UML2 et ses profils pour le temps-réel

ÉCOLE D'ÉTÉ TEMPS RÉEL 2005
GdR ARP – Thème StrQdS
Nancy, 13 - 16 Septembre 2005

Sébastien Gérard
CEA-LIST – Saclay
Sebastien.Gerard@cea.fr

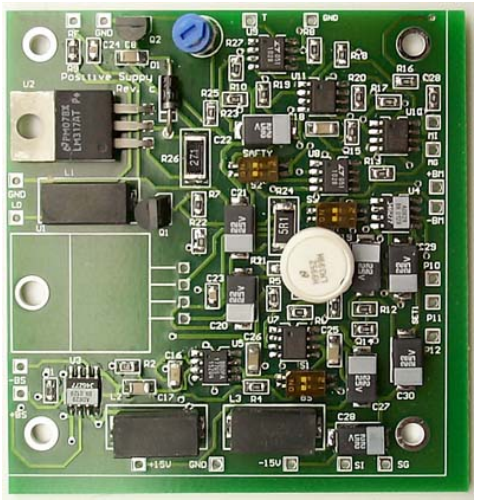
Agenda

- Introduction on MDE and UML2
- Native concepts for RT in UML2
- The UML profile for SPT specification
- Conclusions and perspectives

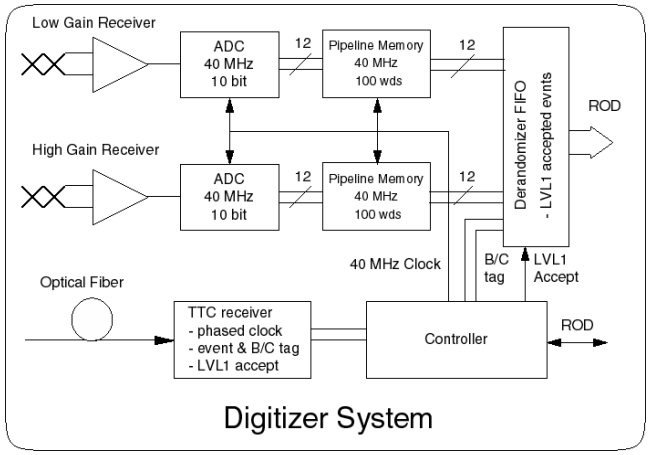
Extracted from B. Selic presentation during Summer School
MDD For DRES 2004 (Brest, September 2004)

Engineering Models

- **Engineering model**
  - A reduced representation of some system that highlights the properties of interest from a given viewpoint



Modeled system



Functional Model

- **We don't see everything at once**
- **We use a representation (notation) that is easily understood for the purpose on hand**

Expected benefits for modeling

- **To deal with complexity**
  - Concrete representation of knowledge and ideas about a system being developed
    ➔ improve communication around a problem
  - Abstract a problem (omits some aspects) to focus on some particular points of interest
    ➔ improve understability of a problem
- **To increase control of complexity**
  - Via a set of nearly independent views of a model
    - ✓ Separation of concerns (e.g. “Aspect Oriented Modeling”)
  - A model may be expressed at different level of fidelity (abstraction)
- **To improve communication to foster information sharing and reuse!**
  - A model is often best suited than a long speech !
  - Graphical notation is often better suited than textual one
- **To have seamless of process based on a pivot model paradigm**

Extracted from B. Selic presentation during Summer School
MDD For DRES 2004 (Brest, September 2004)

Characteristics of Useful Models

- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Inexpensive**
  - Much cheaper to construct and study than the modeled system

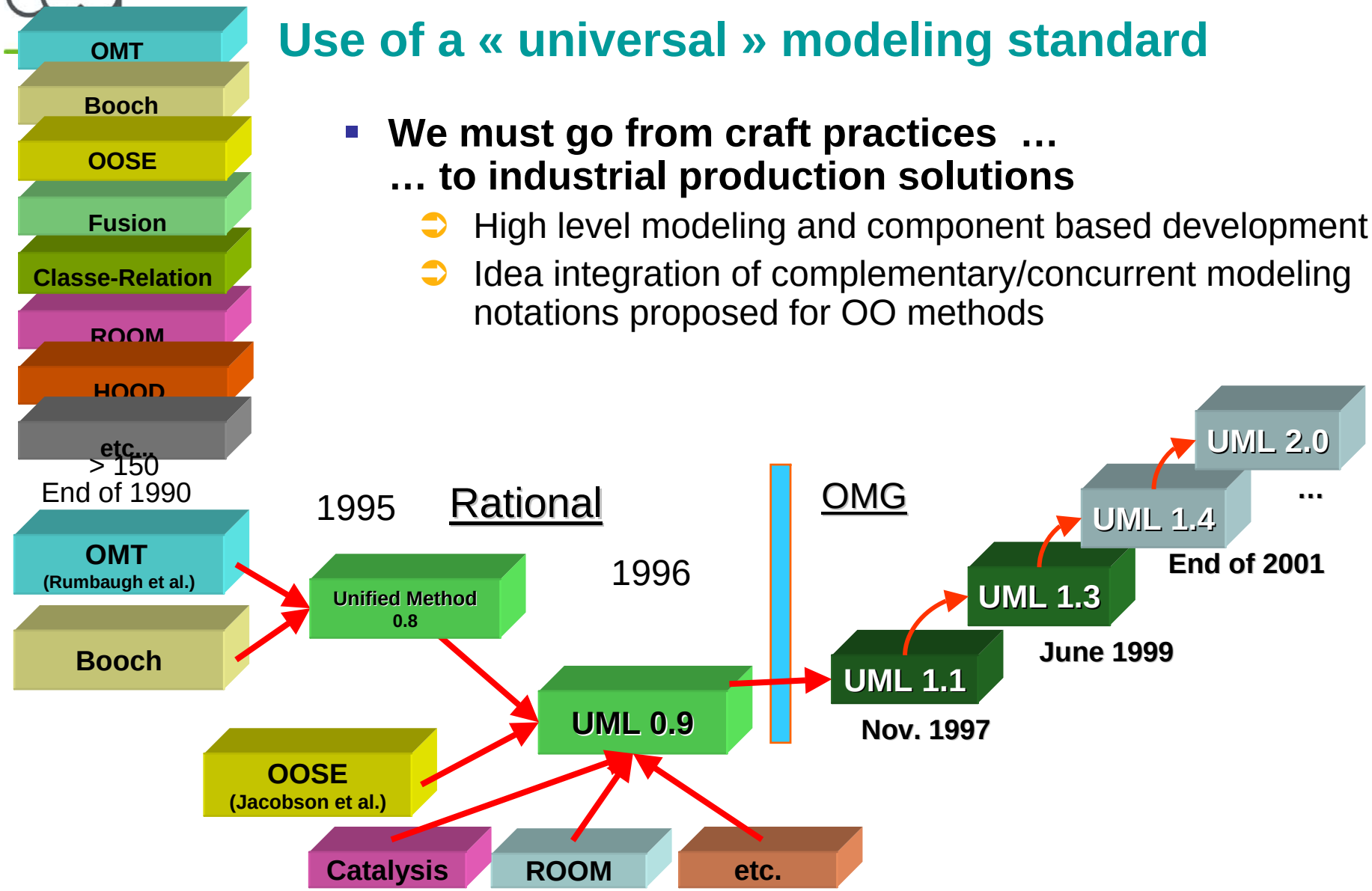
To be useful, engineering models must satisfy all of these characteristics!

