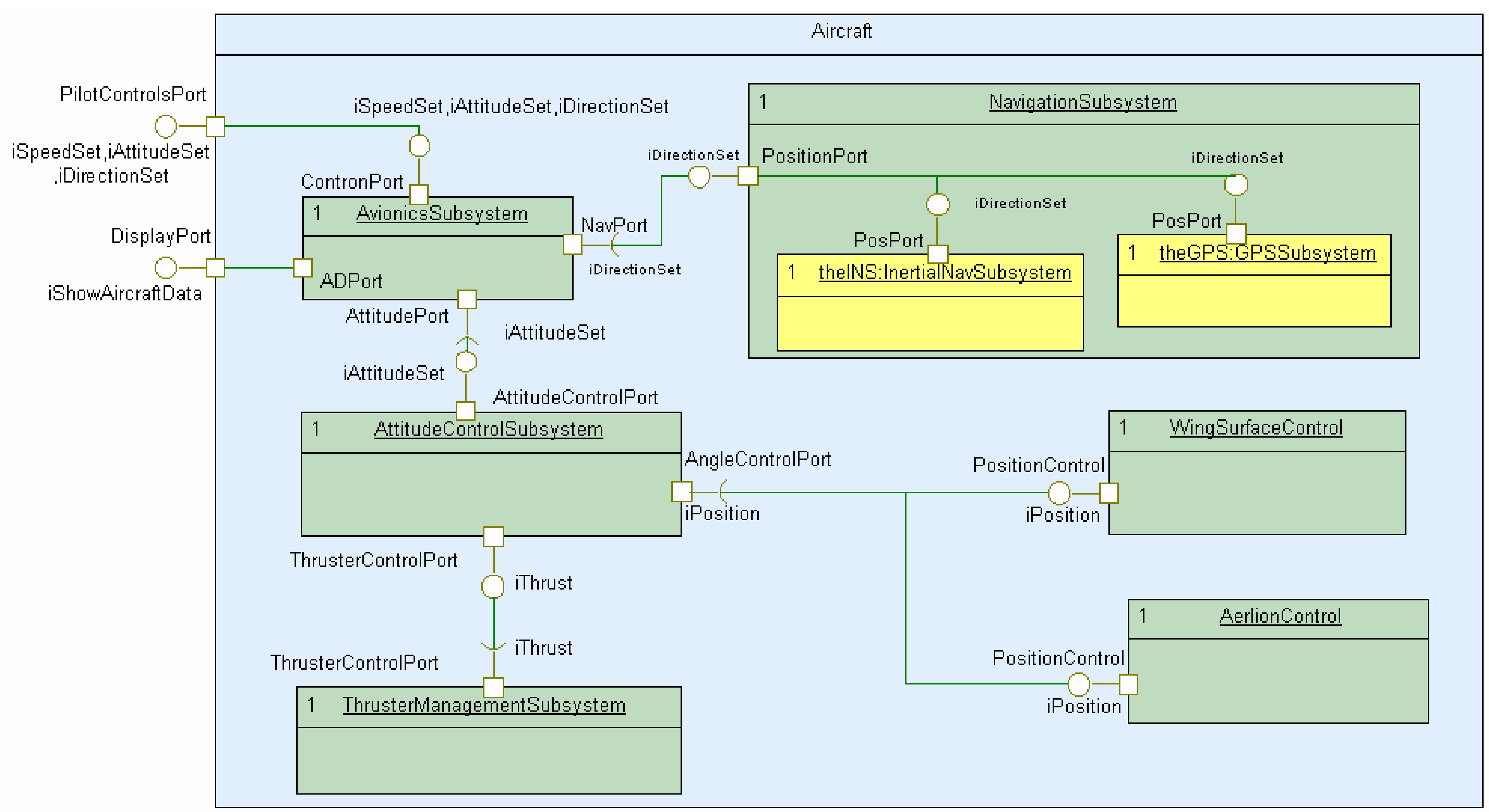
Interfaces with Ports

- Ports are typed by their interfaces
  - Provided interfaces specify services offered
  - Required interfaces specify services needed
  - A port may have either or both



© Telelogic AB

Port Example



© Telelogic AB

Telelogic

How do we describe behavior?

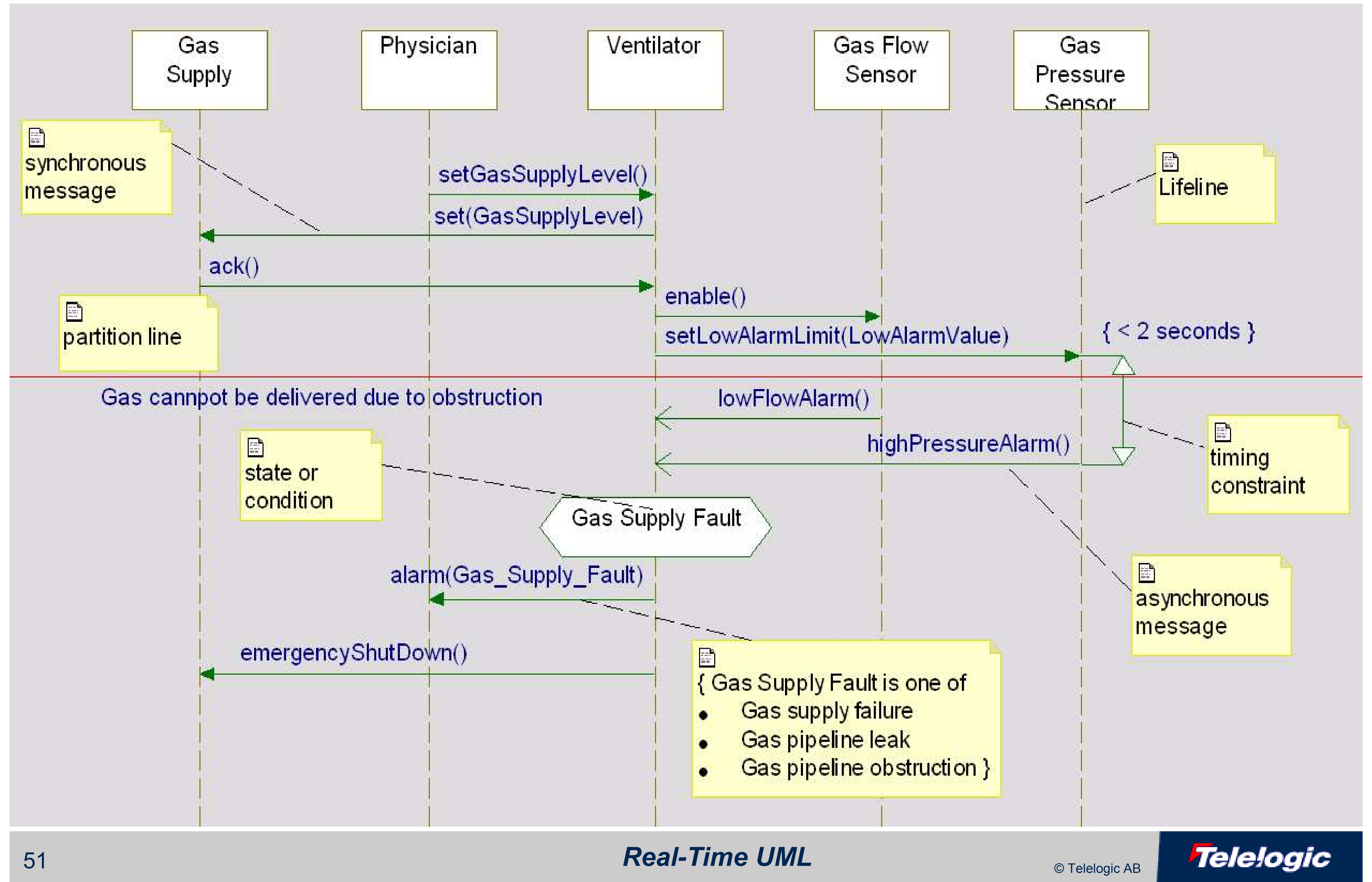
© Telelogic AB

Sequence Diagrams

- Sequence diagrams show the behavior of a group of instances (e.g. objects) over time. Instances may be
  - Objects (most common)
  - Use case instance
  - System
  - Subsystem
  - Actor
- Useful for
  - Capturing typical or exceptional interactions for requirements
  - Understanding collaborative behavior
  - Testing collaborative behavior

© Telelogic AB

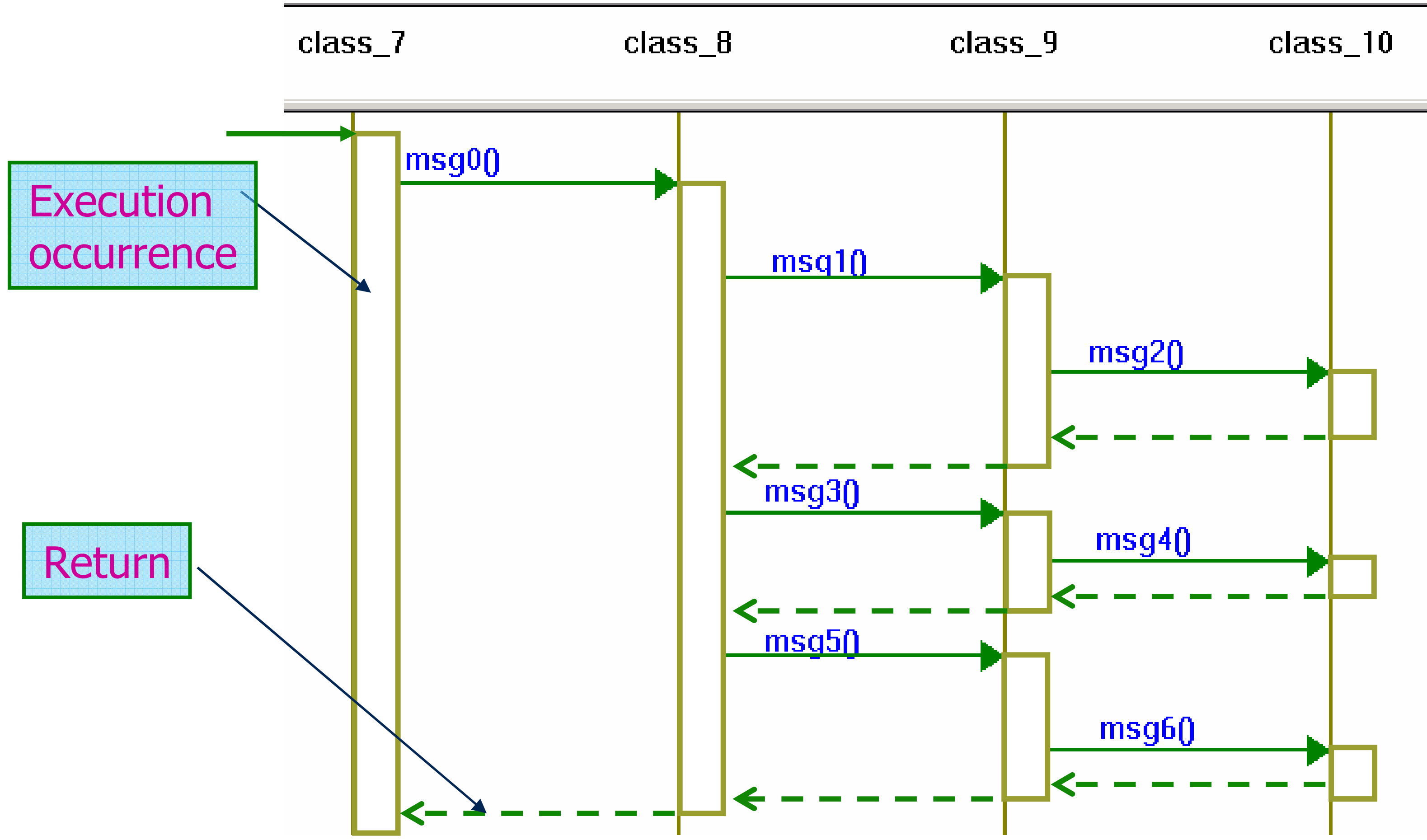
Sequence diagram : Basic Syntax



 © Telelogic AB

Special Case: Execution Occurrence

- Execution occurrences are useful for the *sequential* message exchanges



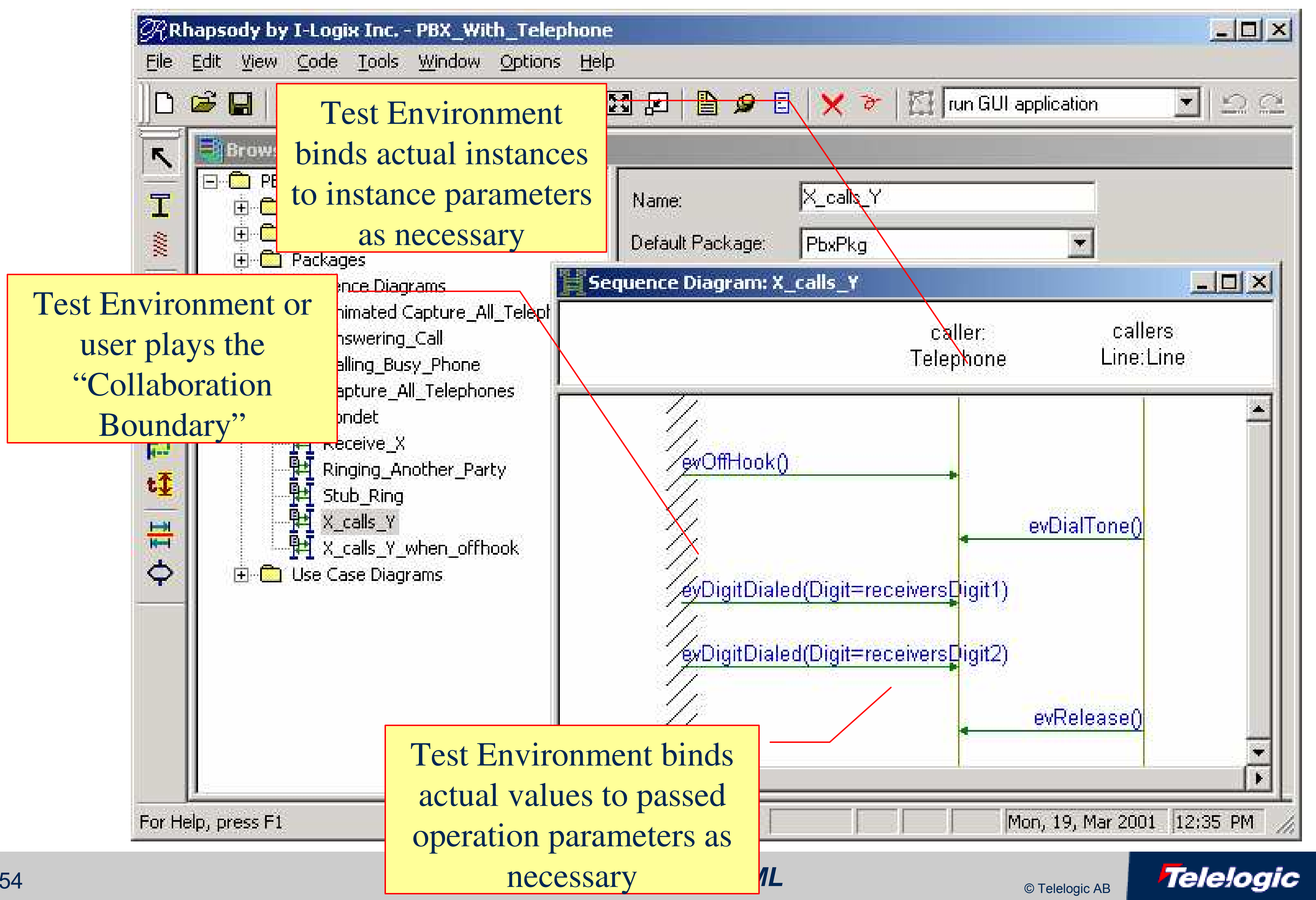
© Telelogic AB
Telelogic

Deriving Test Vectors from Scenarios

- A scenario is an example execution of a system
- A test vector is an example execution of a system with a known expected result
- Scenario → Test Vector
  - Capture preconditions
  - Determine test procedure
    - Identify causal messages and events
    - Instrument test fixtures to insert causal messages
  - Remove optional or "incidental" messages
  - Define pass/fail criteria
    - Identify effect messages / states
    - Postconditions
    - Required QoS

© Telelogic AB

Stimulating the System



© Telelogic AB

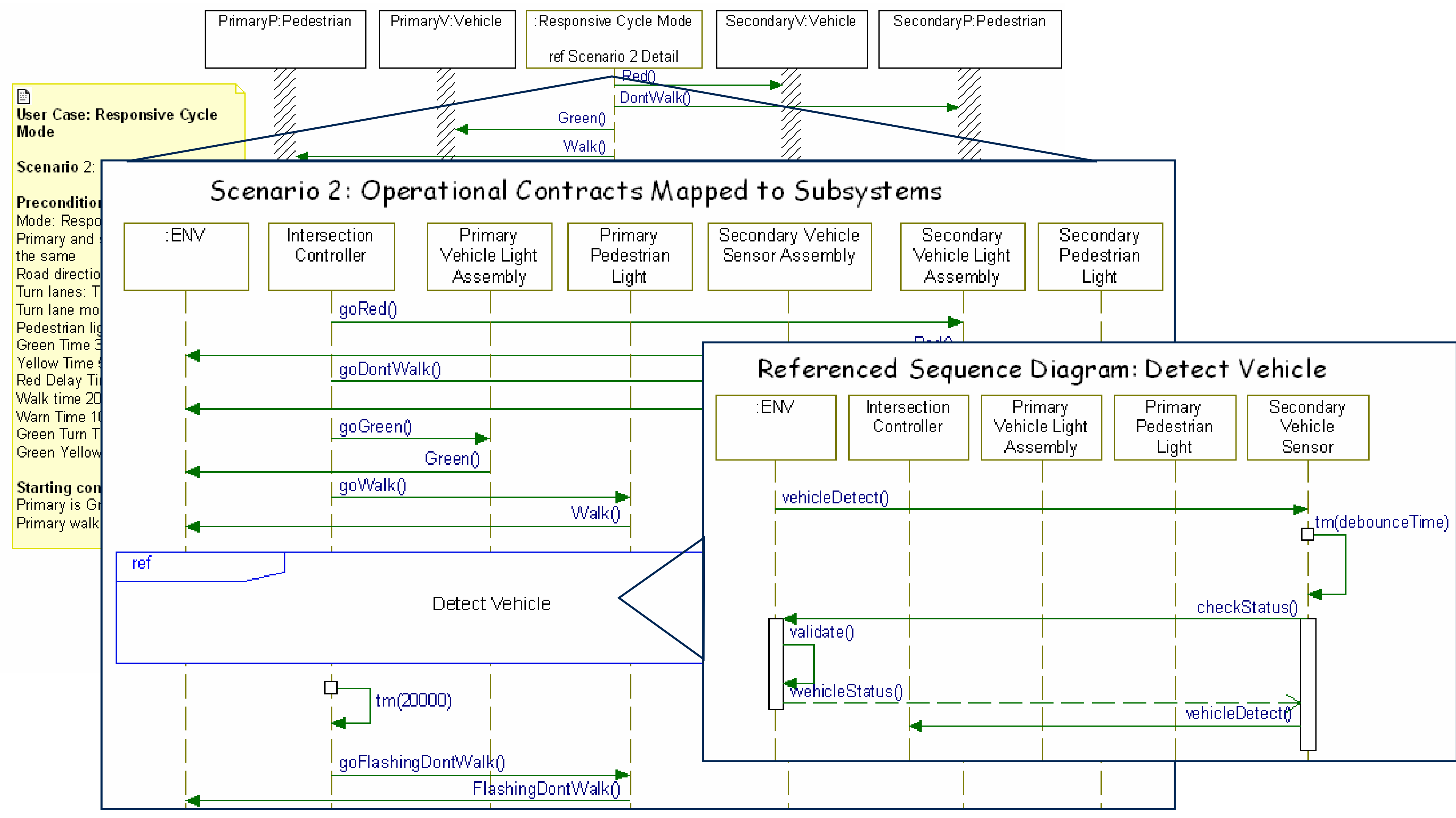
Telelogic

Decomposing Sequence Diagrams

- Sequence diagrams suffer from scalability problems when there are many
  - Lifelines
  - Messages
- UML 2 provides the ability to decompose sequence diagrams both vertically and horizontally
  - Lifelines may be decomposed into interactions (separate sequence diagrams
  - Interaction fragments may be referenced on another diagram