The OMG organism (www.omg.org)

- **Initially centered on CORBA around the "Object Driven Architecture"**
- **Takes UML standardization**
  → becomes more and more important
- **Introduction of the MOF to unify all object concepts for CORBA, UML, etc.**
- **6 technical meetings by years (US, Europe, Asia)**
  - Work orientation is presented during the meeting
  - Only one vote by legal entity …

Use of a « universal » modeling standard

- **We must go from craft practices …**
  … to industrial production solutions
  - High level modeling and component based development
  - Idea integration of complementary/concurrent modeling notations proposed for OO methods



Unified Modeling Language

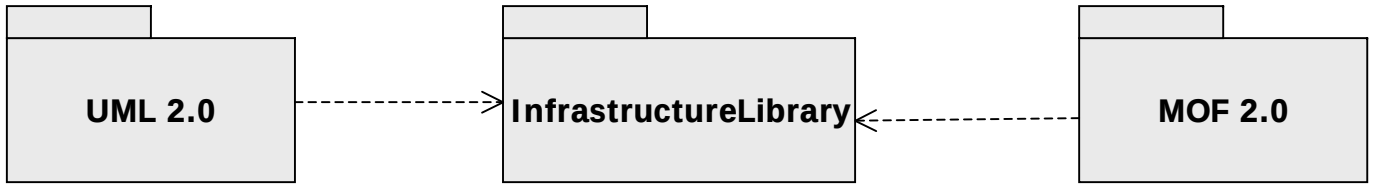
- **UML is a language ➔ syntax + semantics**
  - syntax = rules by which language elements (e.g., words) are assembled into expressions (e.g., phrases, clauses)
  - semantics = rules by which syntactic expressions are assigned meanings

Four RFPs for a new UML standard ➔UML2

- **UML 2.0 Infrastructure RFP**
  - Improve UML alignment with other OMG modeling standards
    - ✓ E.g. MOF and XMI
  - Make the UML easier to understand, implement and extend
  - Improve extenssibiity mechanisms of the UML
- **UML 2.0 Superstructure RFP**
  - Support Component-Based Software Engineering
  - Clarify the semantics of the generalization, dependency, and association relationships
  - Support encapsulation and scalability in behavioral modeling
    - ✓ E.g. for state machines and interactions
  - Remove restrictions on activity graph modeling
- **UML 2.0 OCL RFP**
  - It solicits proposals for defining an OCL metamodel consistent with the UML
- **UML 2.0 Diagram Interchange RFP**
  - It focuses on the problem of UML diagram interchange

Outlines of the infrastructure

- **Main purpouse**
  - Align UML foundation and MOF



- **The Infrastructure Library**
  - Core package defines basic meta-languages for other MMs (eg. UML, CWM, …)
  - Profiles package specifies metamodel extension mechanisms for UML
- **UML can be extended in two ways:**
  - Using Profiles for UML dialect definition
    - ✓ Specialisation of the existing UML meta-model
  - Reusing part of the *InfrastructureLibrary* for defining a new language of the UML families
    - ✓ New meta-models related to the UML

Language formalism for the UML2

- **The UML specification is defined using a meta-modeling approach**
  - Less formal, but more intuitive & pragmatic
- **The specification consists of different packages**
  - Focussed on a particular aspects of the language (12 chapters)
    - ✓ Classes, Actions, …
- **Package organization in the document**
  - Overview
  - Abstract syntax
    - ✓ Defines the syntax in a notation independent way
  - Class descriptions
    - ✓ Informal description
    - ✓ Attributes & Associations
    - ✓ Semantics (using natural language)
    - ✓ **Semantic Variation Points***
    - ✓ Notation
    - ✓ Presentation Options*
    - ✓ Style Guidelines*, Examples*, Rationale, Changes from UML 1.4
  - Diagrams