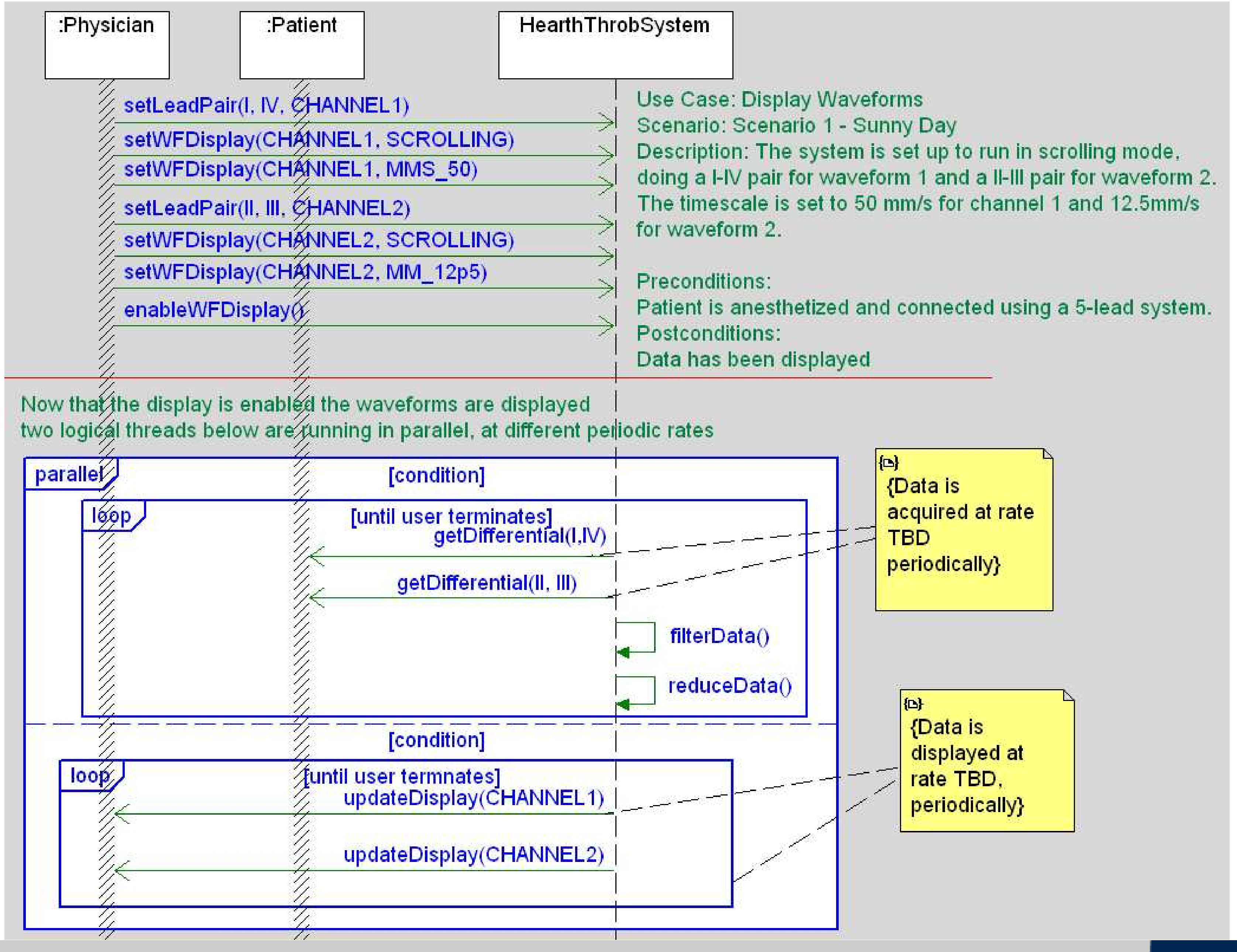
© Telelogic AB

Sequence Diagrams



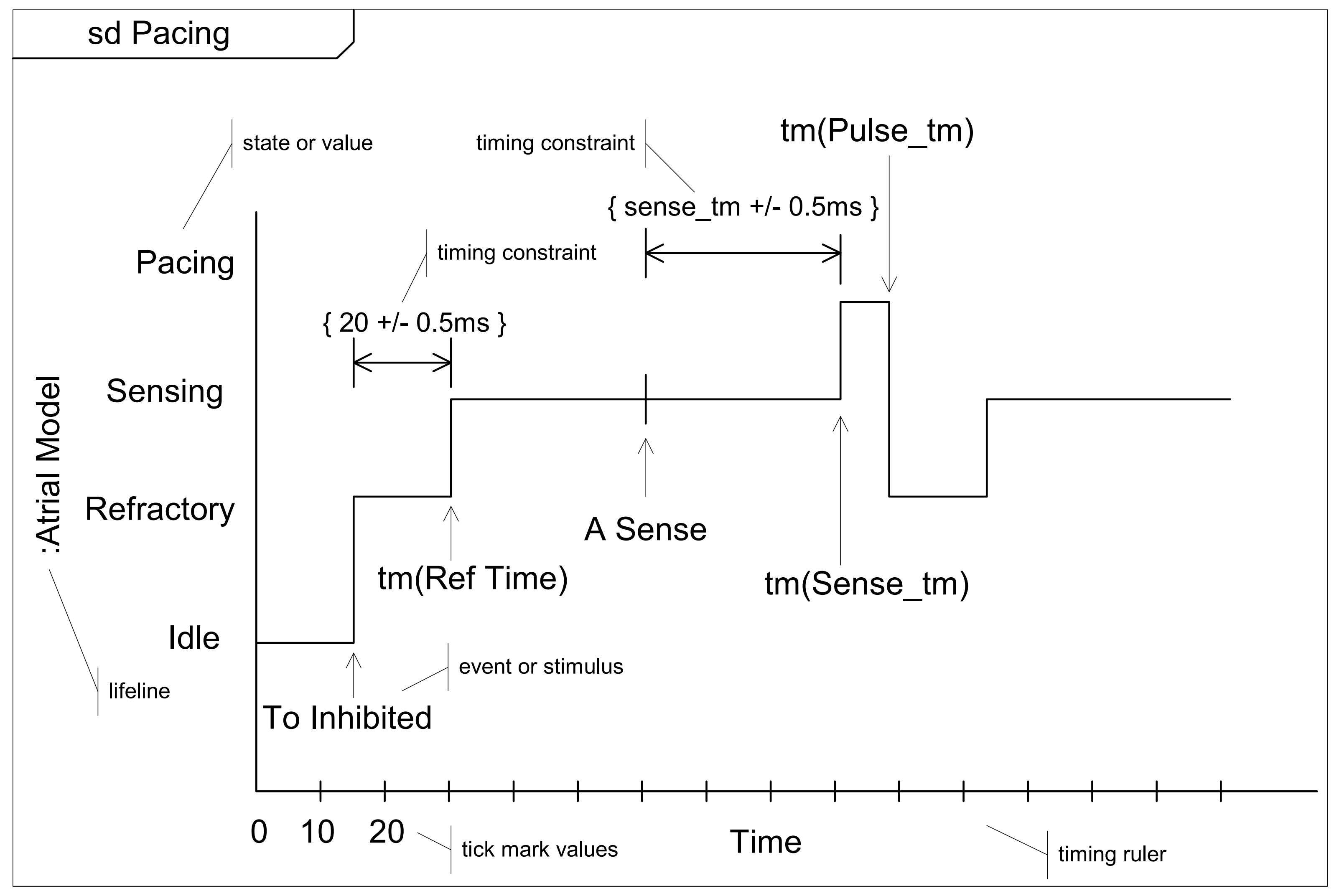
© Telelogic AB

Interaction Operators



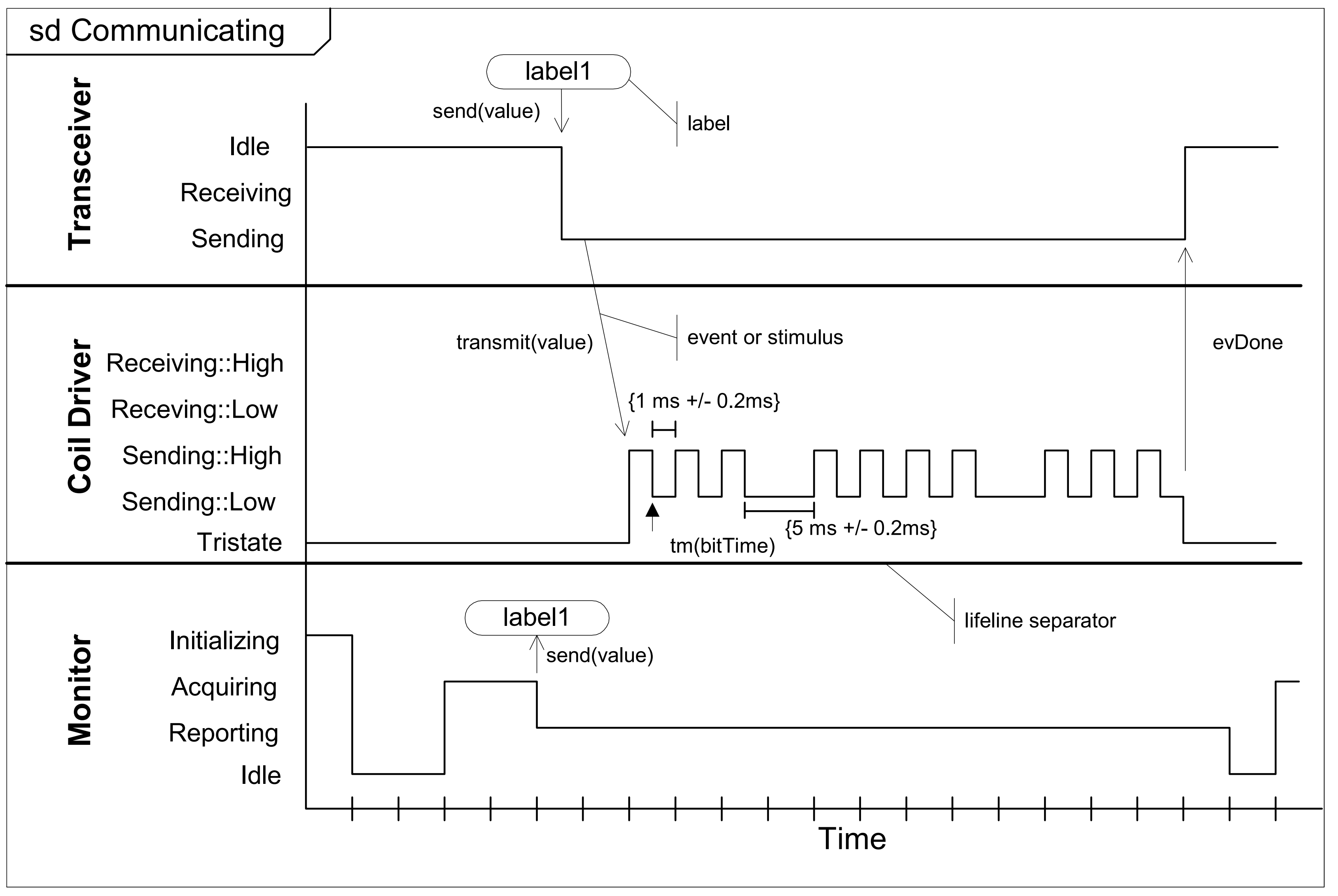
© Telelogic AB Telelogic

UML 2.0 Timing Diagrams



© Telelogic AB

UML 2.0 Timing Diagrams



© Telelogic AB

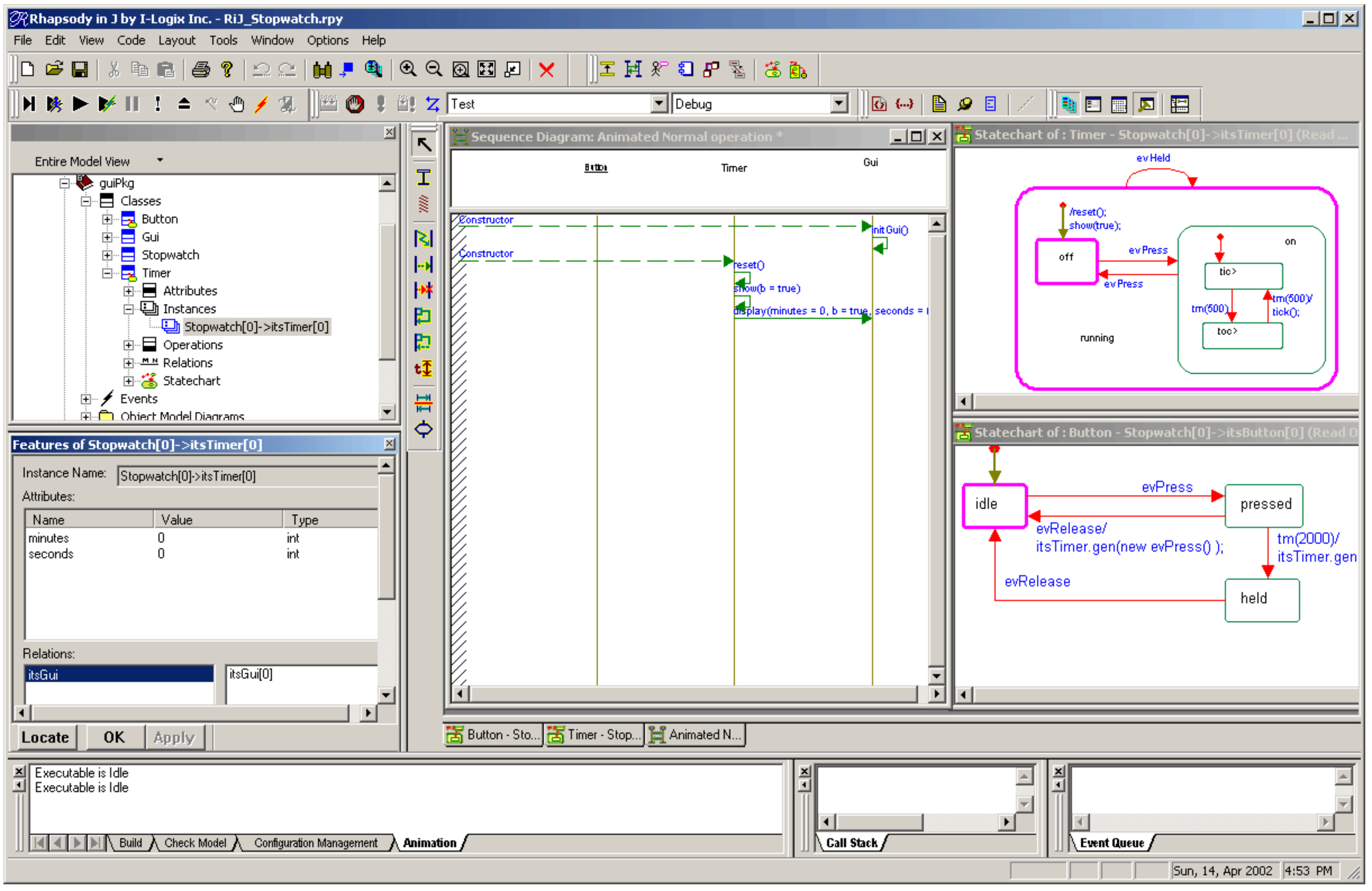
Telelogic

State Behavior

- Useful when object
  - Exhibits discontinuous modes of behaviour
  - Waits for and responds to incoming events
- Sensor example
  - In Start up
    - It must be configured before it can be used
    - It may be turned off
  - When it's ready
    - It rejects requests for data (data not yet available)
    - It can go acquire data upon request
  - Once it has data
    - It may be asked for the sensed value (data has been acquired)
    - It may get data periodically

© Telelogic AB

State Machine Execution



© Telelogic AB

Telelogic

Statecharts

- Created by professor David Harel of I-Logix in late 1980s
- Statecharts were inserted into the UML by Eran Gery and David Harel of I-Logix
- Supports
  - Nested states
  - Actions
  - Guards
  - History
- Advanced features
  - And-states
  - Broadcast transitions
  - Orthogonal regions

© Telelogic AB

Statecharts

- What's a state?

A **state** is a *distinguishable, disjoint, orthogonal condition of existence of an object that persists for a significant period of time*

- What's a transition?

A **transition** is a *response to an event of interest moving the object from a state to a state.*

© Telelogic AB

Statecharts

- What's an action?

An **action** is a run-to-completion behavior. The object will not accept or process any new events until the actions associated with the current event are complete.

- What's the order of action execution?

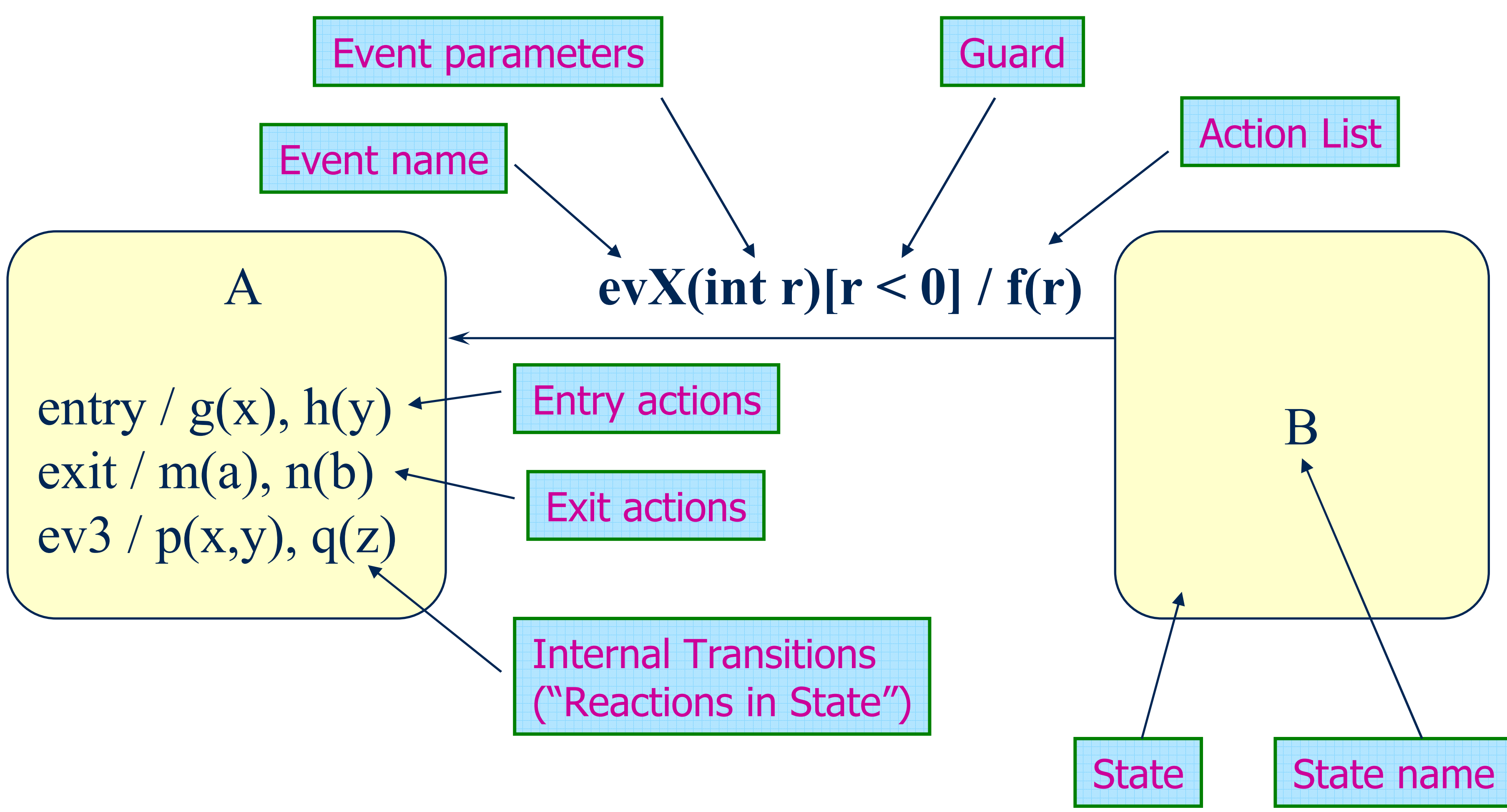
(1) exit actions of current state
(2) transition actions
(3) entry actions of next state

 © Telelogic AB Telelogic

UML State Models

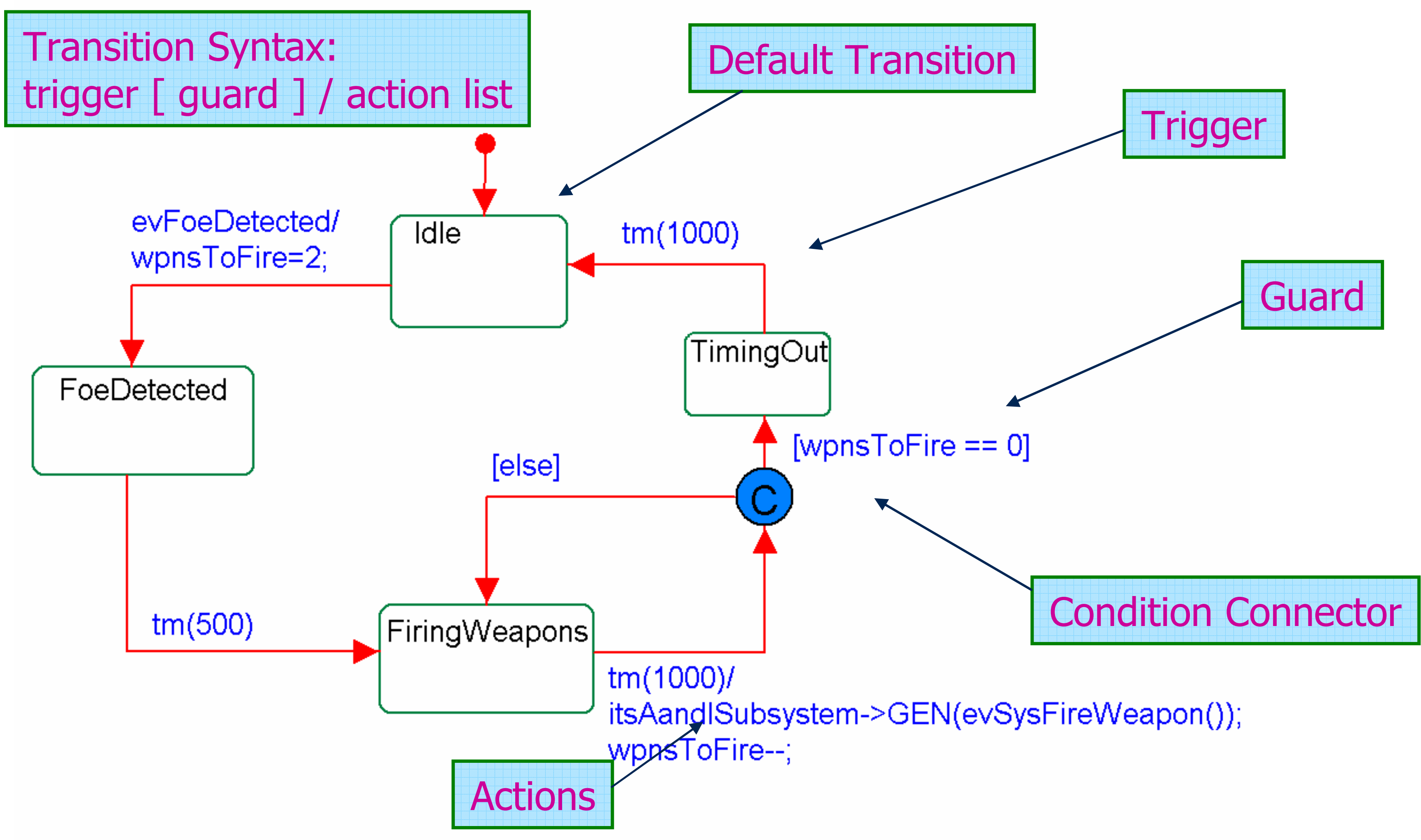
- Specifies the behaviour for reactive classes
- Not all classes have state behaviour
- Notation and semantics are Harel Statecharts
  - Nested States
  - Actions on
    - Transitions
    - State Entry
    - State Exit

Basic Statechart Syntax



© Telelogic AB

Basic Syntax Example



© Telelogic AB Telelogic

Types of Events

- UML defines 4 kinds of events
  - Signal Event
    - Asynchronous signal received e.g. evStart
  - Call Event
    - operation call received e.g. opCall(a,b,c)
  - Change Event
    - change in value occurred
  - Time Event
    - Absolute time arrived
    - Relative time elapsed e.g. tm(PulseWidthTime)

© Telelogic AB

Handling Transitions

- If an object is in a state S that responds to a named event evX, then it will act upon that event
- It will transition to the specified state, if the event triggers a named transition and the guard (if any) on that transition evaluates to TRUE. It will execute any actions associated with that transition
- Handle the event without changing state if the event triggers a named reaction and execute all the list of actions associated with that reaction

© Telelogic AB Telelogic

Transitions: Guards

- A guard is some condition that must be met for the transition to be taken
- Guards are evaluated *prior* to the execution of any action.
- Guards can be:
  - Variable range specification ex: [cost<50]
  - Concurrent state machine is in some state [IS_IN(fault)]
  - Some other constraint (*preconditional invariant*) must be met

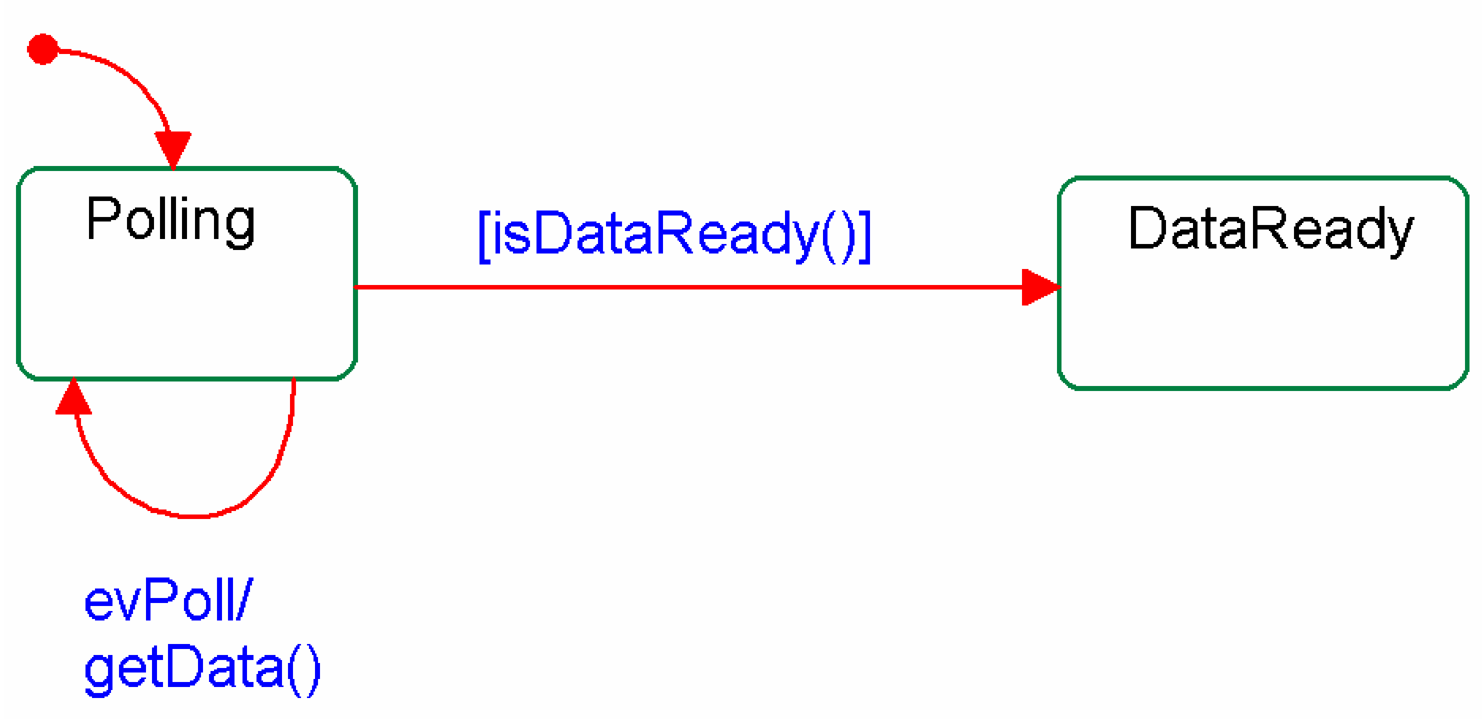
© Telelogic AB

Actions

- Actions are run to completion
  - Normally actions take a short period of time to perform
  - They may be interrupted by another thread execution, but that object will complete its action list before doing anything else
- Actions are implemented via
  - An object's operations
  - Externally available functions
- They may occur when
  - A transition is taken
  - A *reaction* occurs
  - A state is entered
  - A state is exited

© Telelogic AB

Null-triggered Transitions



Null-triggered transitions trigger immediately upon completion of any actions-on-entry. If the guard '*isDataReady()* evaluates to FALSE, then the ONLY WAY it will ever be evaluated again is if event *evPoll* occurs, retriggering the null-triggered transition.

© Telelogic AB

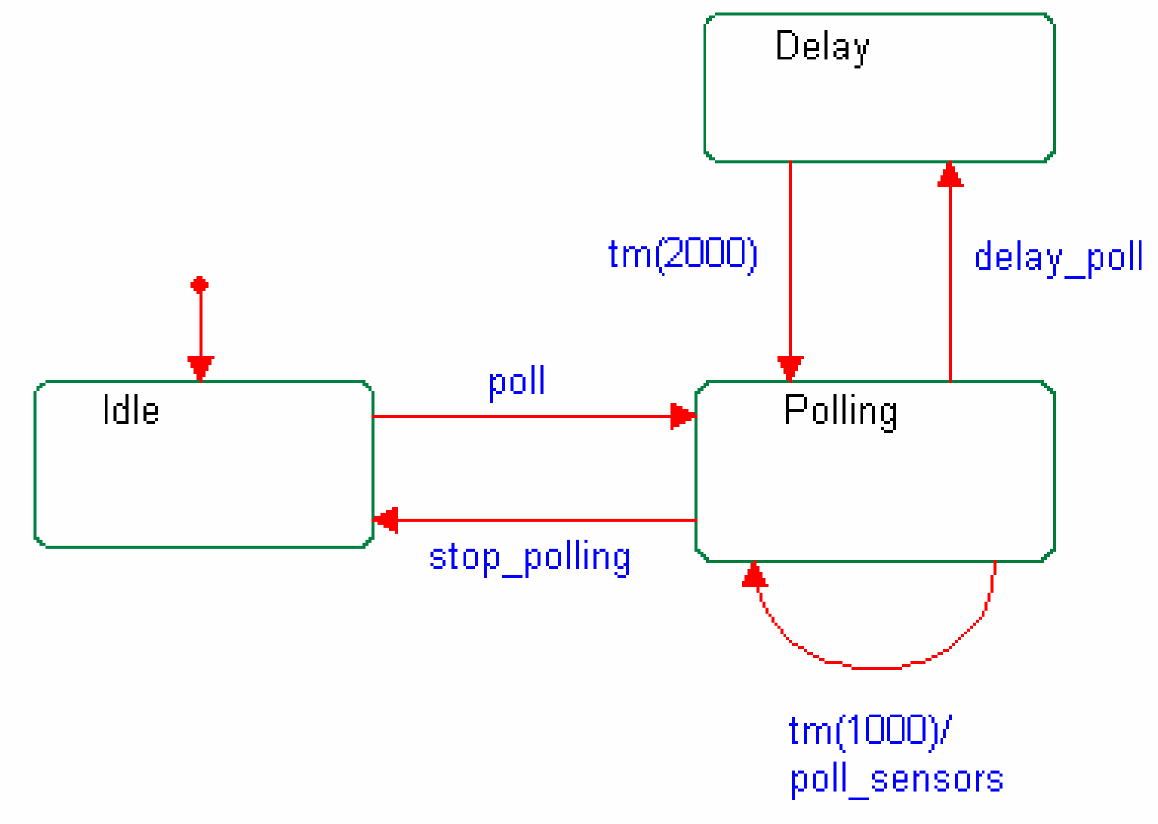
Timeouts

- When an object enters a state, any Timeout from that state are started
- When a Timeout Expires, the State machine receives the expiration as an event
- When an object leaves a state, any timeout that was started on entry to that state are cancelled
- Only one timeout can be used per state, nested states can be used if several timeouts are needed

© Telelogic AB

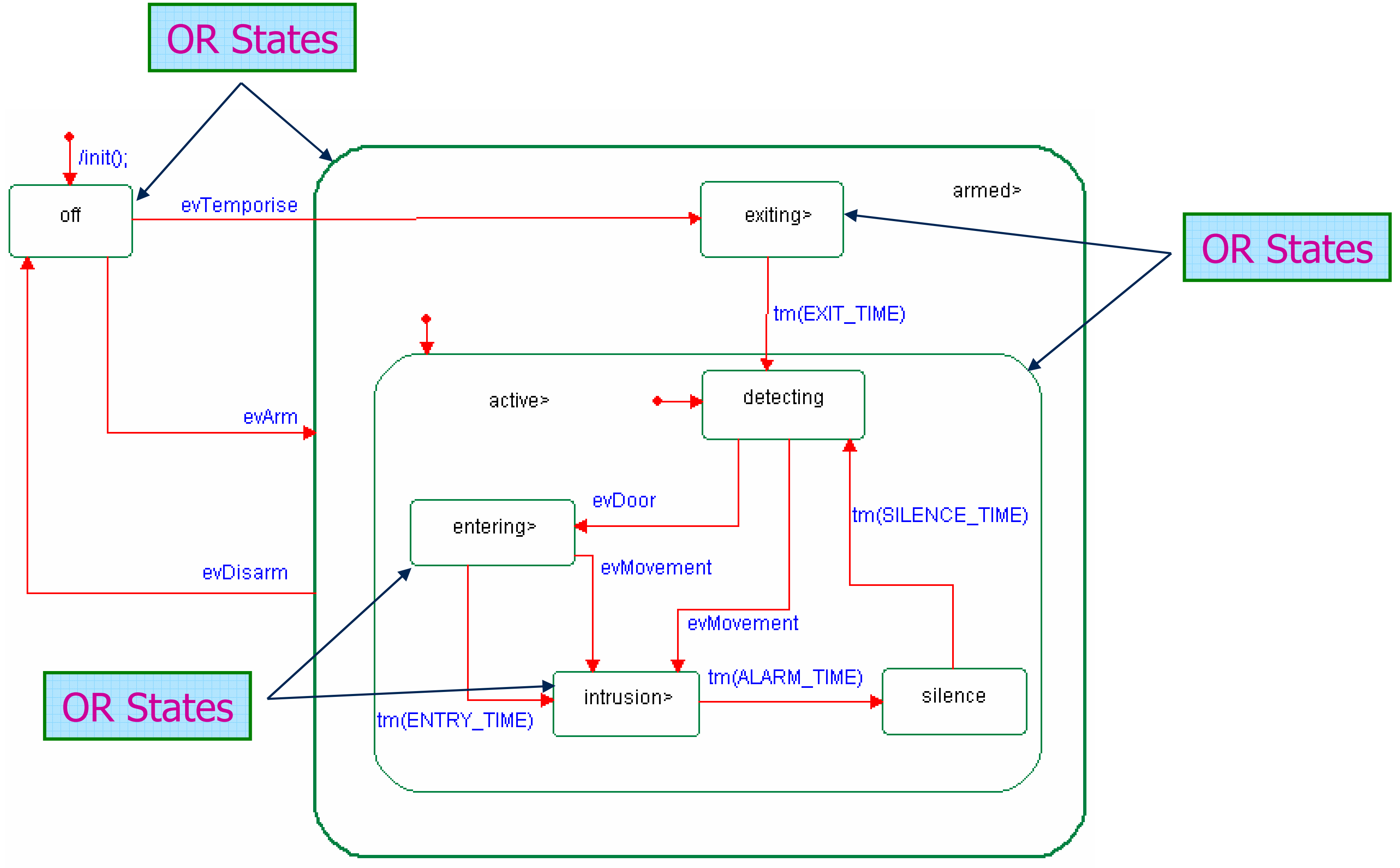
Timeouts

- Timeout Example:



© Telelogic AB

Statechart Syntax – OR States



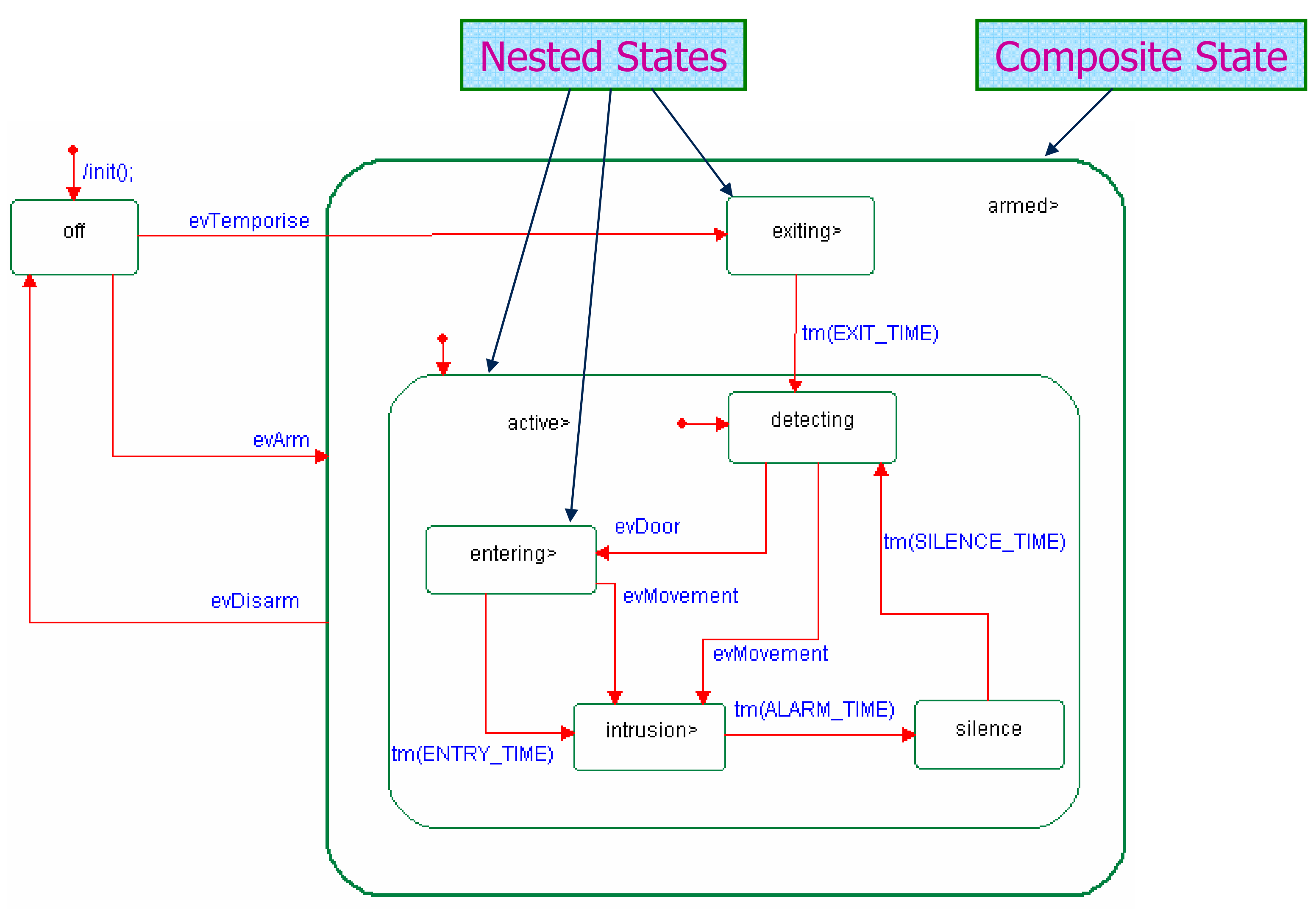
© Telelogic AB
Telelogic

OR-States

- An object must always be in *exactly one* OR-state at a given level of abstraction.
  - The object must be in either *off* or *armed* – it cannot be in more than one or none
  - If the current state is *armed*, then the object must ALSO be in either *exiting* or *active*
  - Note, if IS_IN(detecting) returns TRUE, then IS_IN(active) returns TRUE and IS_IN(exiting) returns FALSE

© Telelogic AB

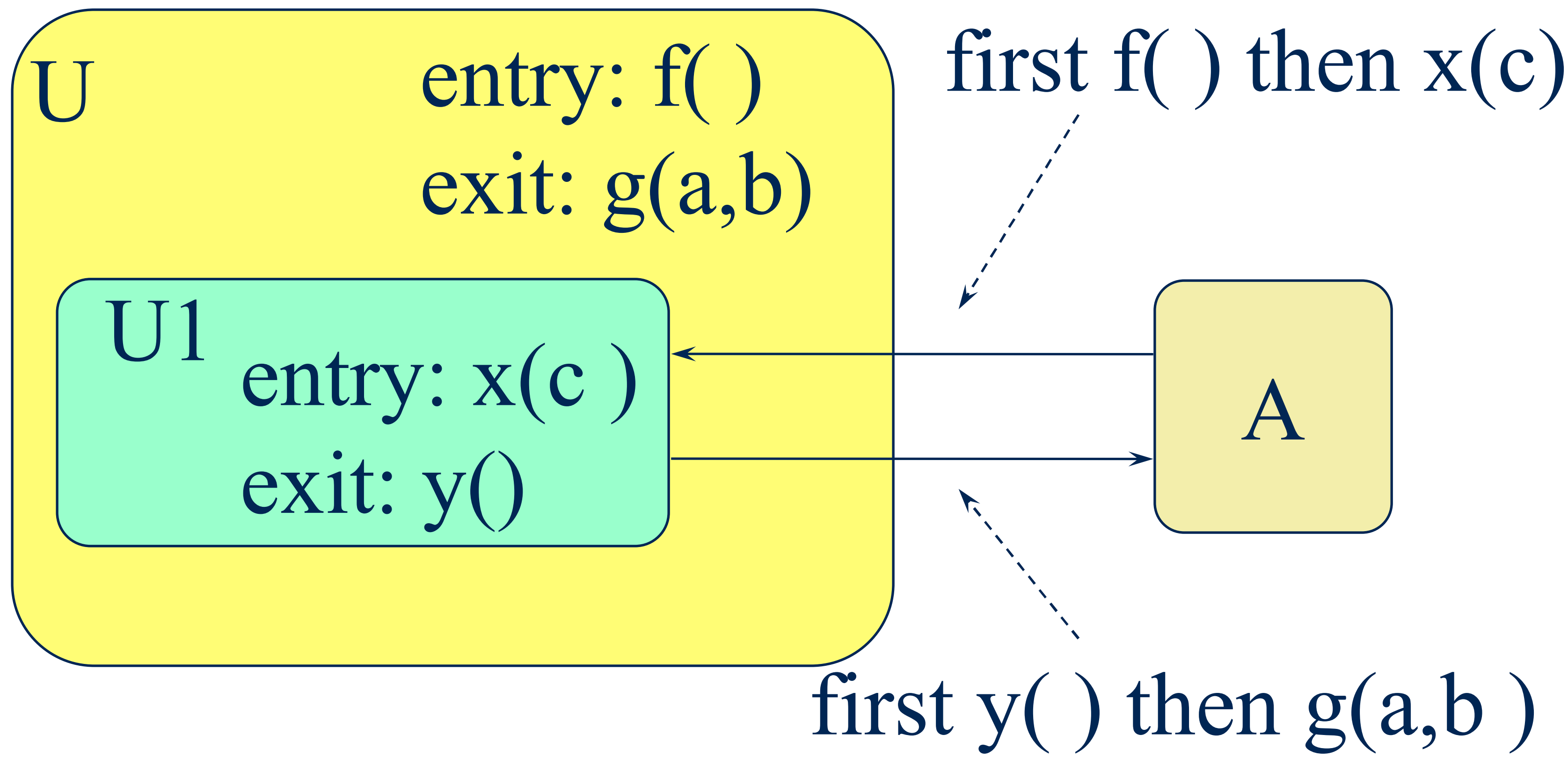
Statechart Syntax – Nested States



© Telelogic AB

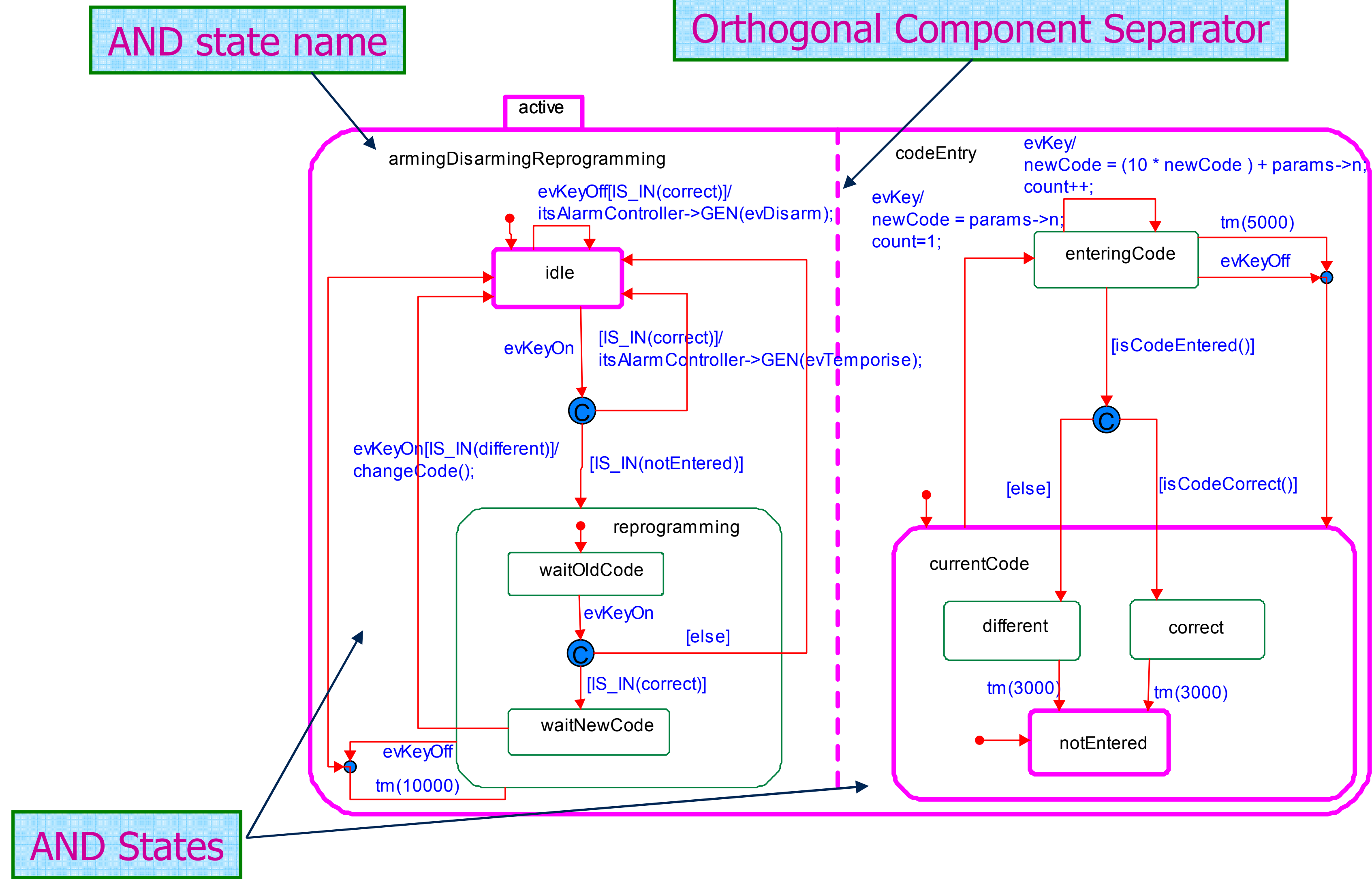
Order of Nested Actions

- Execute from outermost - in on entry
- Execute from innermost - out on exit



© Telelogic AB

Statechart Syntax – AND States



© Telelogic AB

Telelogic

AND-States

- When a state has multiple AND-states, the object must be in exactly one substate of each active AND-State at the same time
- AND-states are logically concurrent
  - All active AND-states receive their own copy of any event the object receives and independently acts on it or discards it
  - Cannot tell *in principle* which and-state will execute the same event first
  - Not necessarily concurrent in the thread or task sense
  - NOTE: UML uses active objects as the primary means of modeling concurrency
  - AND-states may be implemented as concurrent threads, but that is not the only correct implementation strategy

© Telelogic AB

AND-State Communication

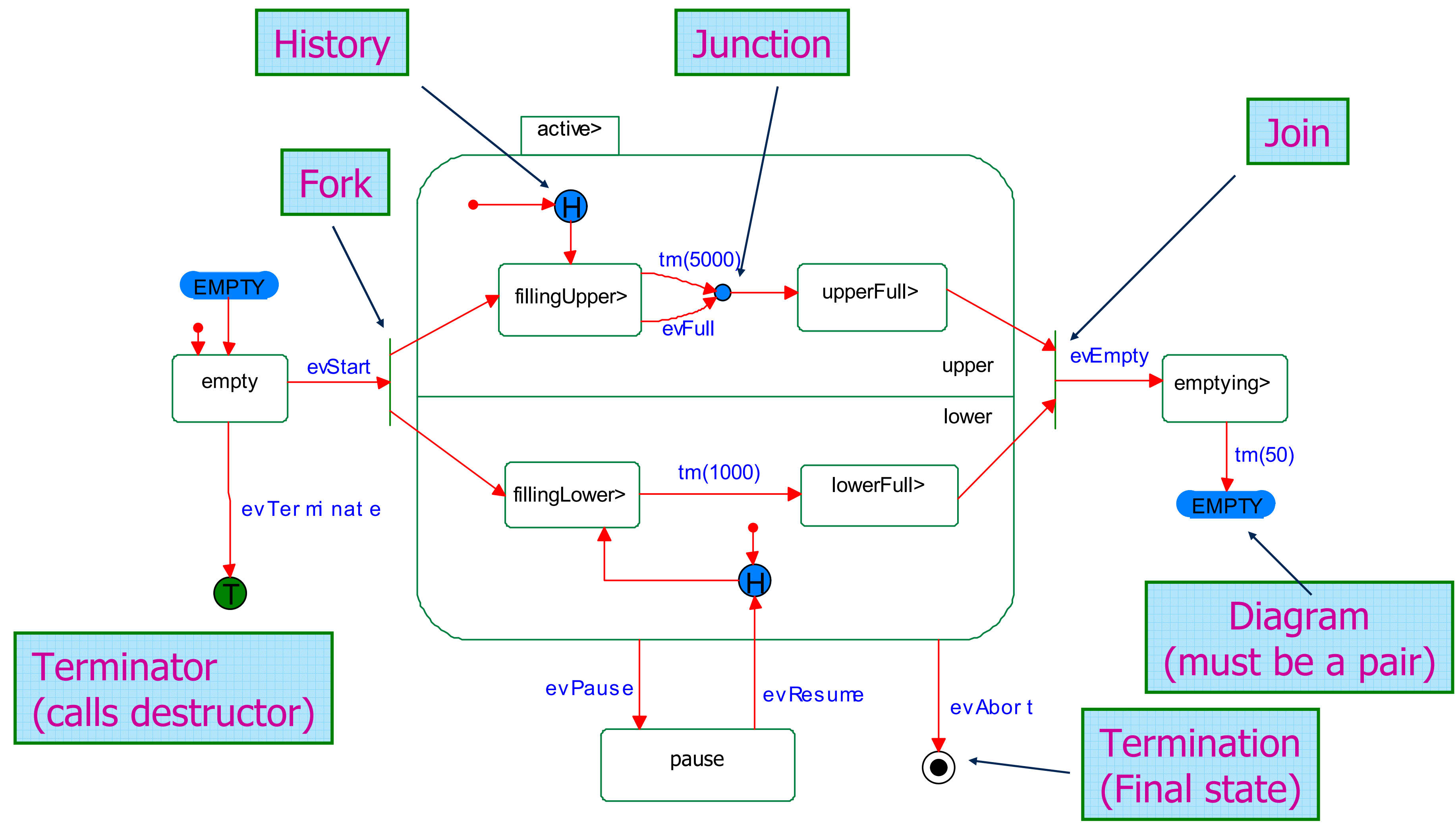
- AND-states may communicate via
  - Broadcast events
    - All active AND-states receive their own copy of each received event and are free to act on it or discard it
  - Propagated events
    - A transition in one AND-state can send an event that affects another
  - Guards
    - [IS_IN(state)] uses the substate of an AND-state in a guard
  - Attributes
    - Since the AND-states are of the same object, they “see” all the attributes of the object