* Optional parts

Agenda

- Introduction on MDE and UML2
- **Native concepts for RT in UML2**
- **The UML profile for SPT specification**
- **Conclusions and perspectives**

Superstrucutre presentation agenda

- Part I: Structure
  - **Chapter 8. Components**
  - Chapter 9. Deployments
- Part II: Behavior
  - Chapter 11. Actions
  - Chapter 12. Activities
  - Chapter 14. Interactions
  - Chapter 15. State Machines

Outlines of the Component concept

- **Self contained unit that encapsulates the state and behavior of a number of classifiers by specifying:**
  - Interfaces
    - ✓ Provided interfaces
      - → Formal contract of the services available for clients.
    - ✓ Required interfaces
      - → Requirements from other components or services in the system.
  - Or ports
    - ✓ Typed by required or/and provided interfaces

- **Substitutable unit that can be replaced at design time or run-time by a component that offers equivalent functionality based on compatibility of its interfaces.**

Superstrucutre presentation agenda

- Part I: Structure
  - Chapter 8. Components
  - **Chapter 9. Deployments**
- Part II: Behavior
  - Chapter 11. Actions
  - Chapter 12. Activities
  - Chapter 14. Interactions
  - Chapter 15. State Machines

Details of the deployments concepts

- **Define the execution architecture of systems that represent the assignment of software artifacts to nodes**
- **Main related concepts**
  - Artifact
    - ✓ Specifies a physical piece of information
    - ✓ E.g. model files, source files, binary, …
  - Device
    - ✓ Model physical computational resource with processing capability
    - ✓ May support artifacts deployment for execution
    - ✓ May consist of other devices
  - ExecutionEnvironment
    - ✓ Implements a standard set of services that Components require at execution
    - ✓ E.g. «OS», «workflow engine», «database system»
  - DeploymentSpecification
    - ✓ General mechanism to parameterize a Deployment relationship

Superstrucutre presentation agenda

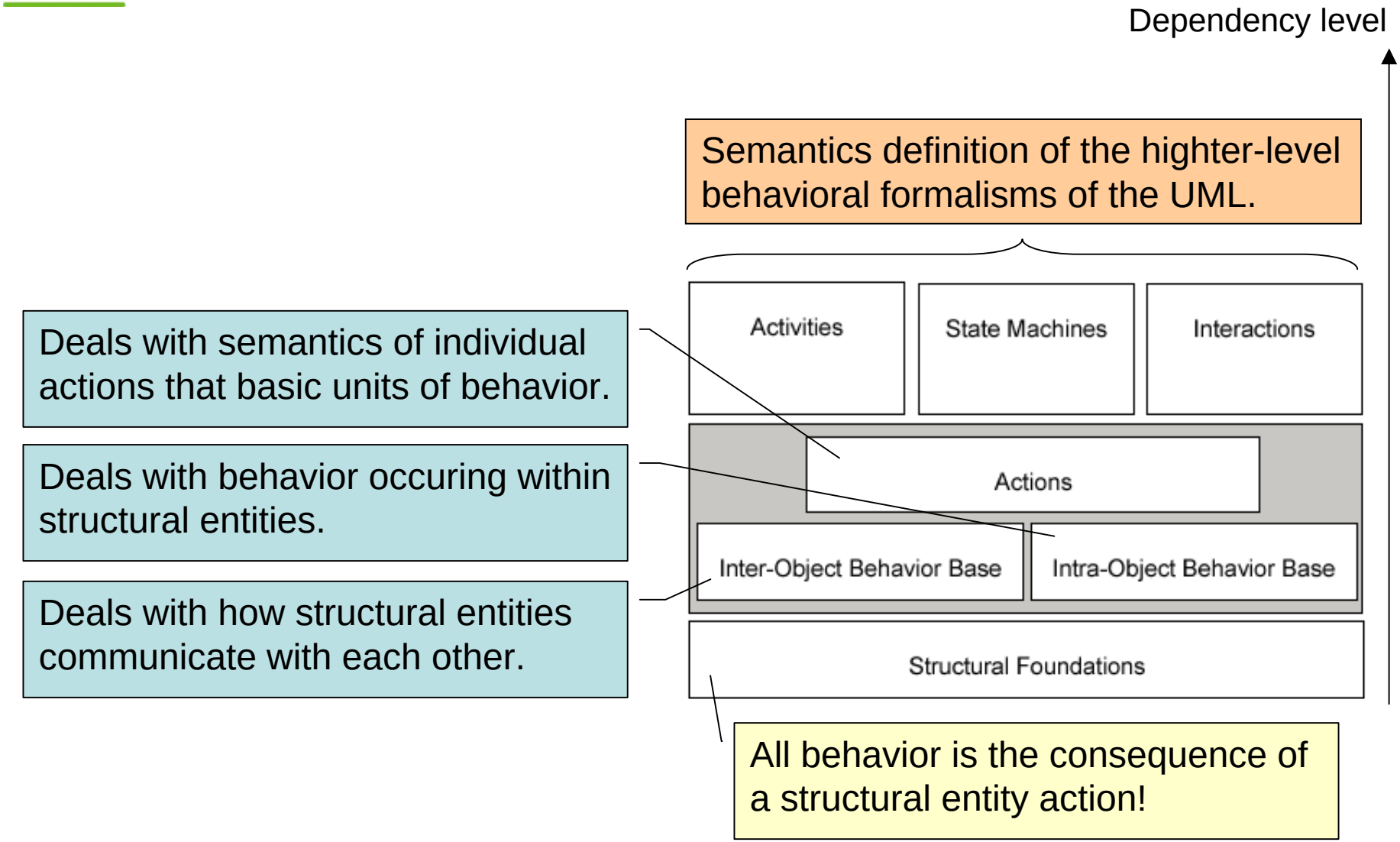
- Part I: Structure
  - Chapter 8. Components
  - Chapter 9. Deployments
- **Part II: Behavior**
  - Chapter 11. Actions
  - Chapter 12. Activities
  - Chapter 14. Interactions
  - Chapter 15. State Machines

Run-time semantics premisses

- **Assumption 1**
  - All behavior in a modeled system is ultimately caused by actions executed by so-called “active” objects.
    - ✓ It includes behaviors that are also objects in UML2.

- **Assumption 2**
  - UML2 behaviors are either event-driven or discrete.

Architecture of the UML semantics areas

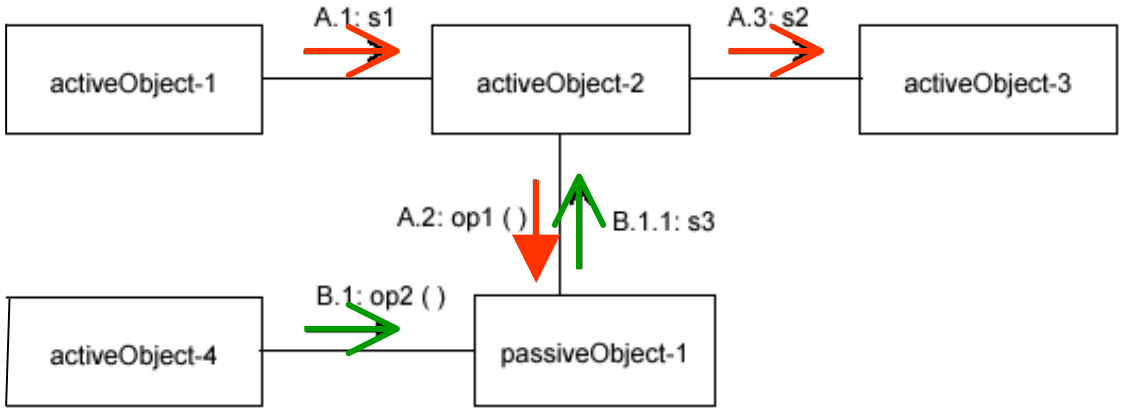


Activities, State Machines and Interactions

- **Activities**
  - Model sequence, conditions and inputs/ouputs for invoking other behaviors.
- **State Machines**
  - Denote how events modify the state of objects and trigger other behaviors.
- **Interactions**
  - Describe exchanges of messages between objects and triggering other behaviors.

The Basic Causality Model

- **Specification of how things happen at run time.**
- **Principle**
  - Objects respond to messages that are generated by objects executing communication actions.
  - When messages arrive, the receiving objects eventually respond by executing the behavior that is matched to that message.
  - Whithin behavior execution, messages may be sent…
- **Example**



Behavior invocation

- **Two possibilities to call behaviors**
  - Directly (➔ CallBehaviorAction)
    - ✓ Used to model call to libraries of usual functional programming language (e.g. the C language)
  - Indirectly via an operation call on a given object
    - ✓ Behavior describes the dynamic of the method implementing the called operation.
    - ✓ Executed behavior is determined dynamically by the target object.