© Telelogic AB

UML Pseudostates

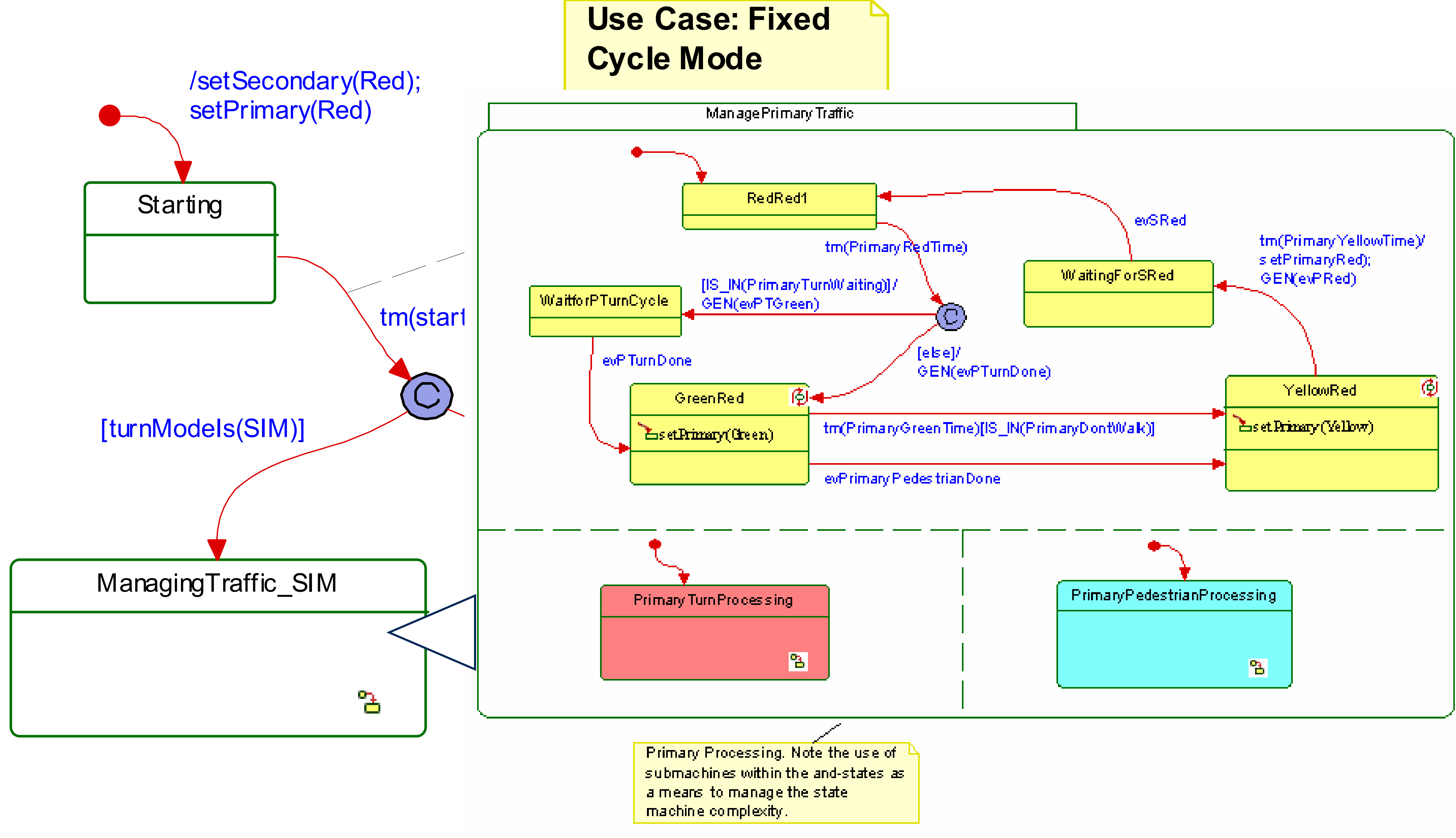
Symbol	Symbol Name	Symbol	Symbol Name
C or ◇	Branch Pseudostate (*type of junction pseudostate*)	H	(Shallow) History Pseudostate
T or ●	Terminal or Final Pseudostate	H*	(Deep) History Pseudostate
			Initial or Default Pseudostate
	Fork Pseudostate		Junction Pseudostate
	Join Pseudostate		Merge Junction Pseudostate (*type of junction pseudostate*)
[g] [g]	Choice Point Pseudostate	label	Entry Point Pseudostate
		label	Exit Point Pseudostate

Statechart Syntax – Pseudostates



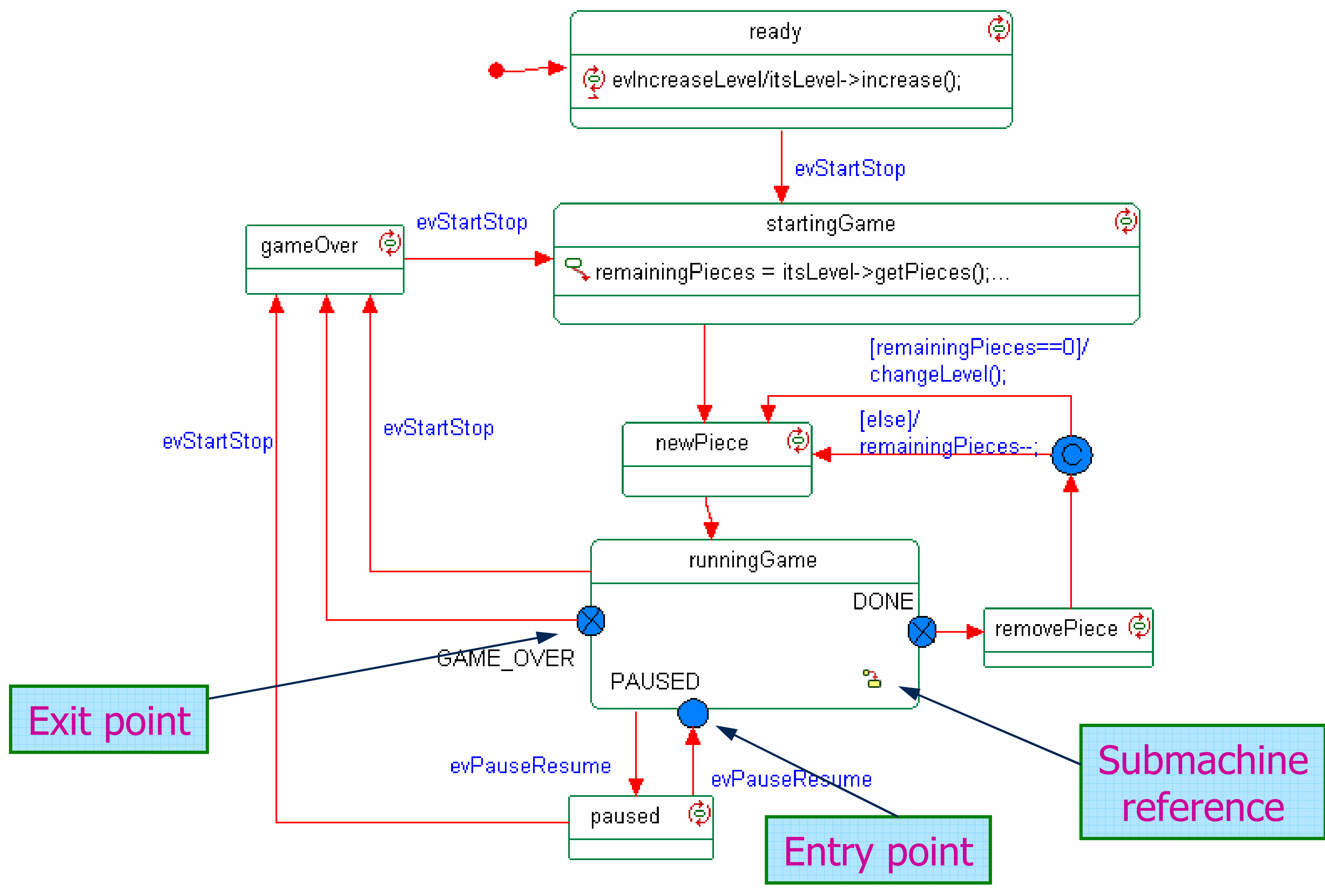
© Telelogic AB
Telelogic

Statecharts



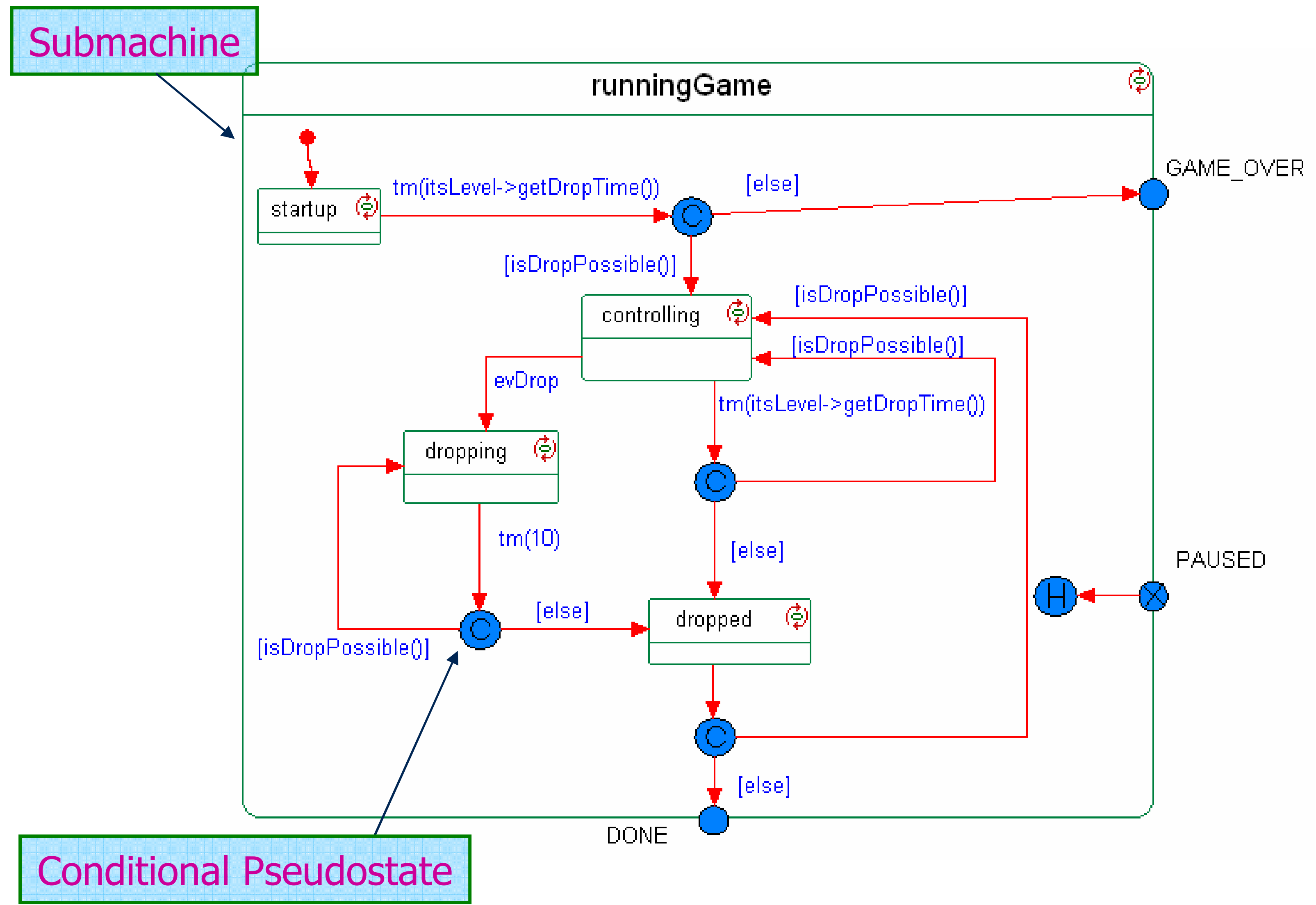
© Telelogic AB
Telelogic

Submachines: Parent



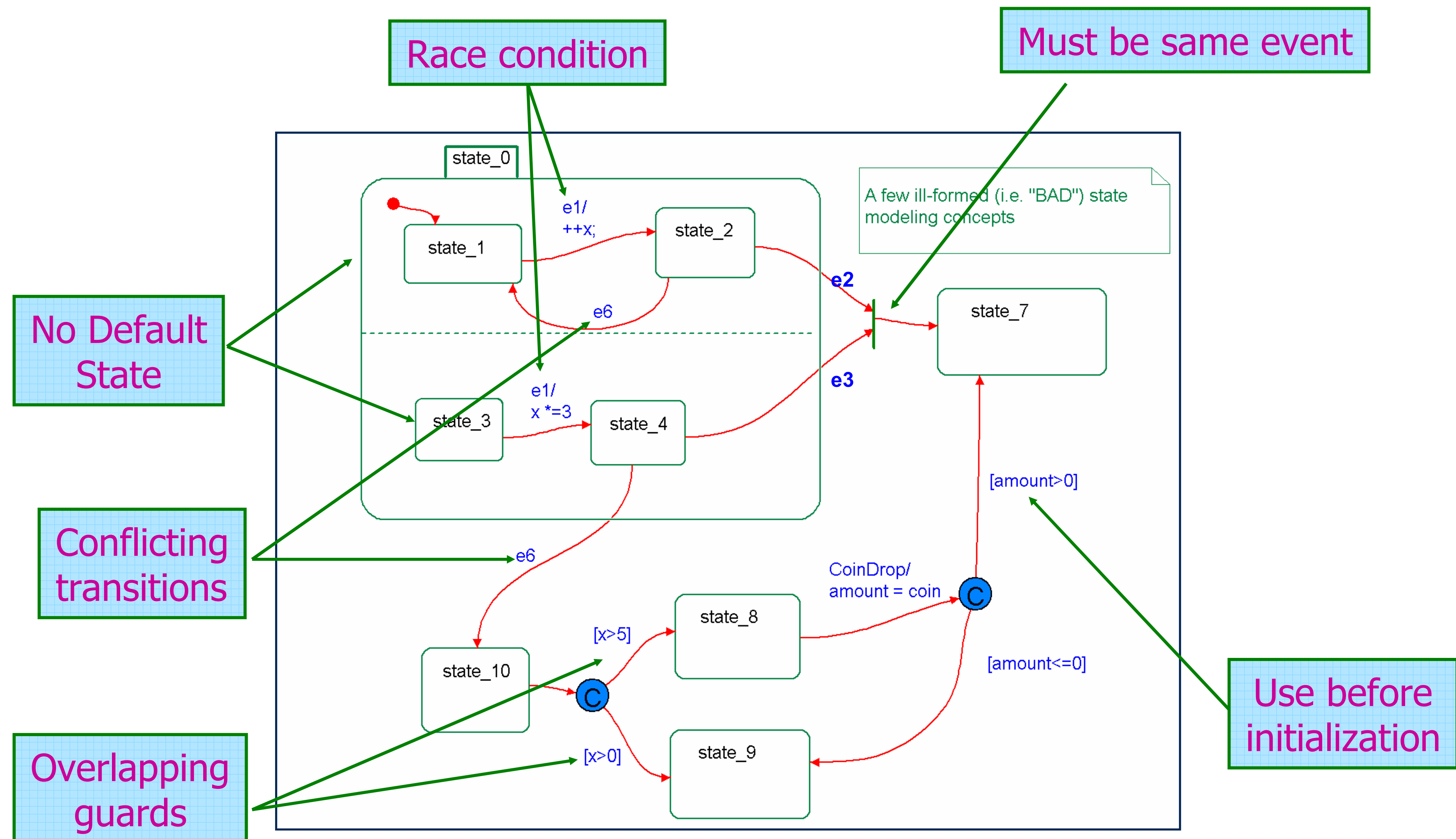
© Telelogic AB
Telelogic

Submachines: Child



© Telelogic AB

Poorly Formed Statecharts



© Telelogic AB Telelogic

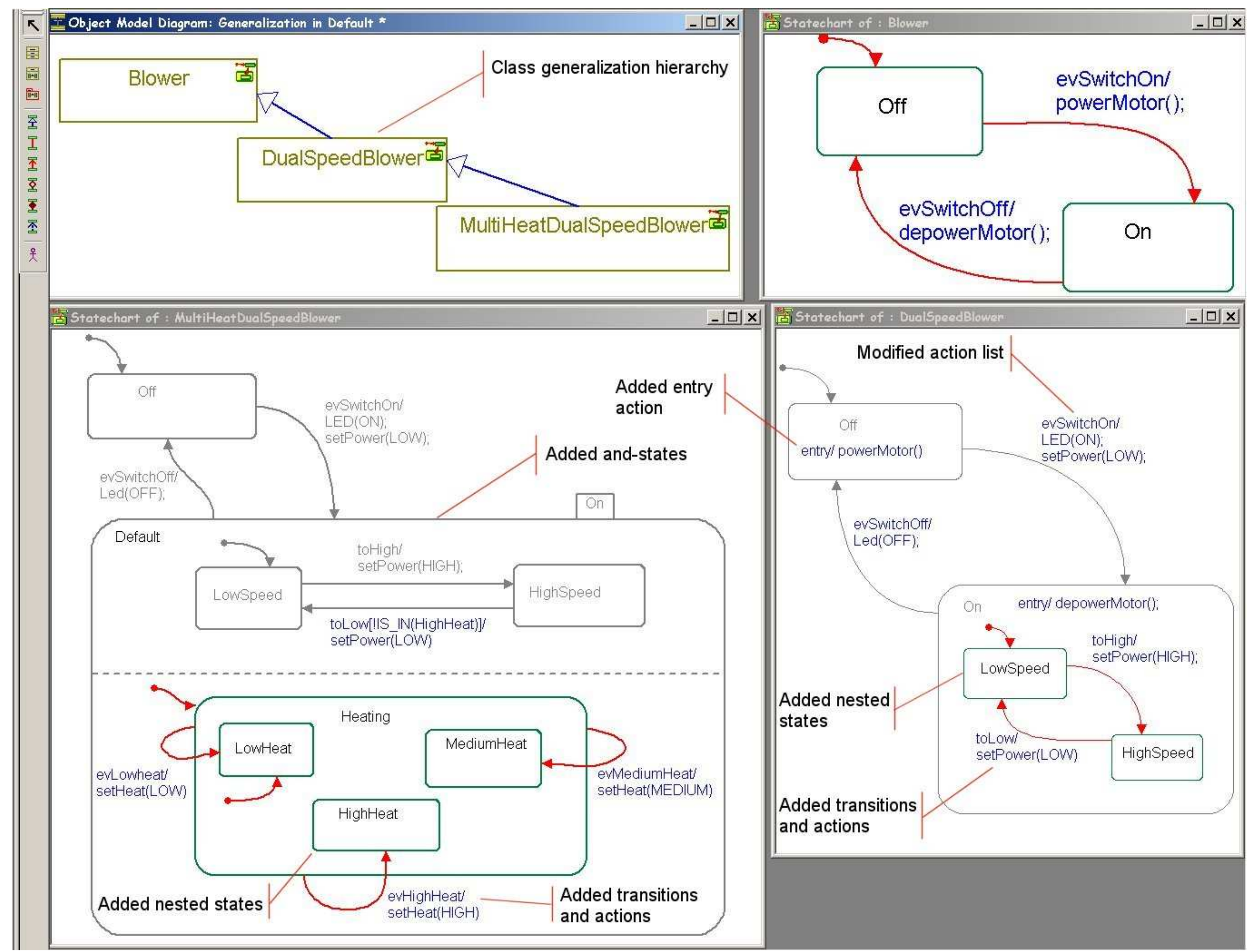
Inherited State Behaviour

- Two approaches to inheritance for generalization of reactive classes
  - Reuse (i.e. inherit) statecharts of parent
  - Use custom statecharts for each subclass

- Reuse of statecharts allows
  - Specialization of existing behaviours
  - Addition of new states and transitions
  - Makes automatic code generation of reactive classes efficient in the presence of class generalization

© Telelogic AB

Example: Generalization



© Telelogic AB

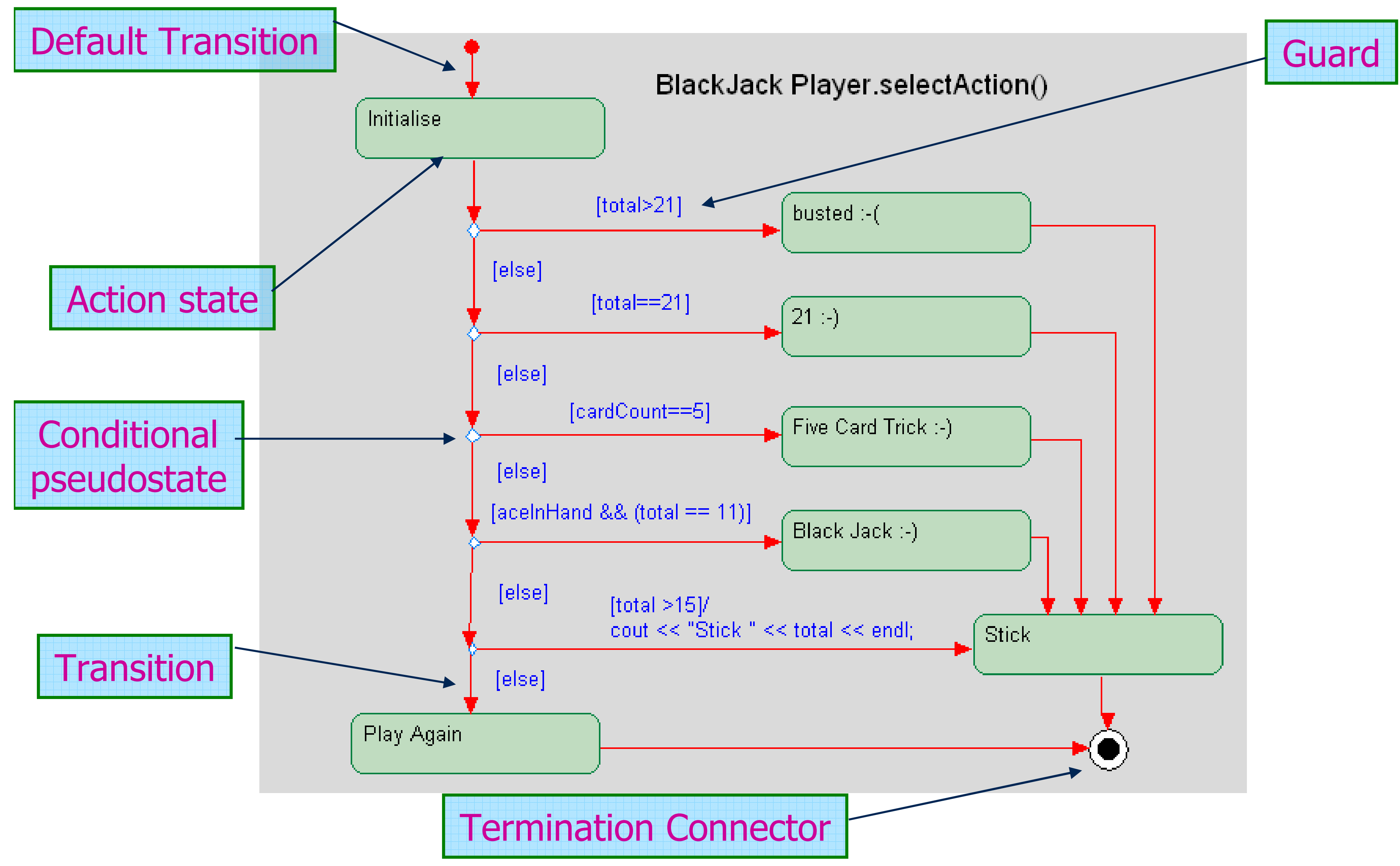
Telelogic

Activity Diagrams

- Change in 2.0 to be based on *token flow semantics*
- Used when the primary means of transitioning from one state to another is upon *completion* of the previous activity not reception of an event
  - An activity diagram can be assigned to either a class, use case or an operation.
  - Useful for describing algorithms
- Each activity has a set of *pins*
  - Input pins bind input parameters to “local variables”
  - Output pins bind output parameters to “output variables”
  - An activity begins when input data appears on all input pins
  - When an activity completes, there is data on all the output pins
  - Activities are no longer triggered by events