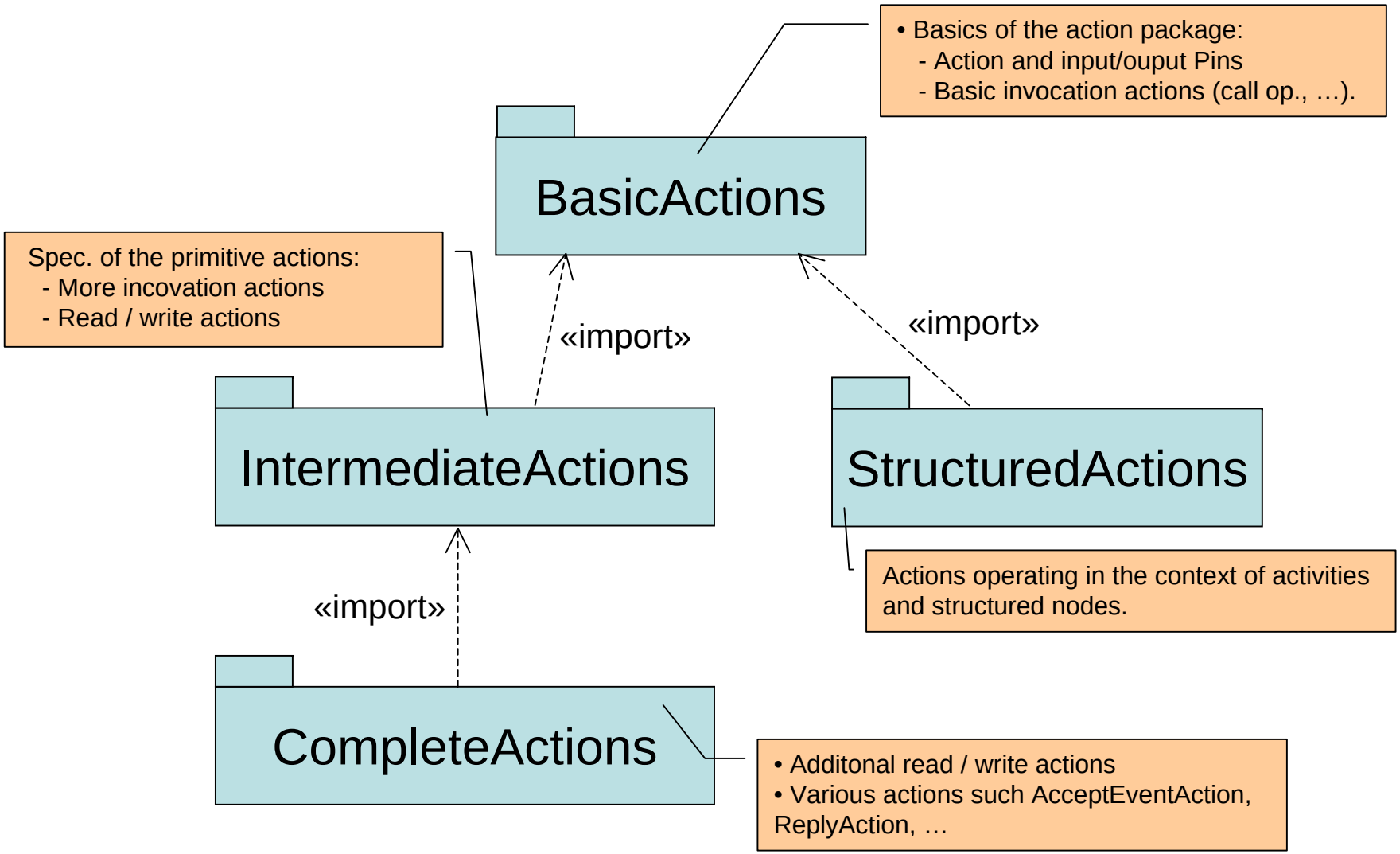
Superstructure presentation agenda

- Part I: Structure
  - Chapter 8. Components
  - Chapter 9. Deployments
- **Part II: Behavior**
  - **Chapter 11. Actions**
  - Chapter 12. Activities
  - Chapter 14. Interactions
  - Chapter 15. State Machines

The Action packages

- **The Actions chapter is concerned with the semantics of individual primitive actions**
- **Action = fundamental unit of behavior specification ensuring UML models to be fully executable**
- **Principle is that actions exchange control / data flows via input and output pins**

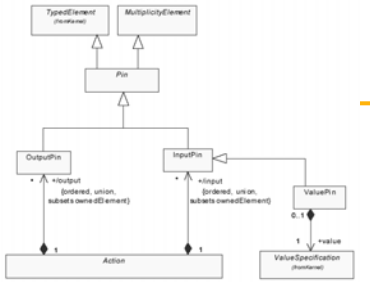
Several levels of action descriptions



Surface action language

- **About actions, the standard focussed in defining the abstract syntax and its semantics**
  - Do not provide any concrete (or surface) language mapping with this proposed semantics!
  - For depicting executable UML models, need to be defined!
- **Main features of the language definition**
  - Necessary to choose/create a notation (textual or/and graphical)
    - ✓ Notation is also called the concrete syntax of the language or surface action language.
  - Necessary to define the mapping with elements of the abstract syntax defined in the various elements of the Action sub-packages
  - Would encompass both primitive actions and the control mechanisms provided by behaviors (Statemachine, Activity, …).
  - May define higher-level constructs to the basic actions of the standard.
    - ✓ e.g. creating an object
      - → CreateObjectAction
        - » Creates an object that conforms to a statically specified classifier and puts it on an output pin at runtime.
        - » The action has no other effect (no behaviors are executed, no initial expressions are evaluated, …) ➔ The new object has no structural feature values and participates in no links.
        - » It is equivalent to the default constructor of C++ for example.
      - → Possible definition of a higher-level construct providing object creation with initialization as a single unit as a shorthand for several actions.

(BasicAction) – Action and Pins



- **Action**
  - Abstract class
    - ✓ All action executions will be executions of specific kinds of actions.
      - → e.g. CreateObjectAction, CallAction, …
  - Action context is setup by the classifier that owns the behavior containing the action and it determines:
    - ✓ When the action executes.
    - ✓ And what actual inputs are.
- **InputPin**
  - An action cannot start execution if an input pin has fewer values than the lower multiplicity.
  - The upper multiplicity determines how many values are consumed by a single execution of the action.
- **OutputPin**
  - Object nodes that deliver values to other actions through object flows.

(BasicAction) - Invocation actions

- ***InvocationAction* [abstract]**
  - Abstract class for the various actions that invoke behavior.
- ***CallAction* [abstract]**
  - Two communication modes:
    - ✓ Synchronous call
      - → The caller waits for completion of the invoked behavior.
      - → Possibility to manage exceptions that may be generated by the invoked behavior.
    - ✓ Asynchronous call
      - → The caller proceeds immediately and does not expect a return value.
  - CallOperationAction
    - ✓ Operation call to a target object, where it may cause the invocation of associated behavior.
  - CallBehaviorAction
    - ✓ Invokes a behavior directly rather than invoking a behavioral feature that, in turn, results in the invocation of that behavior.
    - ✓ Argument values of the action are available for the invoked behavior execution.
- **SendSignalAction**
  - Action creating signal instance from its inputs, and transmits it to **the** target object.
  - Targetted object may then:
    - ✓ Either fire a state machine transition **Or** execute an activity.
  - Argument values are available to the execution of associated behaviors.
  - Asynchronous communication: caller does not wait any response.

Superstrucutre presentation agenda

- Part I: Structure
  - Chapter 8. Components
  - Chapter 9. Deployments
- **Part II: Behavior**
  - Chapter 11. Actions
  - **Chapter 12. Activities**
  - Chapter 14. Interactions
  - Chapter 15. State Machines

Various levels of Activities

- **BasicActivities**
  - Support for traditional sequential flow charts modeling but without concurrency.
- **IntermediateActivities**
  - Include concurrent control and data flow.
  - Support modeling similar to traditional Petri.
- **CompleteActivities**
  - Add constructs that enhance the lower level models, such as edge weights and streaming.
- **StructuredActivities**
  - Support modeling of traditional structured programming constructs, such as loops and conditionals.
- **CompleteStructuredActivities**
  - Add support for data flow output pins of conditionals and loops.
- **ExtraStructuredActivities**
  - Supports exception handling and invocation of behaviors on sets of values.

Tenet of activity semantics

- **Used a virtual machine style**
  - More intuitive description for user point of view.
  - Petri-nets like semantics, i.e.:
    - ✓ Based on routing of tokens (i.e. control and data values) through a graph of nodes connected by edges.
    - ✓ Nodes and edges semantics defines rules for token movements.
- **Actions are executed because previous actions finish executing when:**
  - Input objects and data become available.
  - Or events occur external to the flow.

Activity diagrams outlines

- **Oriented flow graph**
  - Consists of nodes connected by edges.
    - ✓ Tokens flow along edges.
    - ✓ Tokens are operated by nodes.
- **Node types**
  - Action nodes
    - ✓ Transform input control/data values into output control/data values for other actions.
  - Control nodes
    - ✓ Route control/data values through the graph.
      - → e.g. Fork, Join, …
  - Object nodes
    - ✓ Store temporarily object/data values.
- **Edge types**
  - Control flow edges
    - ✓ Synchronize the target action starting with the completion of the source action.
    - ✓ Support only control values.
  - Object flow edges
    - ✓ Support for data value passing between actions.
    - ✓ Support only data values.

(FundamentalActivities) – Activity semantics

- **The semantics of activities is based on token flow.**
  - Flow = execution of one node affects and is affected by the execution of other nodes, and such dependencies are represented by edges in the activity diagram.

- **Token**
  - May contain object, data, or locus of control.
  - Is present in the activity diagram at a particular node.
  - Distinct from any other, even if it contains the same value as another token.

- **Node execution**
  - Start
    - ✓ When specified conditions on the input tokens of a node are satisfied.
      - → Conditions depend on the kind of node.
    - ✓ When a node begins execution, tokens are accepted from some or all of its input edges and a token is placed on the node.
  - Completion
    - ✓ A token is removed from the node and tokens are offered to some or all of its output edges.
  - Order
    - ✓ Nodes linked by edges execute sequentially.
    - ✓ Two actions not directly or indirectly ordered by flow relationships execute concurrently.

(FundamentalActivities) – Activity semantics (seq.): nodes and edges have token flow rules

- **Rules**
  - Nodes rules
    - ✓ Control when tokens enter or leave nodes.
  - Edges rules
    - ✓ Define when a token may be taken from the source node and moved to the target node.
  - No specification of the order in which rules are applied on the various nodes and edges in an activity.

    ➔ Execution profiles may tighten the rules to enforce various kinds of semantics.

- **Traverse-to-completion semantics**
  - A token traverses an edge when it satisfies the rules for target node, edge, and source node all at once.
  - A source node can only offer tokens to the outgoing edges, rather than force them along the edge, because the tokens may be rejected by the edge or the target node on the other side.

(FundamentalActivities) – Activity semantics (seq.)

- **Activities context and parameters.**
  - Activities may be attached to Classifier.
    - ✓ Context = the Classifier.
    - ➔ Access to the attributes and operations of its context object and any objects linked to the context object, transitively.
  - Activity as method of a behavioral feature.
    - ✓ Context = the classifier owing the behavioral feature.
    - ➔ Access to the parameters of the behavioral feature.
  - Activity inputs are supplied by the invocation action of the calling activity.

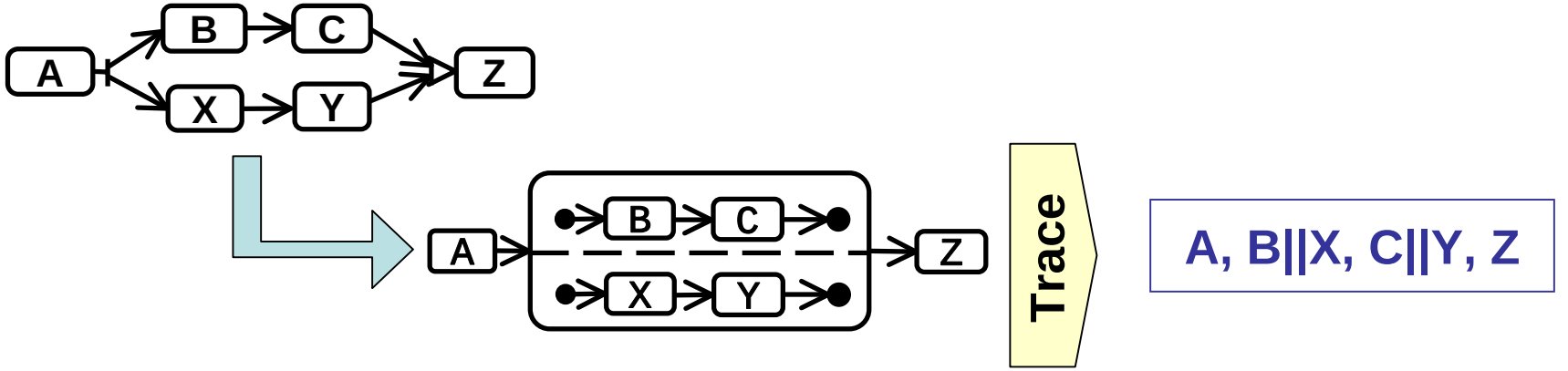
- **Activity invocation time.**
  - If depict a method of behavioral feature ➔ when the behavioral feature is invoked.
  - If depict a classifier ➔ at classifier instantiation time.
  - Or directly by other activities.