© Telelogic AB

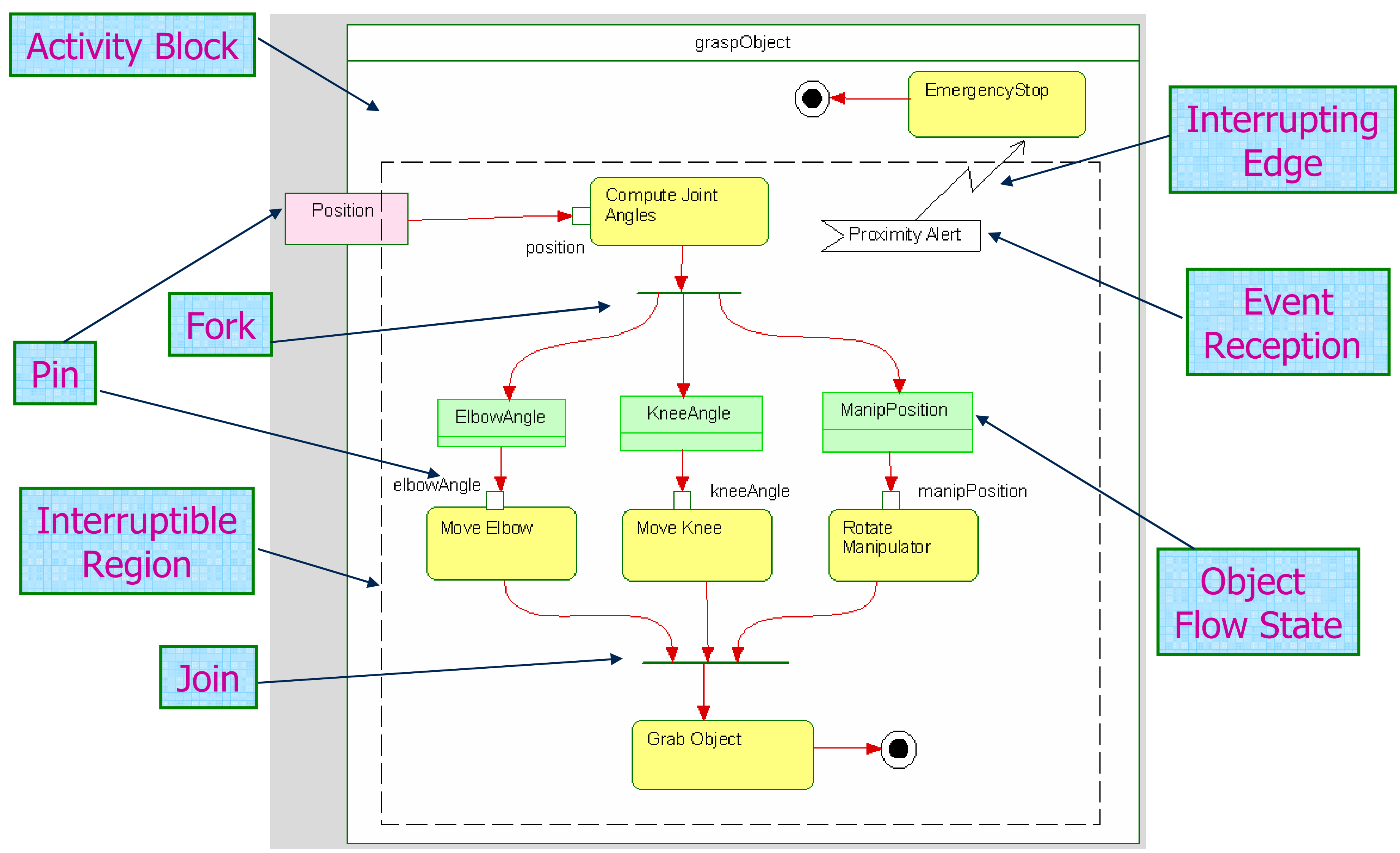
Activity Diagram : Basic Syntax



© Telelogic AB

Telelogic

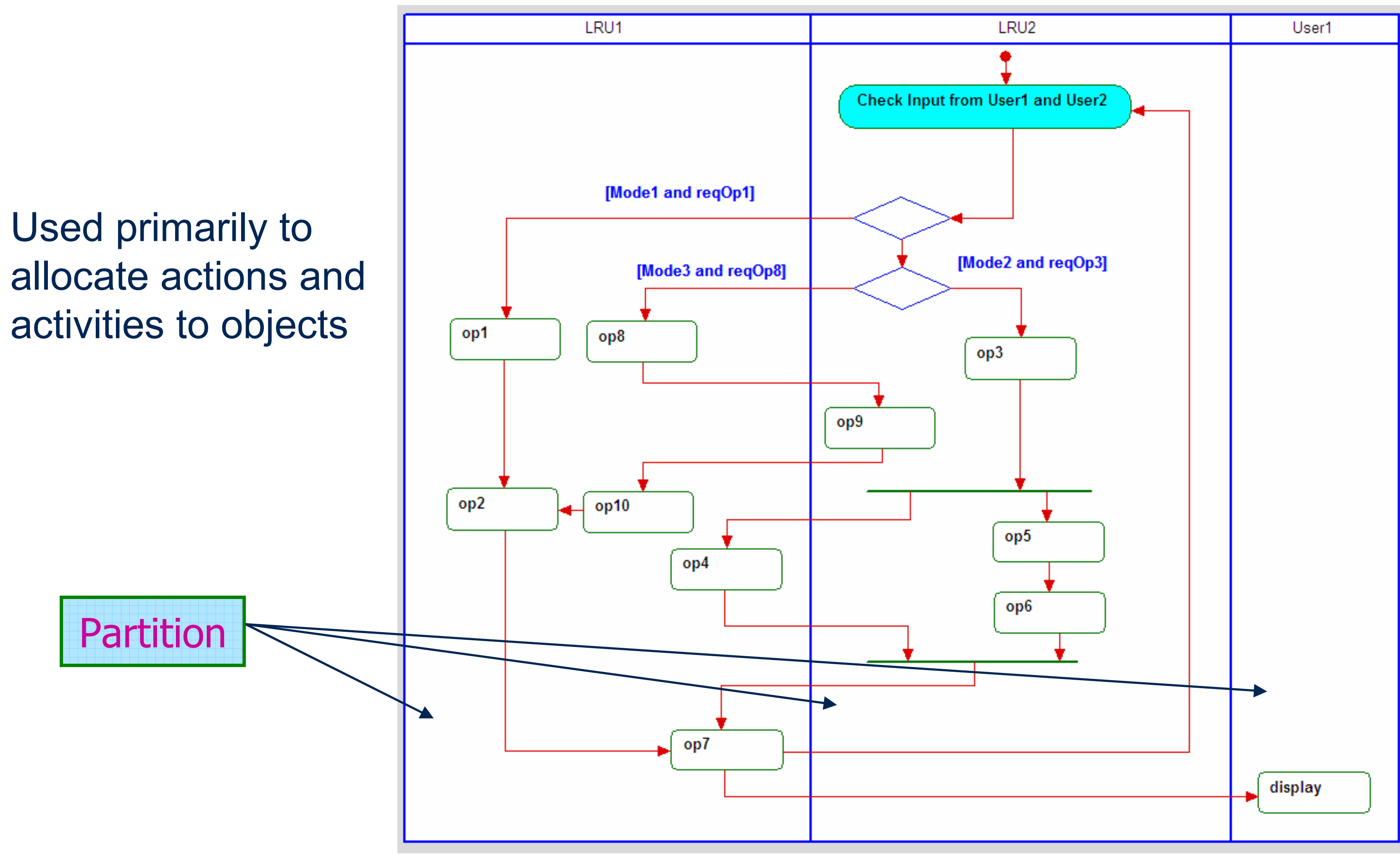
Activity Diagram : Advanced Syntax



© Telelogic AB Telelogic

Activity Diagram: Partitions (Swim Lanes)

Used primarily to allocate actions and activities to objects

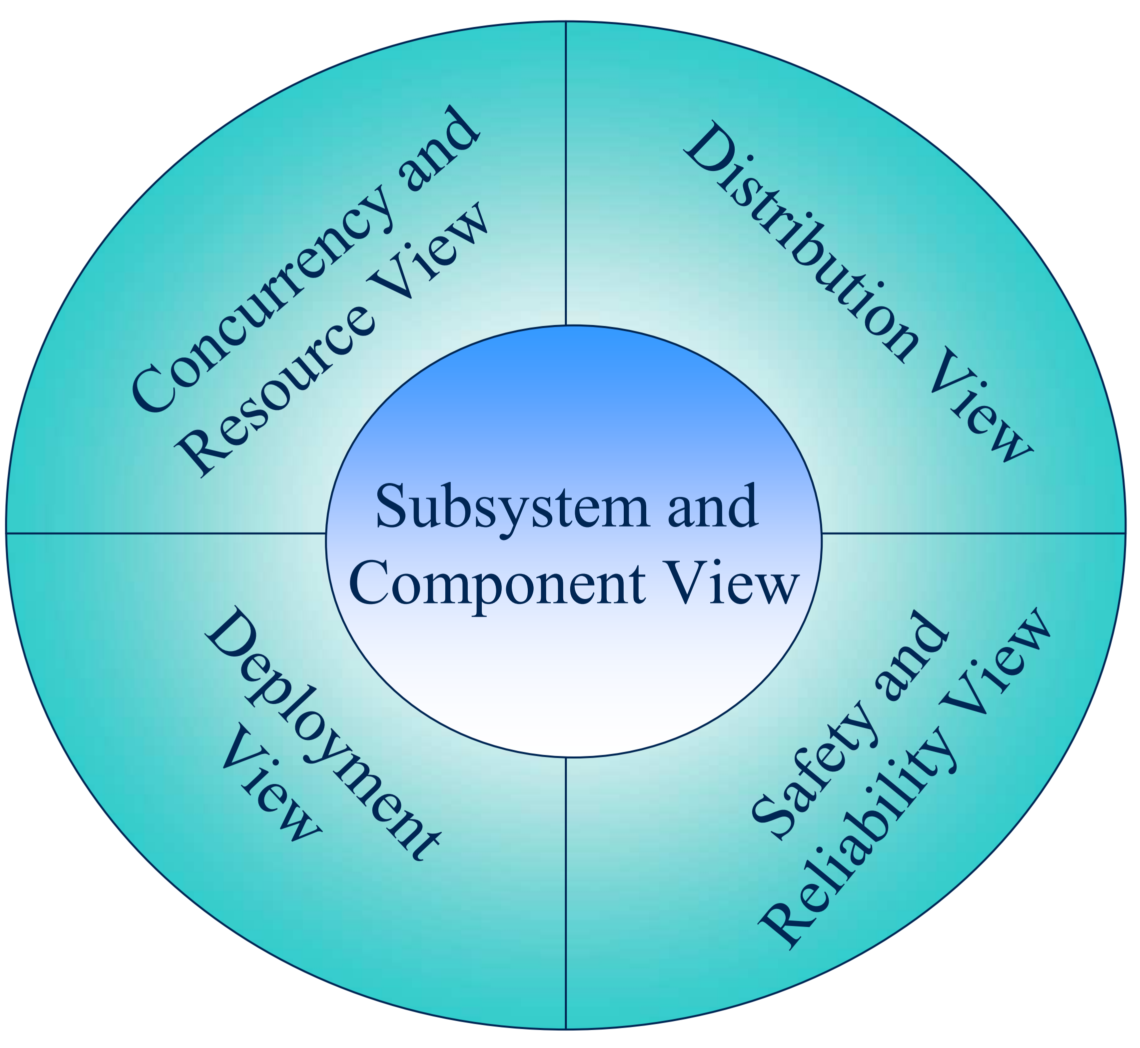


© Telelogic AB

Architecture

© Telelogic AB

Physical Architectural Views



© Telelogic AB

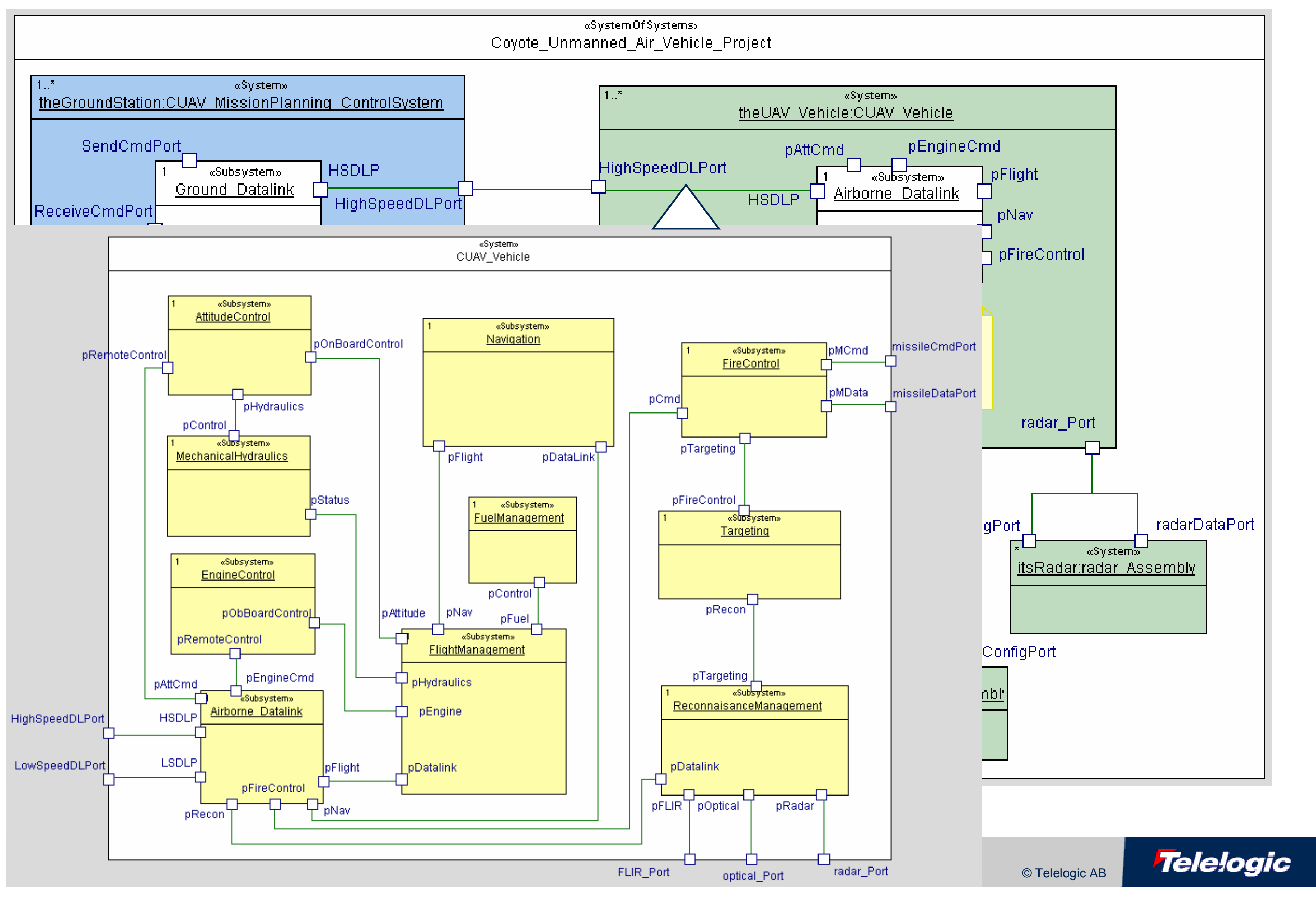
Telelogic

Physical Architectural Views

- Construct architectural design models
  - Subsystem Model
  - Concurrency Model
  - Distribution Model
  - Safety and Reliability Model
  - Deployment Model
- Capture with
  - Class Diagrams
  - Package Diagrams
  - Subsystem Diagrams
  - Task Diagrams
  - Deployment Diagrams

© Telelogic AB

Subsystem Architecture



© Telelogic AB
Telelogic

Subsystem and Component View

- A component
  - is the basic reusable element of software
  - organizes objects together into cohesive run-time units that are replaced together.
  - provides language-independent opaque interfaces
  - a metasubtype of (structured) Class
- A subsystem
  - is a large object that provides opaque interfaces to its clients and achieves its functionality through delegation to objects that it owns internally
  - contains components and objects on the basis of common run-time functional purpose
  - a metasubtype of Component

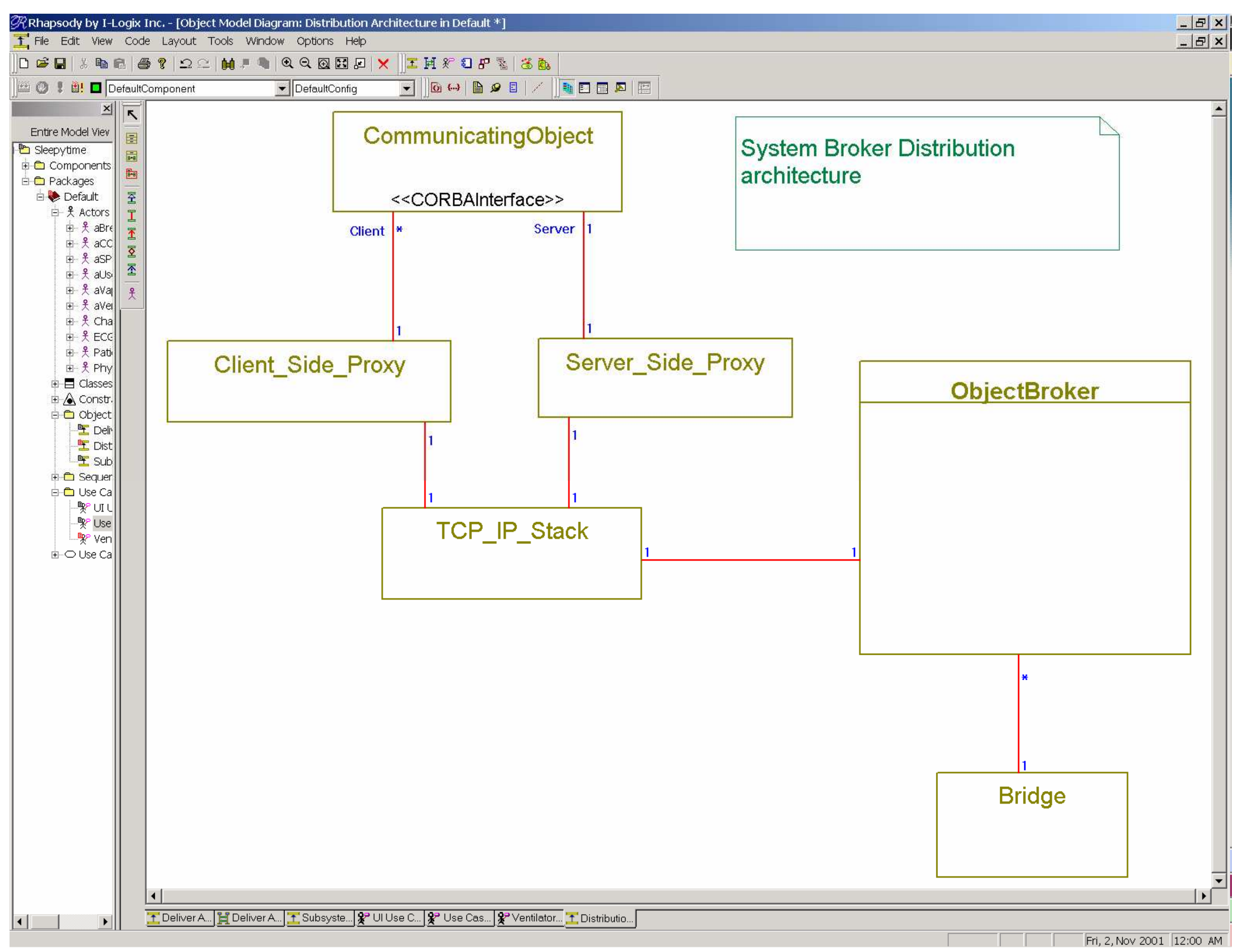
© Telelogic AB

Distribution Architecture

- Distribution model refers to
  - Policies for distribution objects among multiple processors and communication links, e.g.
    - Asymmetric distribution (dedicated links to objects with a priori known location)
    - Publish-Subscribe
    - CORBA and Broker symmetric distribution
  - Policies for managing communication links
    - Communication protocols
    - Communication quality of service management

© Telelogic AB

Distribution Architecture



© Telelogic AB

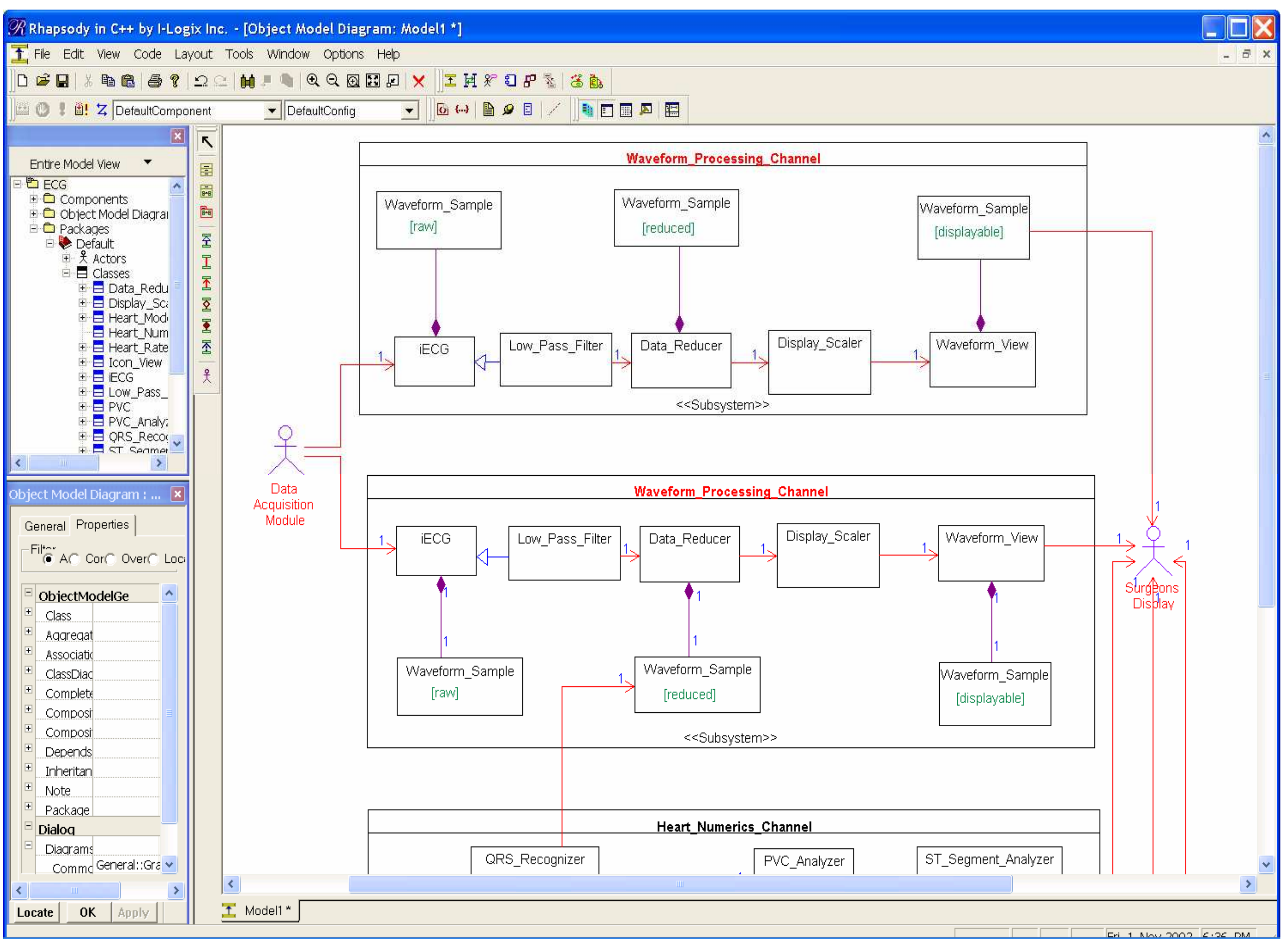
Safety and Reliability Model

- Safety and reliability model refers to the structures and policies in place to ensure
  - Safety
    - Freedom from accidents or losses
  - Reliability
    - High MTBF
    - Fault tolerance
- Safety and fault tolerance always require some level of redundancy