(IntermediateActivities) – Activity semantics: Concurrency

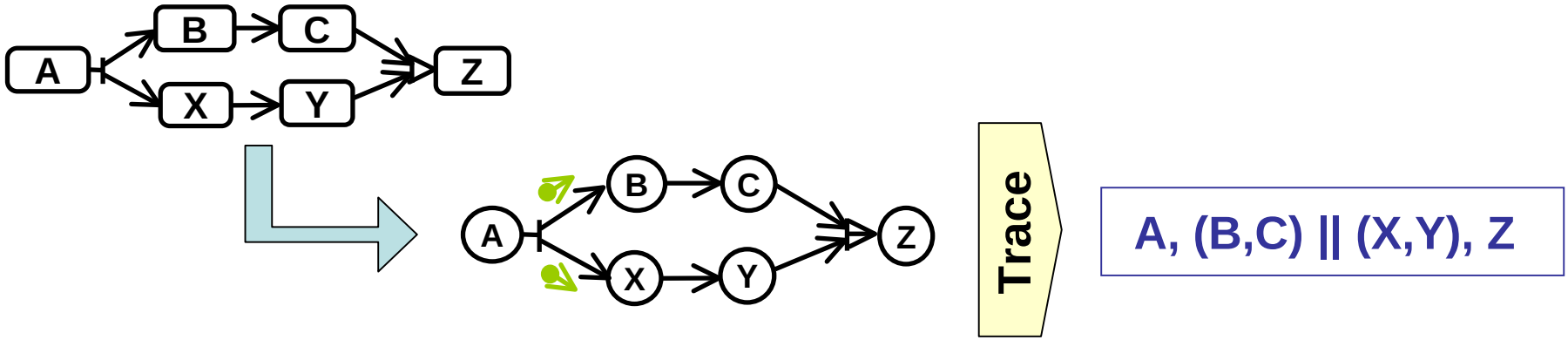
- **Is single execution (isSingleExecution attribute) ?**
  - True => only one execution for each call
    - ✓ Several tokens may flow concurrently in the same activity execution
    - ✓ Possible interactions between token streams
      - → e.g. some token may wait others
      - → e.g. one flow may abort all other because reaching a final sate first.
  - False => one execution per call
    - ✓ No possible problems between tokens, because no interactions in this case!