Safety and reliability of object models is described more completely in *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*

 © Telelogic AB

Safety and Reliability Architecture



© Telelogic AB
Telelogic

Concurrency Architecture

- Refers to
  - Identification of task threads and their properties
  - Mapping of passive classes to task threads
  - Identification of synchronization policies
  - Task scheduling policies
- Unit of concurrency in UML is the «active» object
  - «active» objects are added in architectural design to organize passive objects into threads
  - «active» objects contain passive semantic objects via composition and delegate asynchronous messages to them

© Telelogic AB

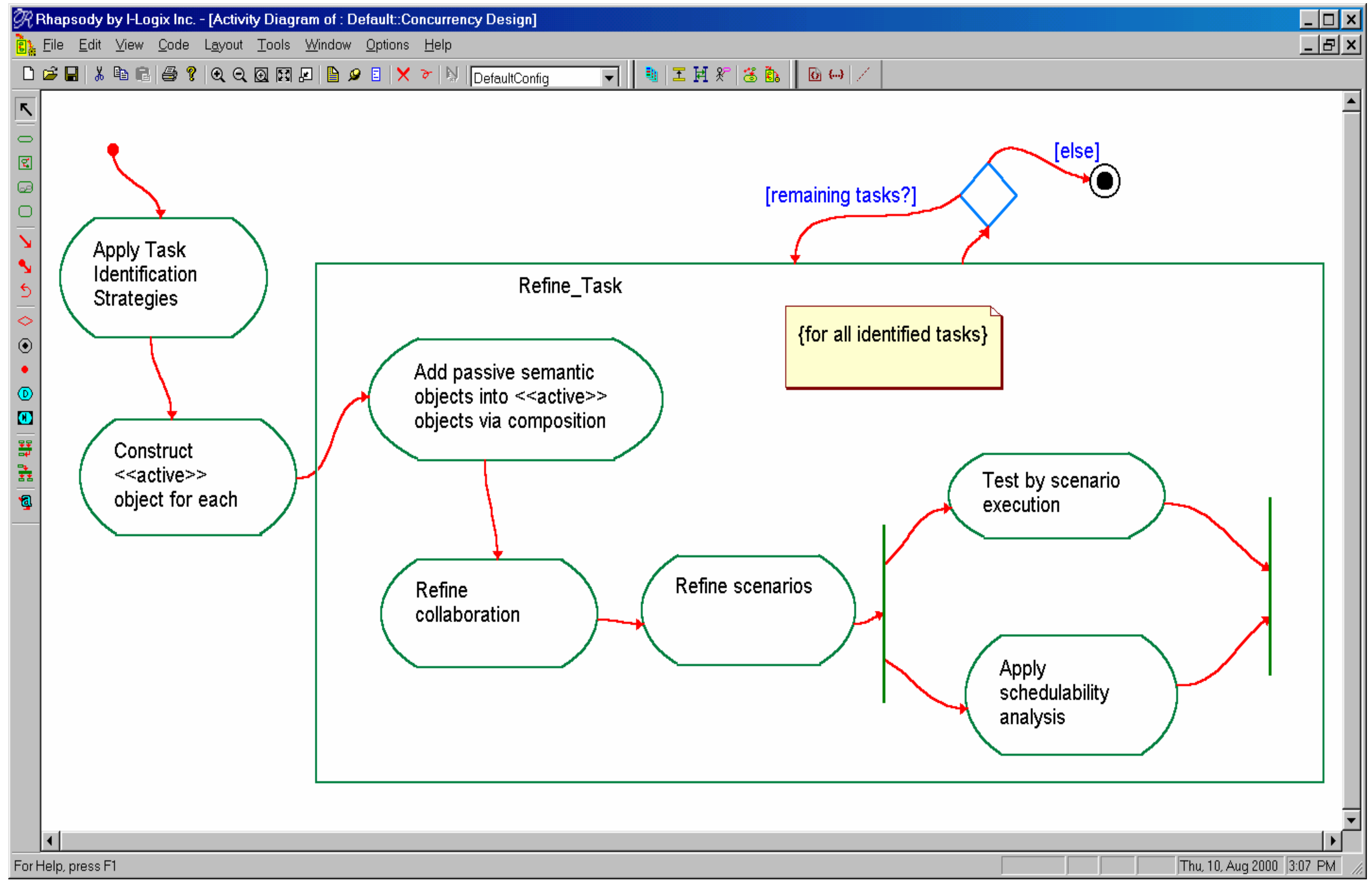
Task Identification

Task Identification Strategy	Description
Single event groups	for simple systems, you may define a thread for each event type
Event source	group all events from a single source together for a thread
Related information	For example, all numeric heart data
Interface device	For example, a bus interface
Event properties	**Events with the same period, or aperiodic events**
Target object	For example, waveform queue or trend database
Safety Level	For example, BIT, redundant thread processing, watchdog tasks

Best for schedulabilty

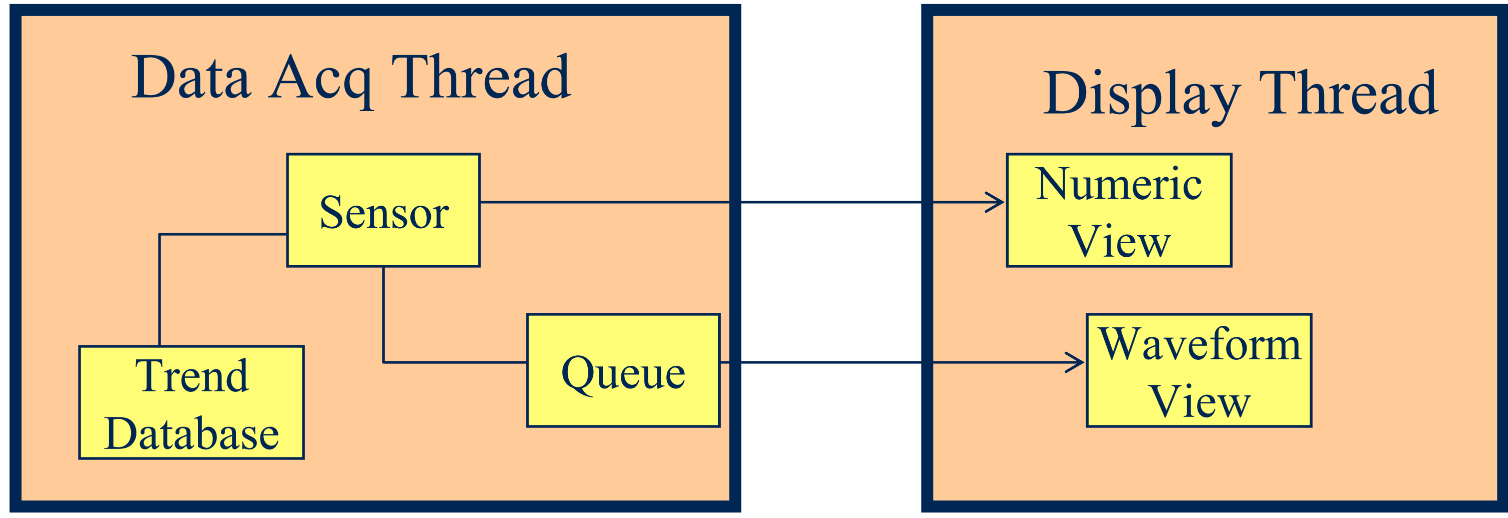
© Telelogic AB

Defining the Concurrency Model



Concurrency Model

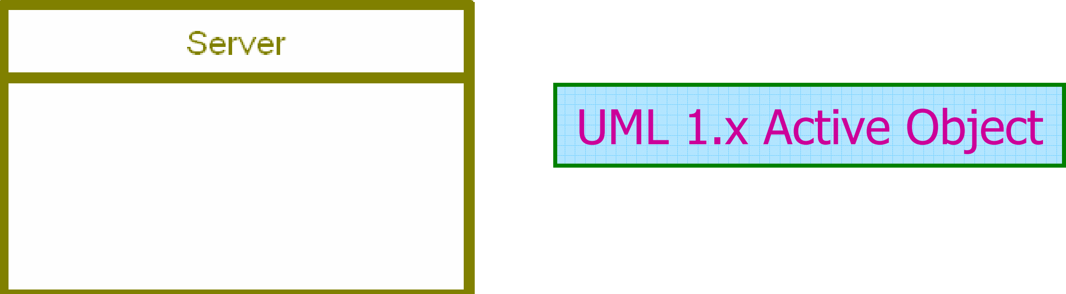
- Active object is a stereotype of an object which owns the root of a thread
- Active objects normally aggregate passive objects via composition relations
- Standard icon is a class box with heavy line



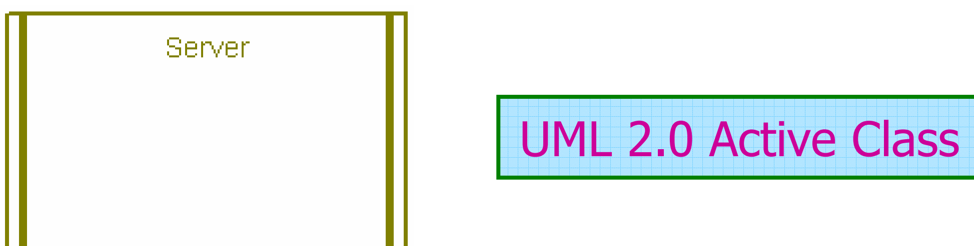
© Telelogic AB

Active Classes

- In UML 1.x the unit of concurrency was called the Active Object, shown with a thick border



- In UML 2.0 the notation has changed to double vertical lines, and is called an Active Class



© Telelogic AB

Basic Definitions

- Urgency

Urgency refers to the nearness of a deadline

- Criticality

Criticality refers to the importance of the task's correct and timely completion

 © Telelogic AB

Basic Definitions

- Priority

Priority *is a numeric value used to determine which task, of the current ready-to-run task set will execute preferentially*

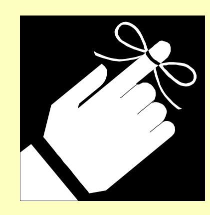
- Timeliness

Timeliness *refers to the ability of a task to predictably complete its execution prior to the elapse of its deadline*

© Telelogic AB Telelogic

Basic Definitions

- Deadline



*A **deadline** is a point in time at which the completion of an action becomes incorrect or irrelevant*

- Schedulability

*A task set is **schedulable** if it can be guaranteed that in all cases, all deadlines will be met*

© Telelogic AB

Basic Definitions

- Arrival Pattern

*The **arrival pattern** for a task or triggering event is either time-based (periodic) or event-based (aperiodic)*

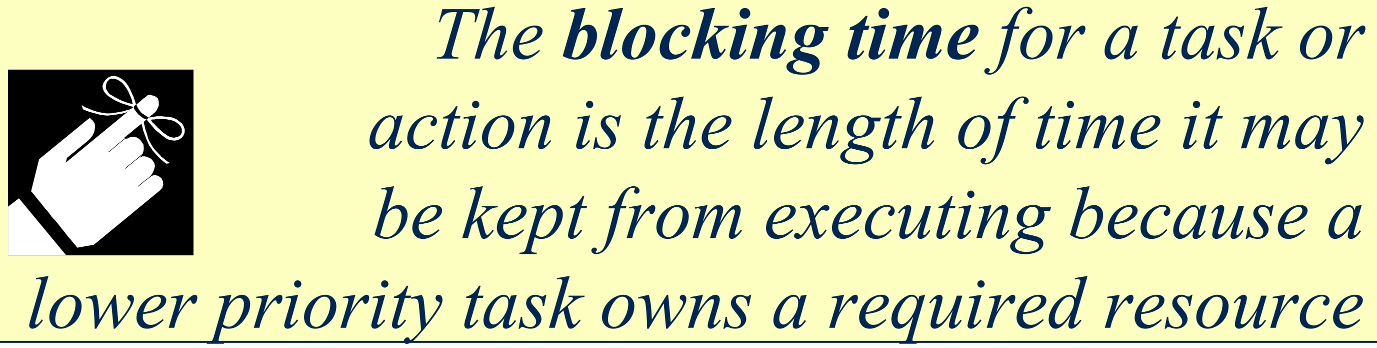
- Synchronization Pattern

***Synchronization pattern** refers to the how the tasks execute during a rendezvous, e.g. synchronous, balking, waiting, or timed*

© Telelogic AB

Basic Definitions

- Blocking Time



*The **blocking time** for a task or action is the length of time it may be kept from executing because a lower priority task owns a required resource*

- Execution Time

*The **execution time** for a task or action is the length of time it requires to complete execution*

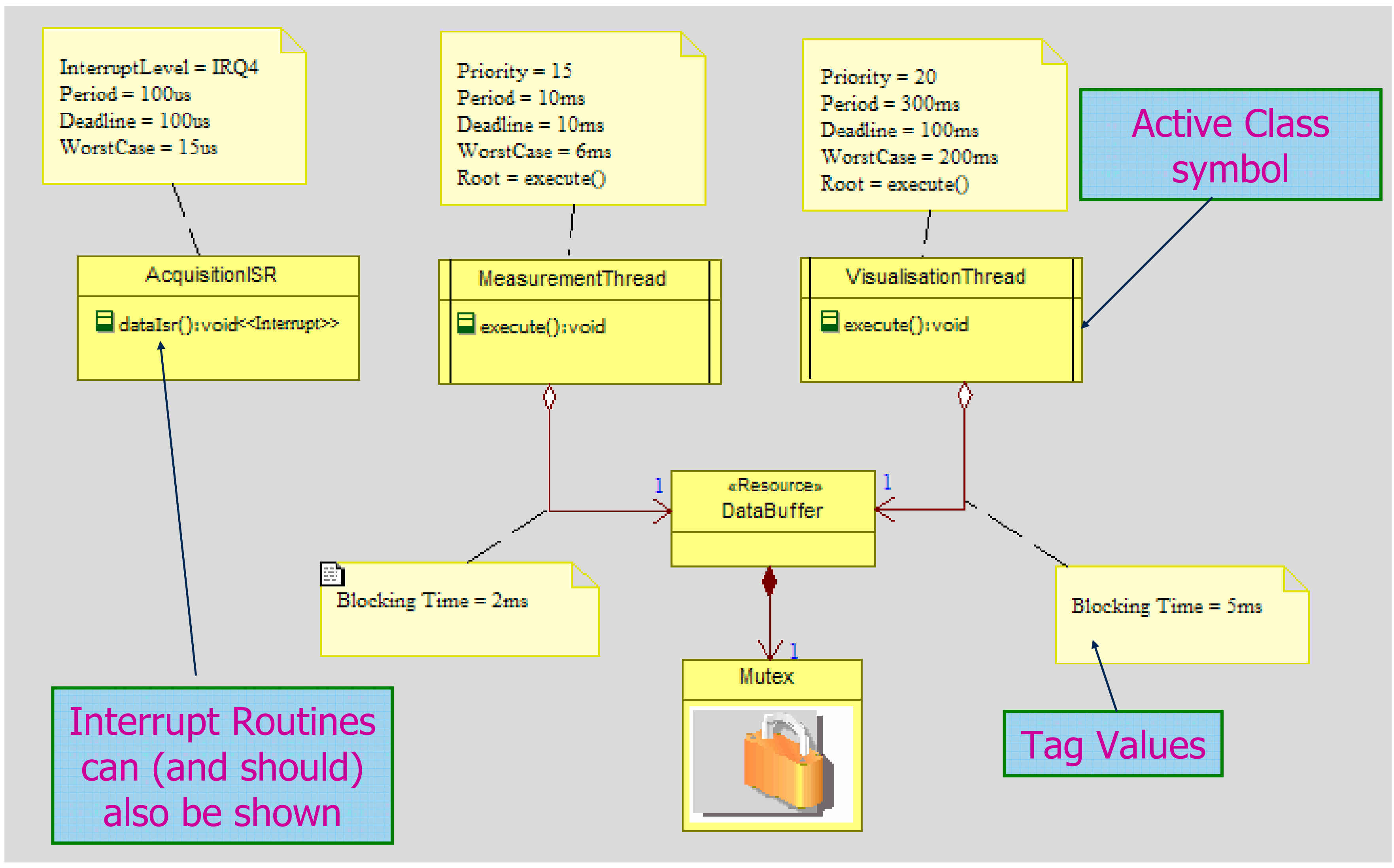
© Telelogic AB

Task Diagram

- A task diagram is a class diagram that shows only model elements related to the concurrency model
  - Active objects
  - Semaphore objects
  - Message and data queues
  - Constraints and tagged values
- May use opaque or transparent interfaces

© Telelogic AB

Task Diagram

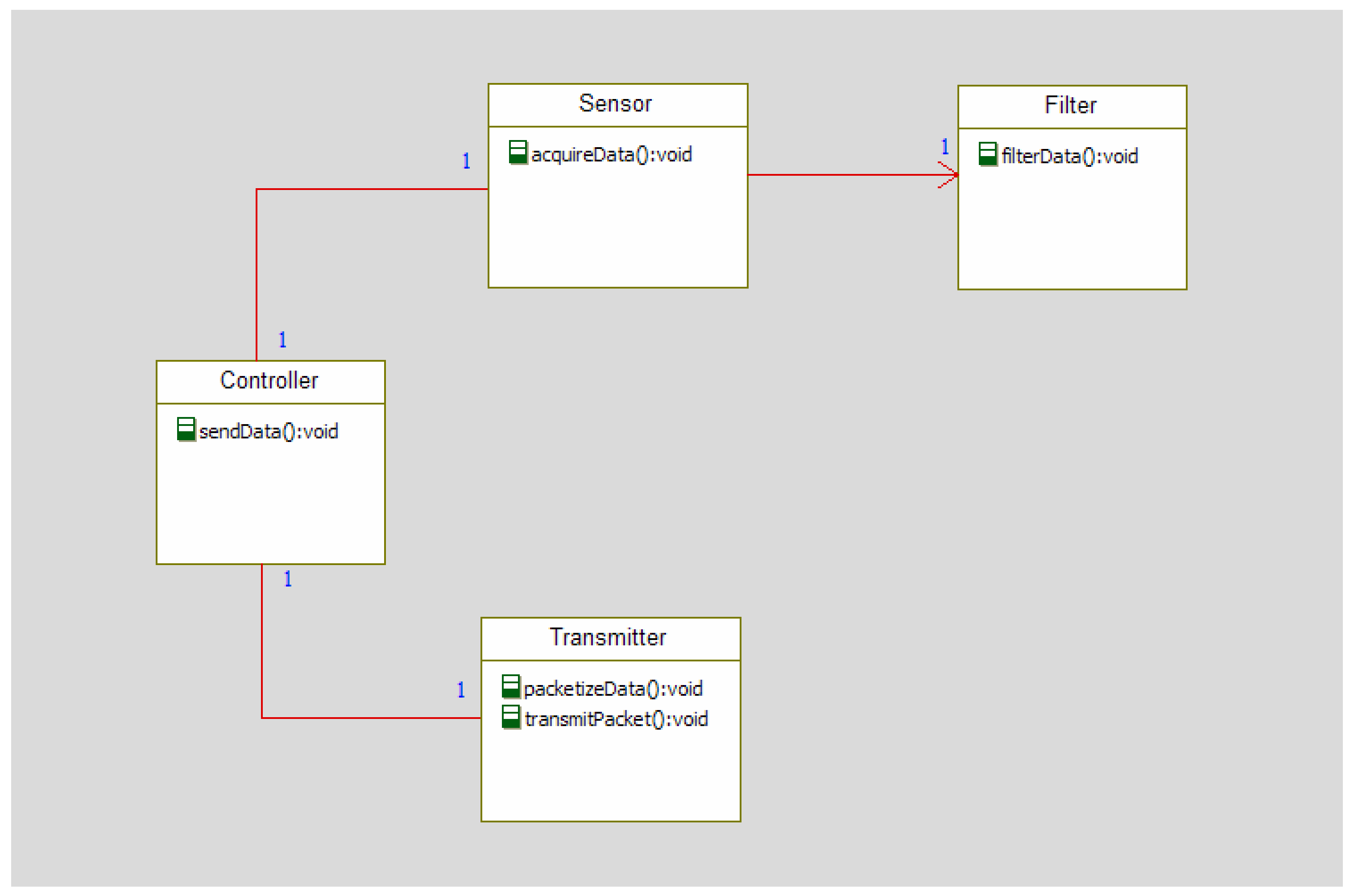


Task Performance Budgets

- The context defines the end-to-end performance requirements
- This determines the overall task budget
  - Computation of total budget may *not* just be simply adding up the times due to concurrency
- Individual operations and actions within tasks must be assigned portions of the overall budget
  - Action budgets should be checked during unit test
  - Action budgets should take into account potential blocking

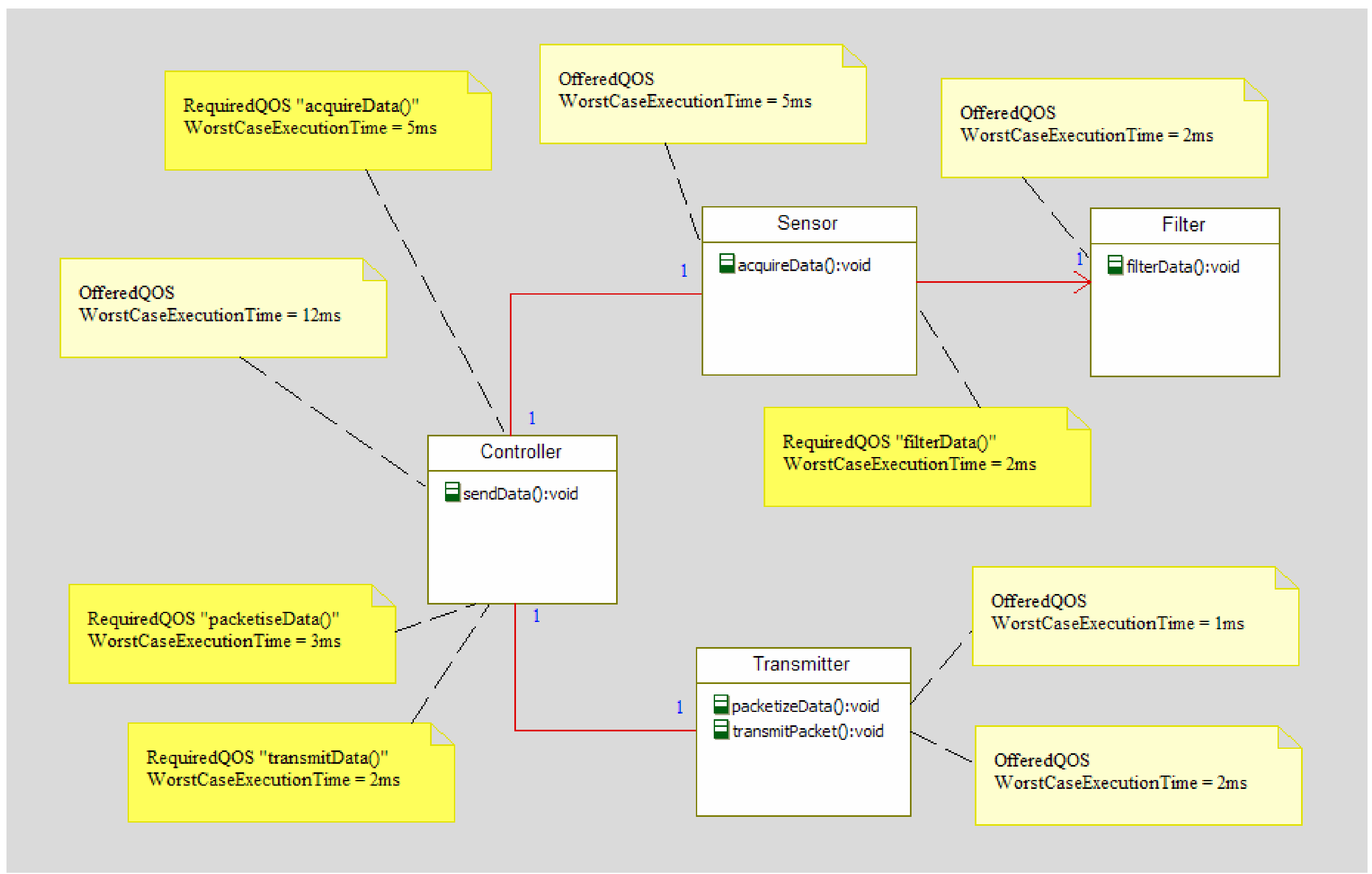
© Telelogic AB

Required / Offered QoS



© Telelogic AB

Required / Offered QoS



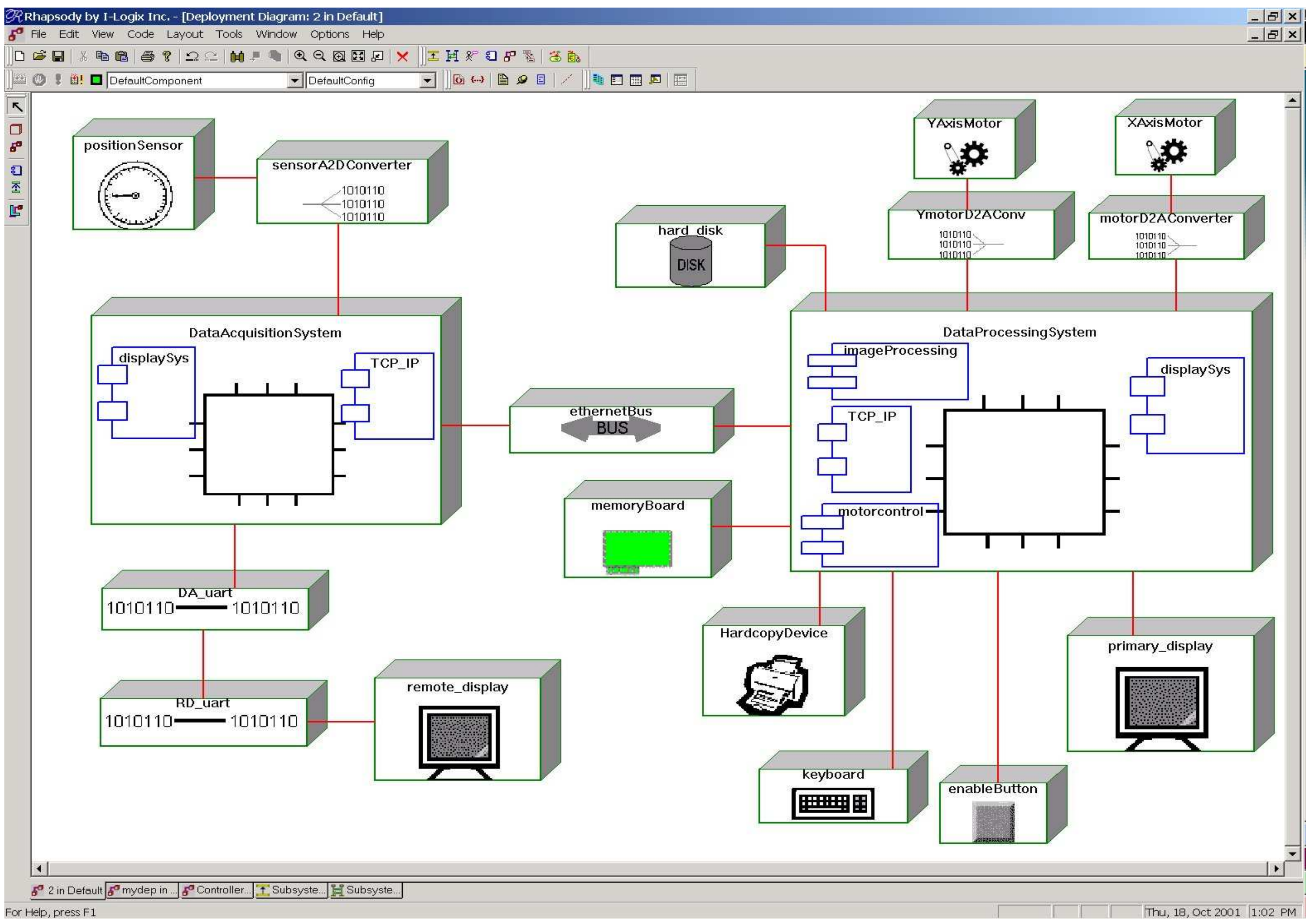
 © Telelogic AB Telelogic

Deployment Architecture

- Maps software components and subsystems to hardware
  – Represents a device as a node
  – Iconic stereotypes are common
- Identifies physical connections among devices
  – May be buses, networks, serial lines, etc
- Two primary strategies
  – Asymmetric
    - Design-time mapping of software elements to HW
  – Symmetric
    - Dynamic run-time mapping of software elements to HW

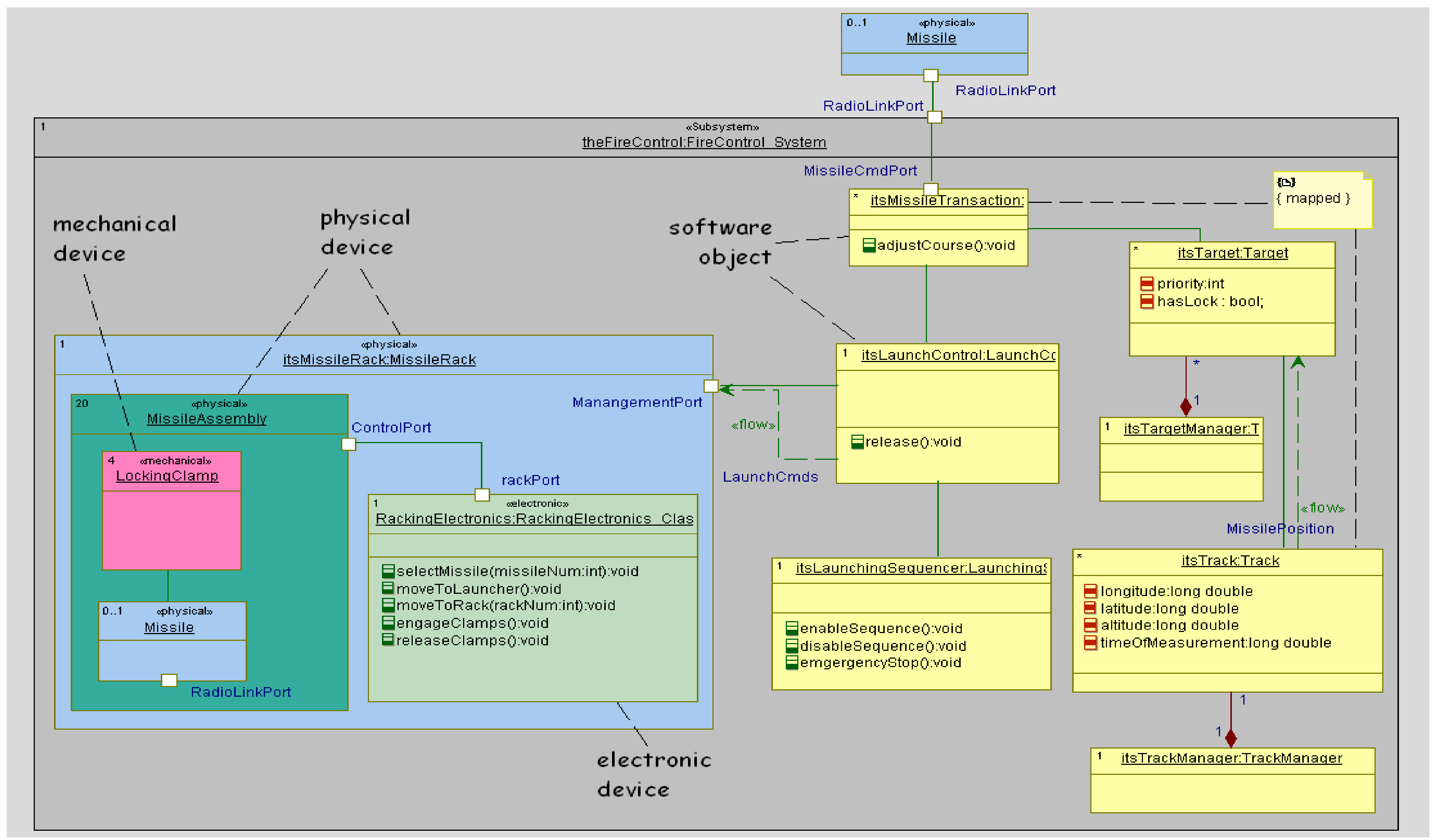
© Telelogic AB

Deployment Architecture



© Telelogic AB

Deployment via Class Diagram



© Telelogic AB

Telelogic

The UML Profile for Schedulability, Performance, and Time

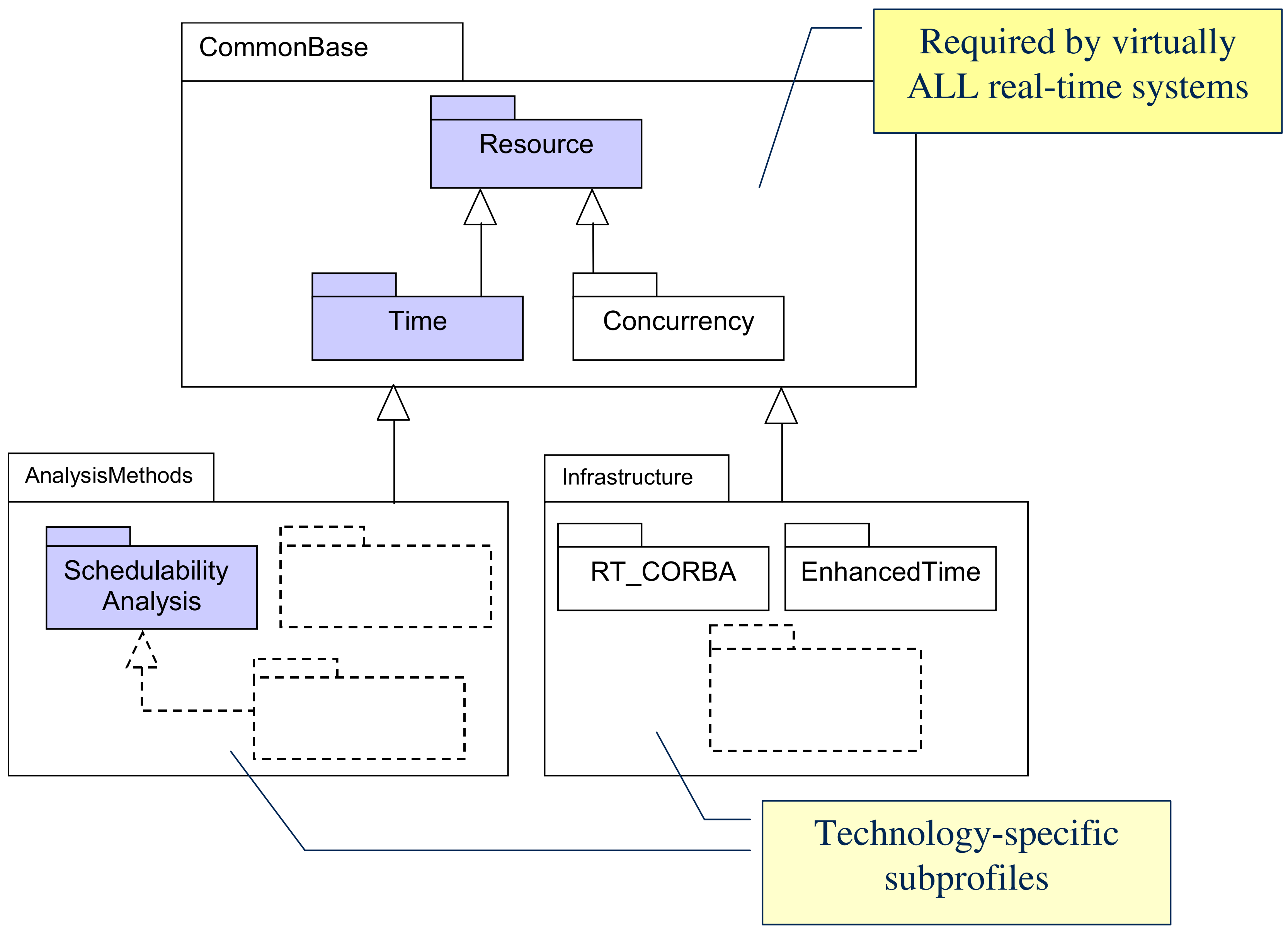
© Telelogic AB

General Approach

- Use light-weight extensions to add standard modeling approaches and elements
  - Stereotypes, e.g. resources
  - Tagged values, e.g. QoS properties
- Divide submission into sub-profiles to allow easier comprehension and usage of relevant parts

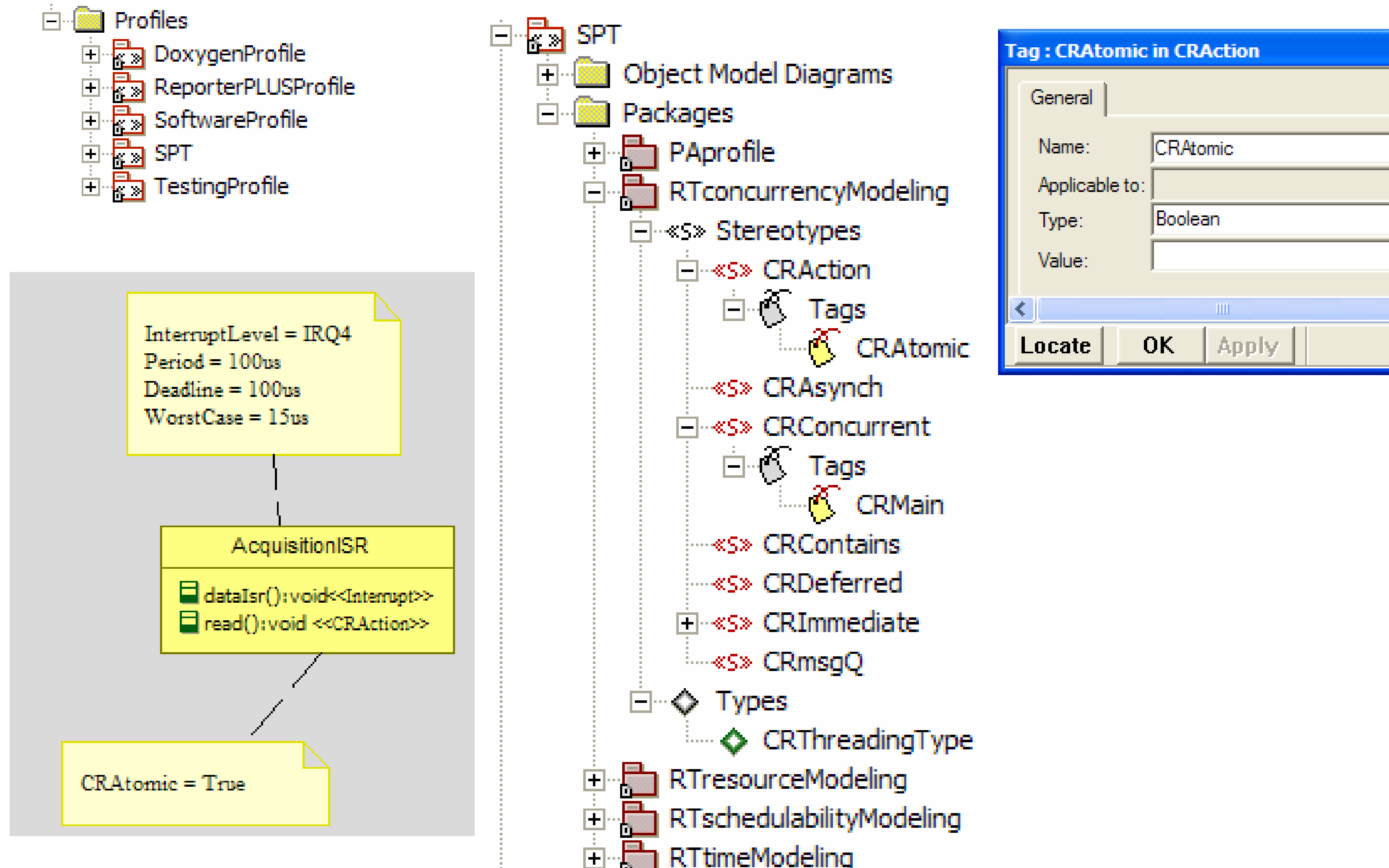
 © Telelogic AB

RT Profile Structure

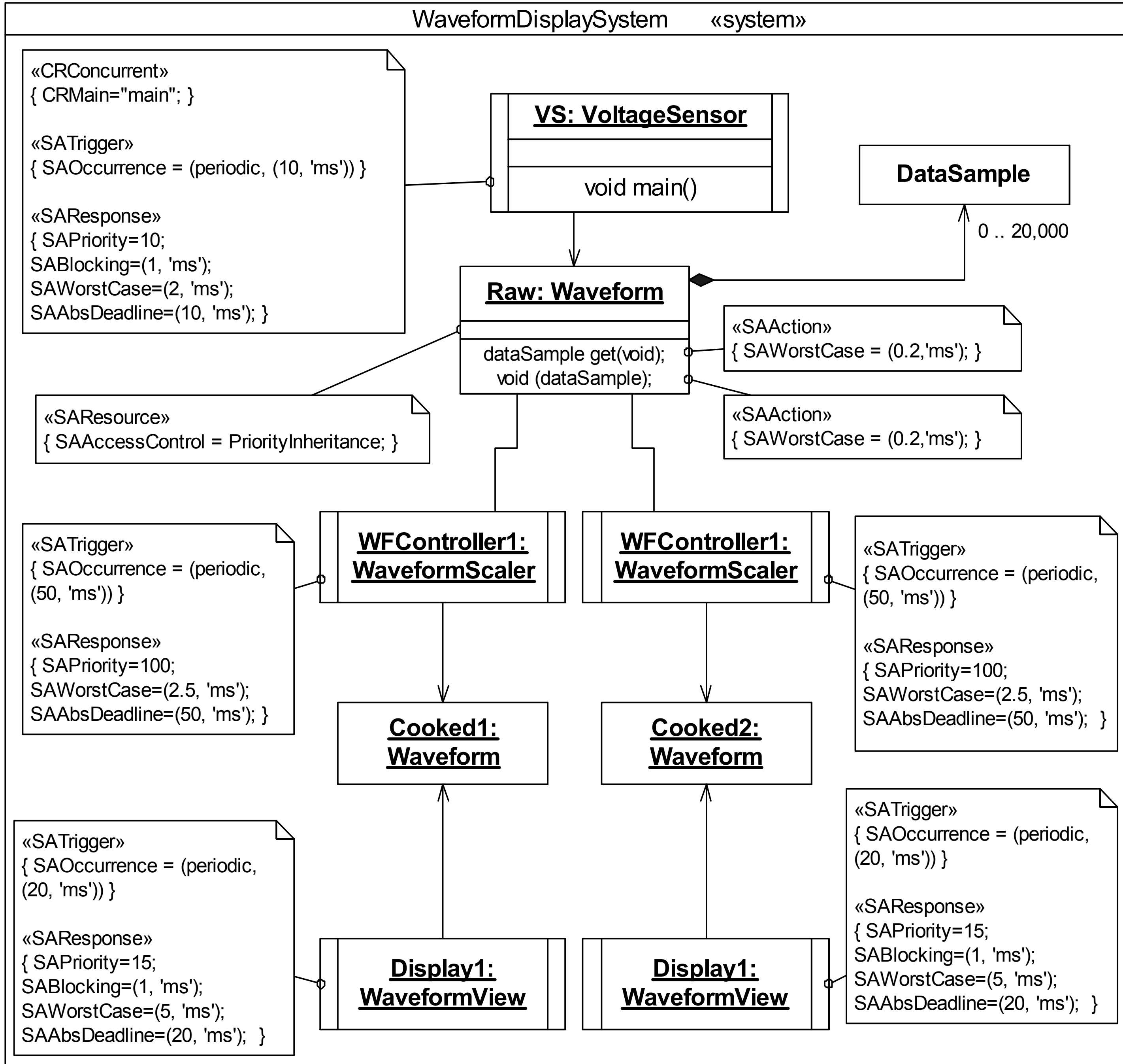


© Telelogic AB

SPT Profile

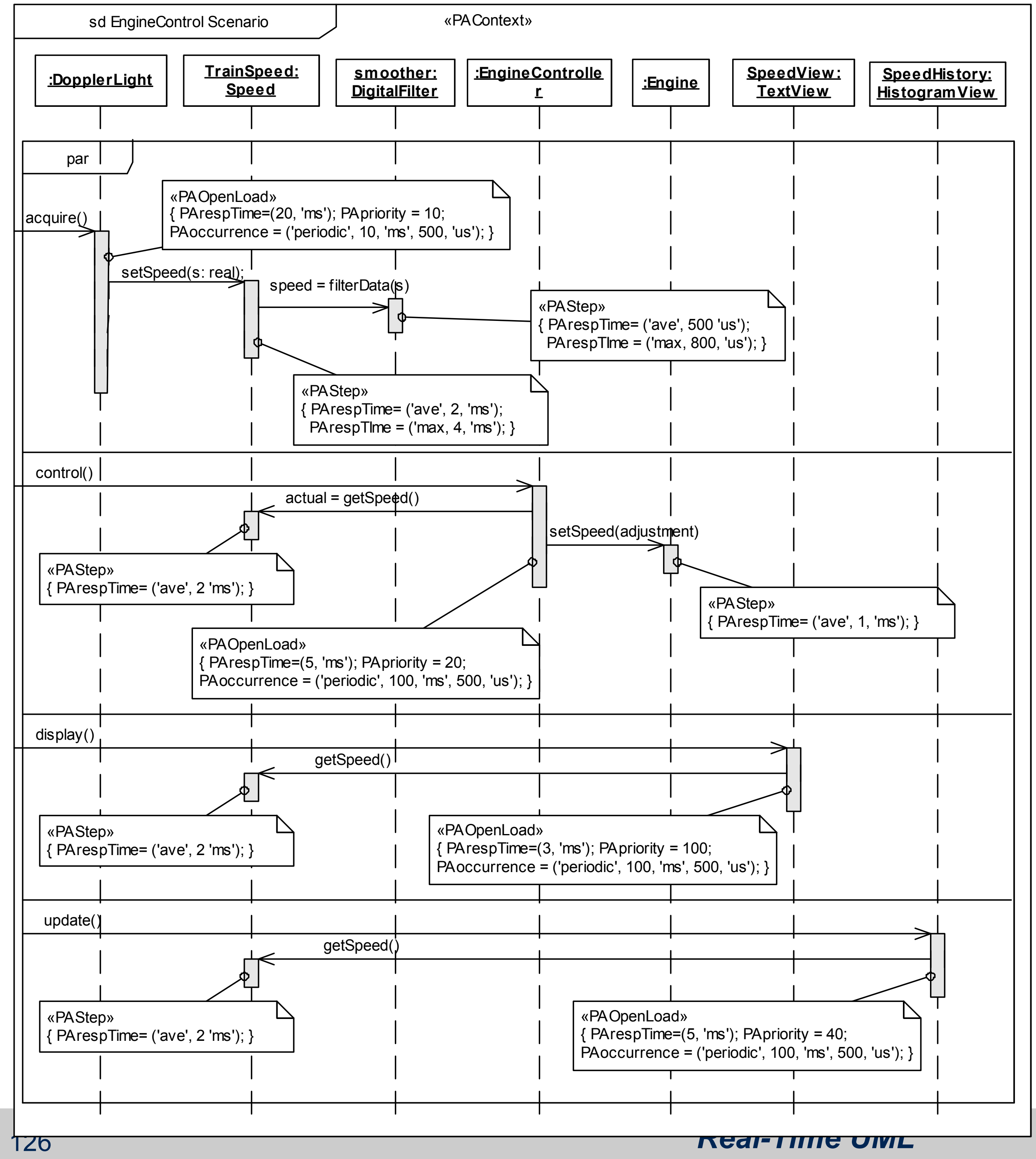


Example



© Telelogic AB Telelogic

Example



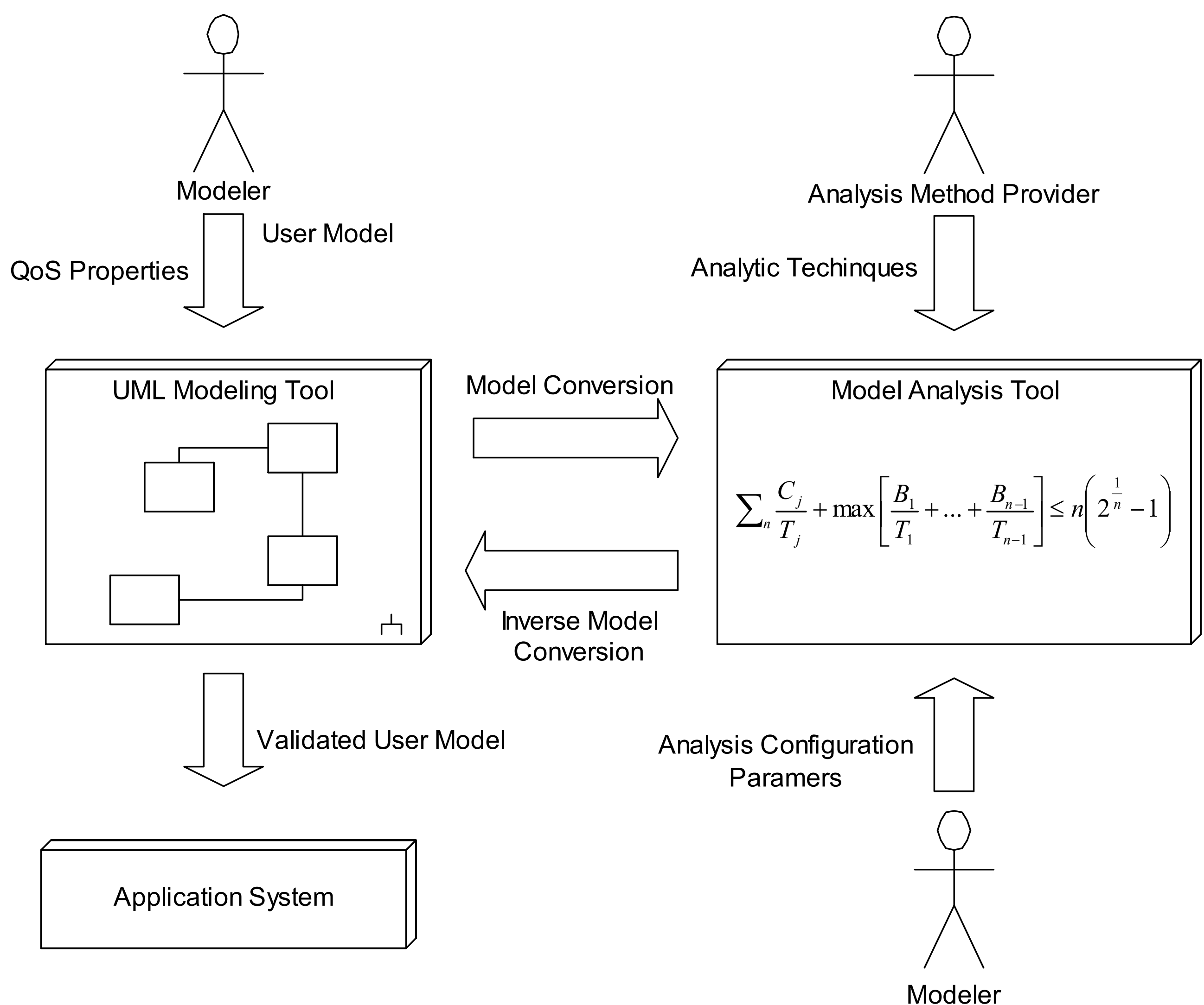
© Telelogic AB
Telelogic

GIGO

- Select the appropriate stereotypes and tags of the schedulability model to match the kind of analysis desired
  - Global RMA
    - Elements: active objects, resources
    - Tags: execution time, deadline, period, priority, blocking time, priority ceiling
  - Detailed RMA
    - Elements: active objects, resources, actions, events, scenarios, scenario steps, messages
    - Tags: execution time, deadline, period, priority, blocking time, priority ceiling
  - Simulation
    - Depends on particular approach
  - etc

© Telelogic AB

Model Processing



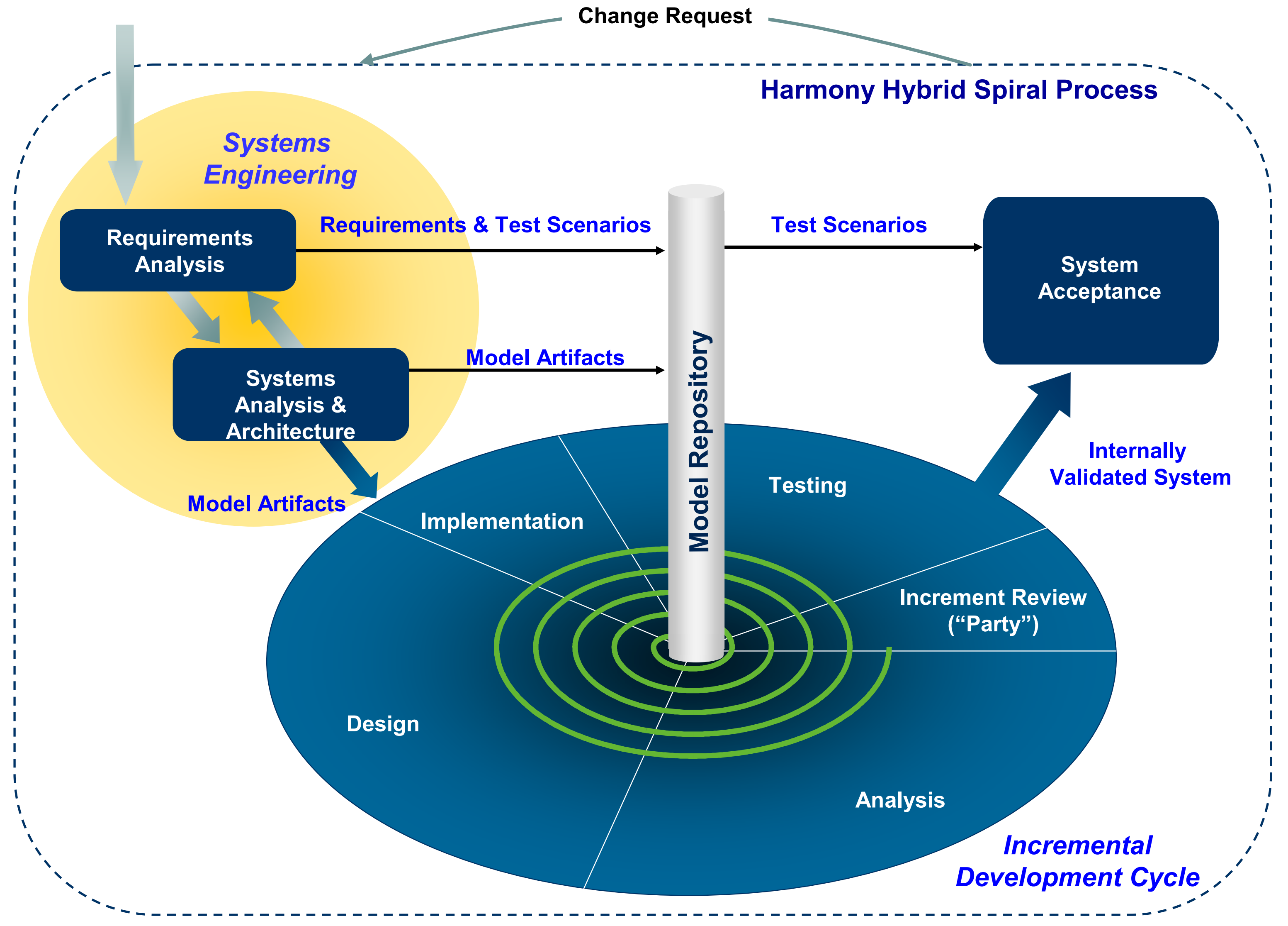
$$\sum_n \frac{C_j}{T_j} + \max\left[\frac{B_1}{T_1} + \ldots + \frac{B_{n-1}}{T_{n-1}}\right] \leq n\left(2^{\frac{1}{n}} - 1\right)$$

© Telelogic AB
Telelogic

Harmony™ Systems to Software Process

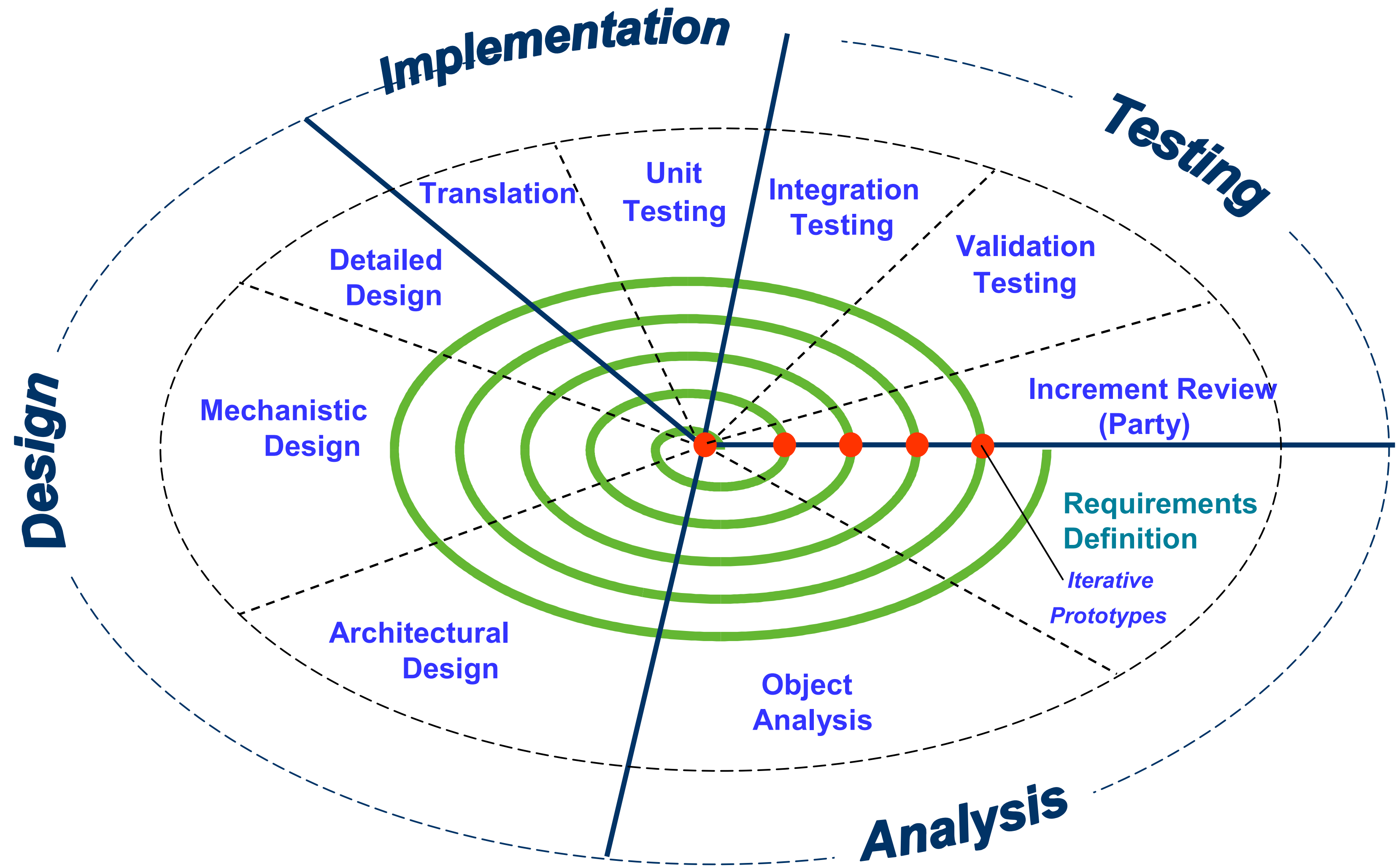
© Telelogic AB

Harmony Process Hybrid Spiral (General form)



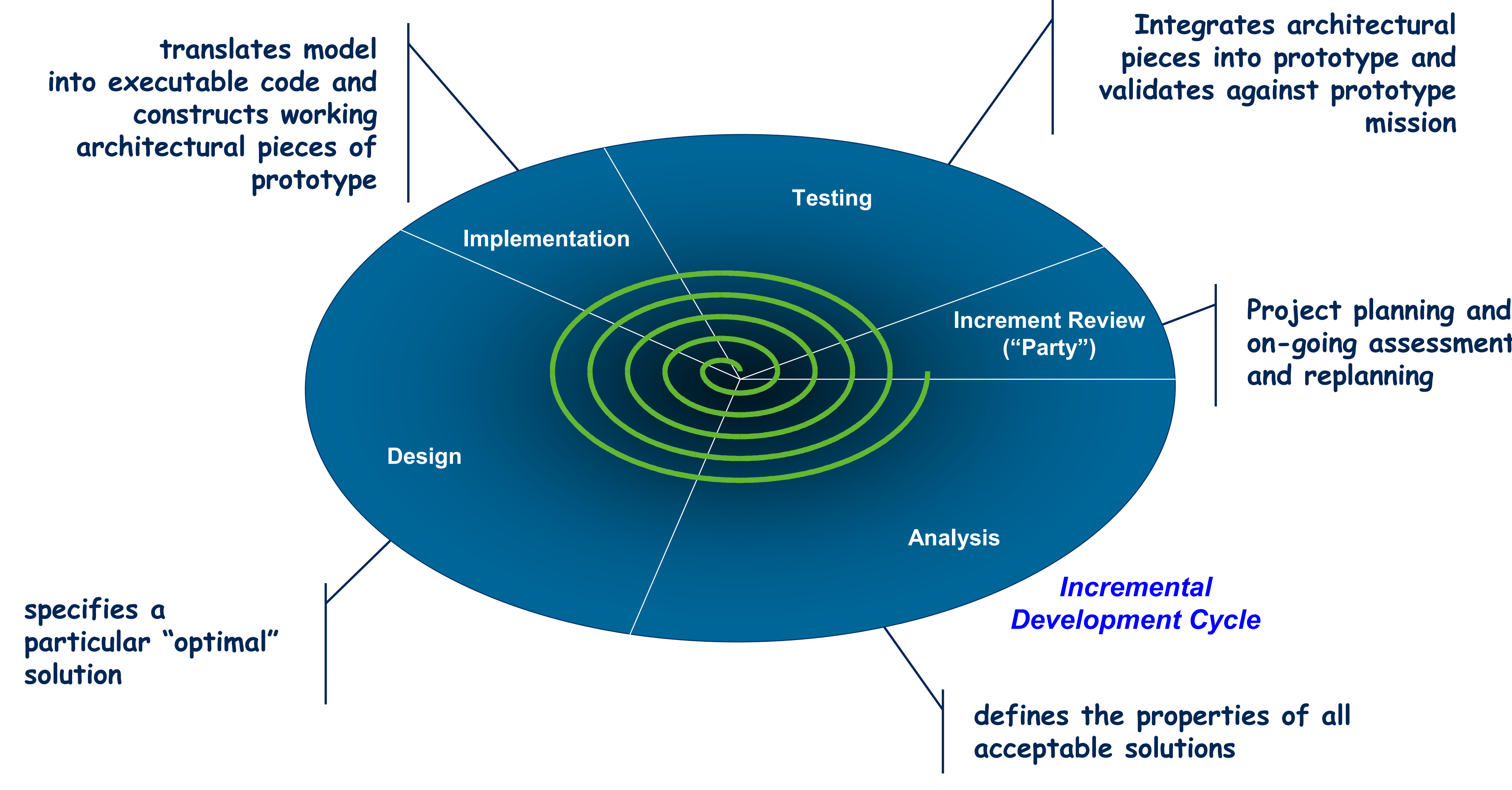
© Telelogic AB Telelogic

Harmony-SW Process



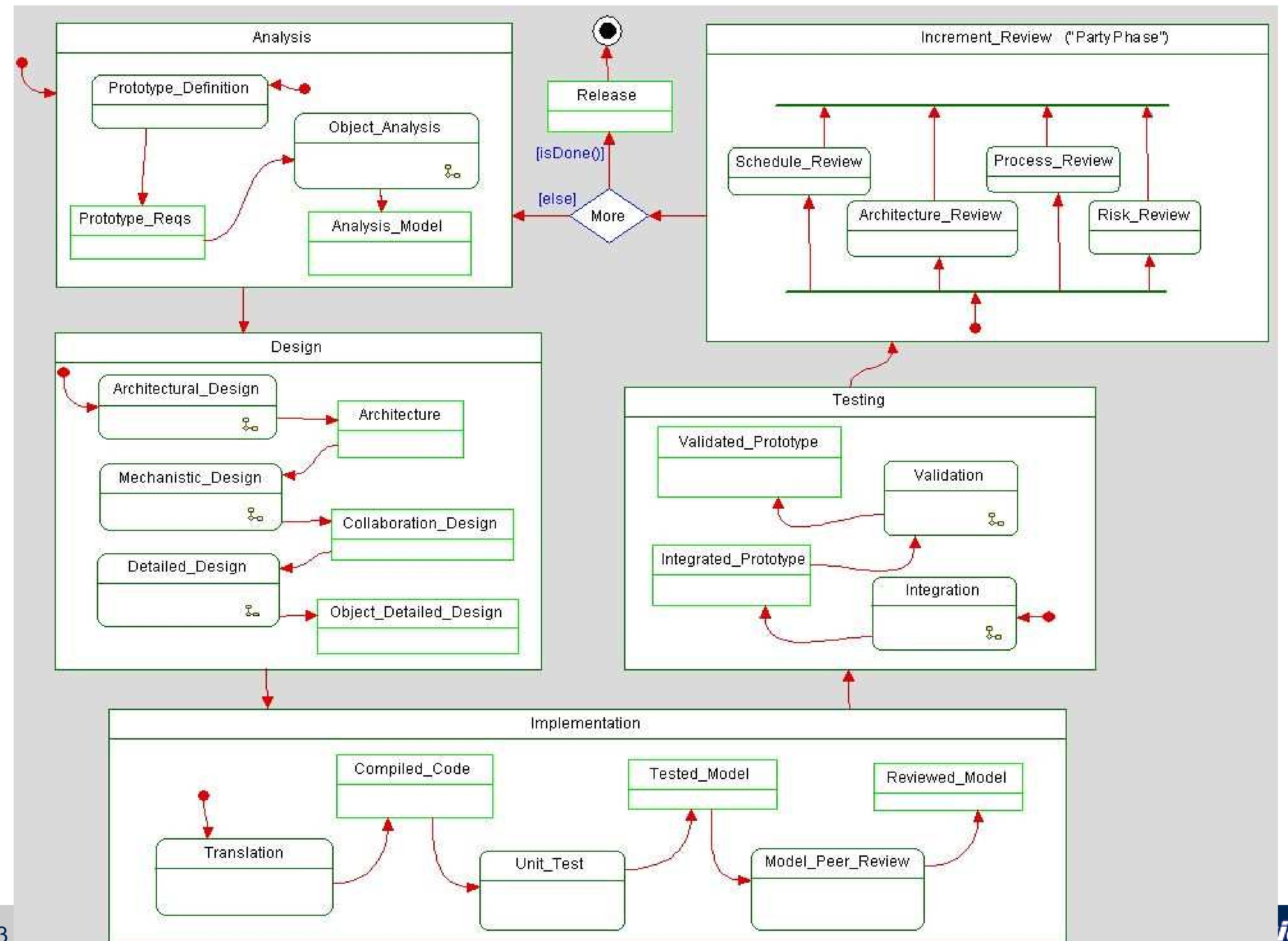
© Telelogic AB

Microcycle Flow

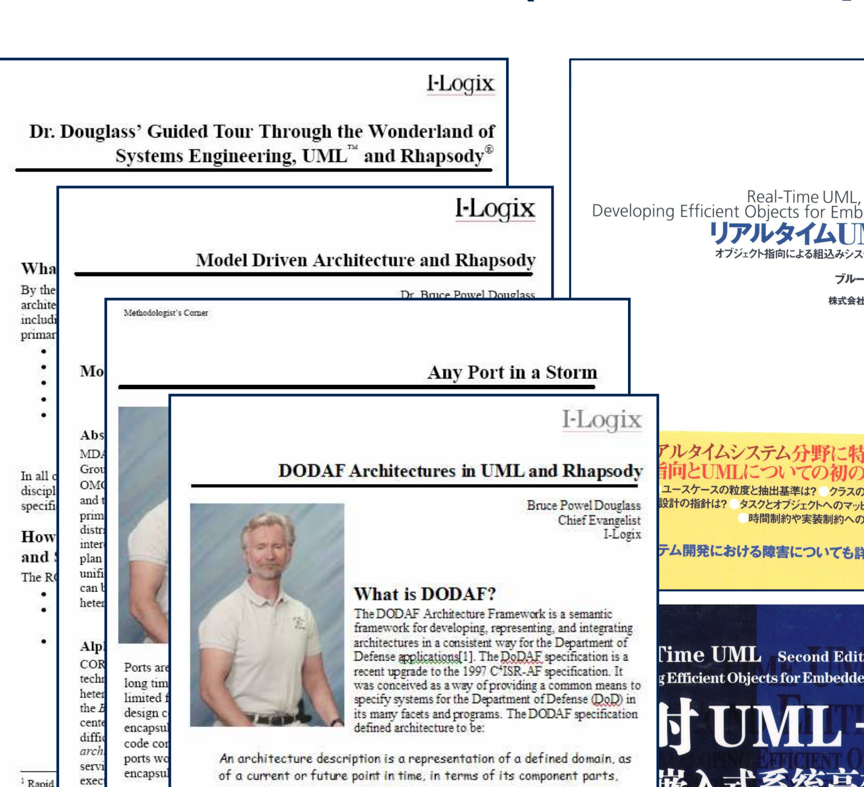


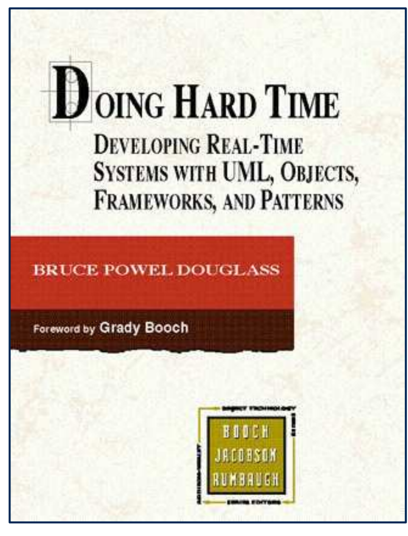
© Telelogic AB

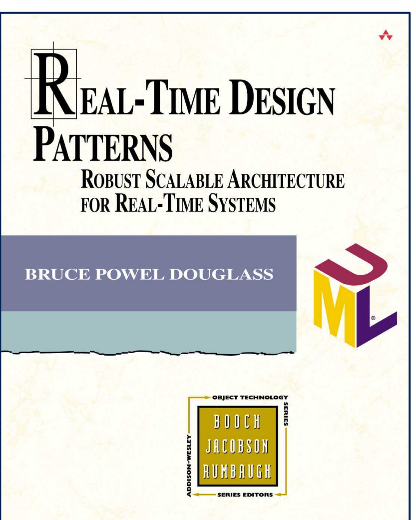
Harmony Incremental Spiral Workflows



References (For white papers see www.ilogix.com)







I-Logix

DODAF Architectures in UML and Rhapsody

Bruce Powel Douglass
Chief Evangelist
I-Logix

What is DODAF?

The DODAF Architecture Framework is a semantic framework for developing, representing, and integrating architectures in a consistent way for the Department of Defense applications[1]. The DoDAF specification is a recent upgrade to the 1997 C4ISR-AF specification. It was conceived as a way of providing a common means to specify systems for the Department of Defense (DoD) in its many facets and programs. The DODAF specification defined architecture to be:

An architecture description is a representation of a defined domain, as of a current or future point in time, in terms of its component parts, what those parts do, how the parts relate to each other, and the rules and constraints under which the parts function. What constitutes each of the elements of this definition depends on the degree of detail of interest. For example, domains can be at any level, from DoD as a whole down to individual functional areas or groups of functional areas. Component parts can be anything from "U.S. Air Force" as a component of DoD, down to a "satellite ground station" as a component part of a communications network, or "workstation A" as a component part of system "x." What those parts do can be as general as their high- level operational concept or as specific as the lowest- level action they perform. How the parts relate to each other can be as general as how organizations fit into a very high- level command structure or as specific as what frequency one unit uses in communicating with another. The rules and constraints under which they work can be as general as high-level doctrine or as specific as the e-mail standard they must use.

The term *architecture* is generally used both to refer to an architecture description and an architecture implementation. Hereafter in this document, the term *architecture* will be used as a shortened reference