Parallelism in UML2 activities vs. UML1.x

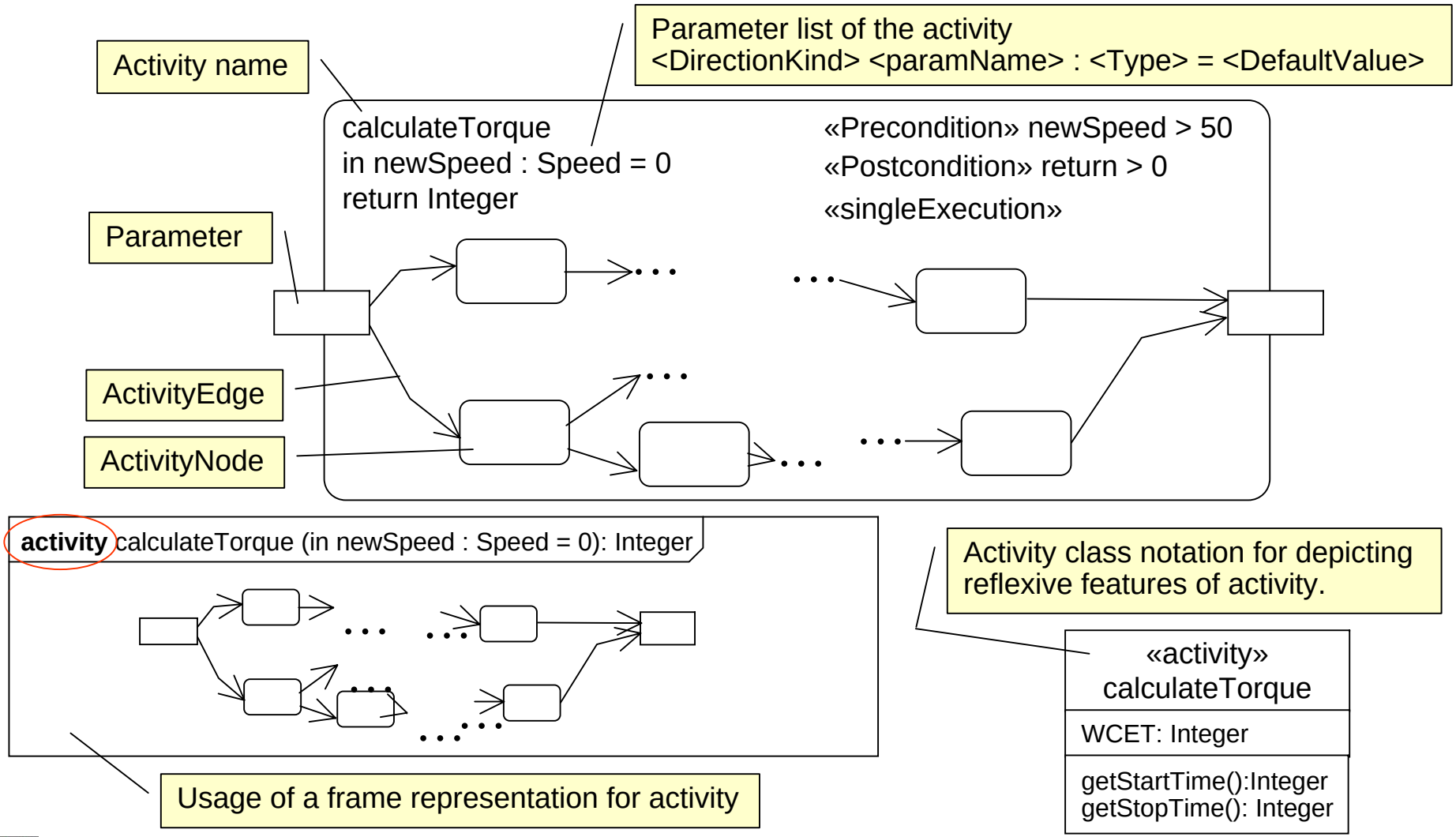
- **Parallelism in UML1 activities**



- **Parallelism in UML2 activities**

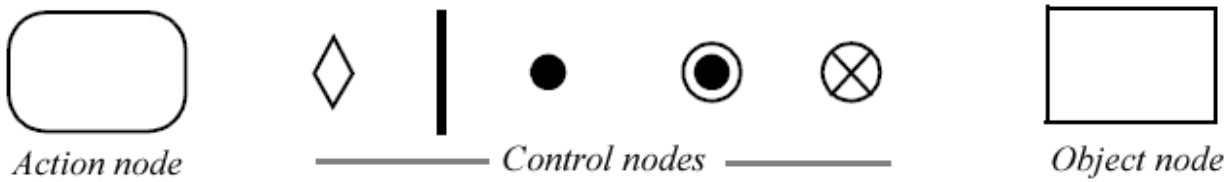


(FundamentalActivities) – Activity notation



ActivityNode [abstract]

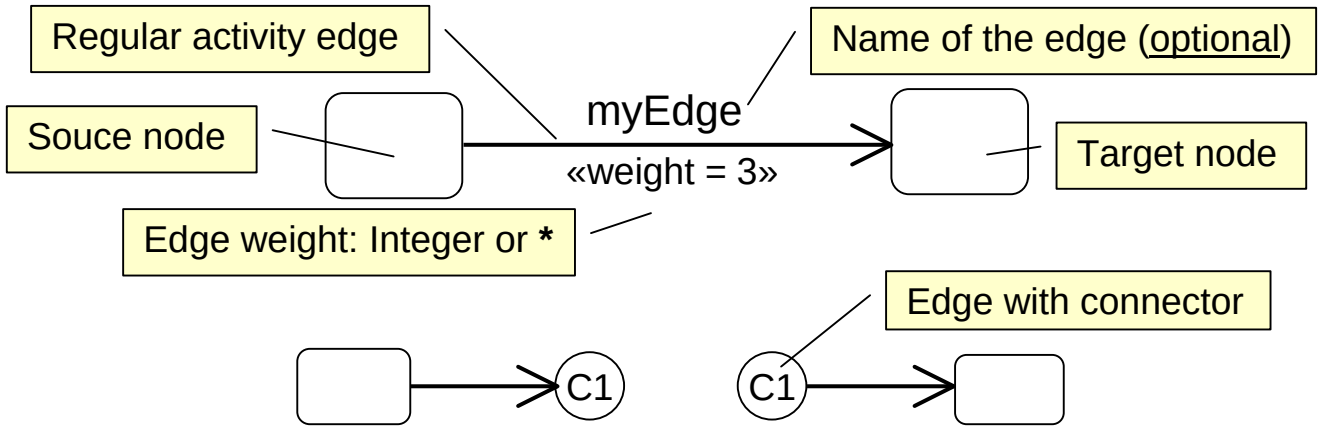
- **Abstract class for the steps of an activity**
  - Executable nodes, control nodes and object nodes.



- **(BasicActivities) Nodes can be:**
  - Replaced in generalization (BasicActivities).
  - Contained in interruptible regions (CompleteActivities).

ActivityEdge [abstract]

- **Abstract class for defining node connections**
  - Directed connections.
    - ✓ i.e. owns a source and a target **ActivityNode**.
  - Support for token flows.
- **Rules for token flow depend on the kind of edge and characteristics of its source & target.**
- **Two kinds of edge: ControlFlow & ObjectFlow**
- **Weight of edges (CompleteActivities)**
  - Number of objects consumed from the source node on each traversal.
  - Default value = 1.
- **Notation**



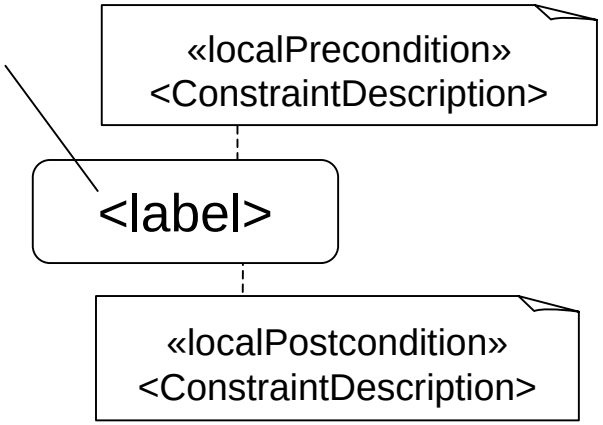
Actions – semantics details

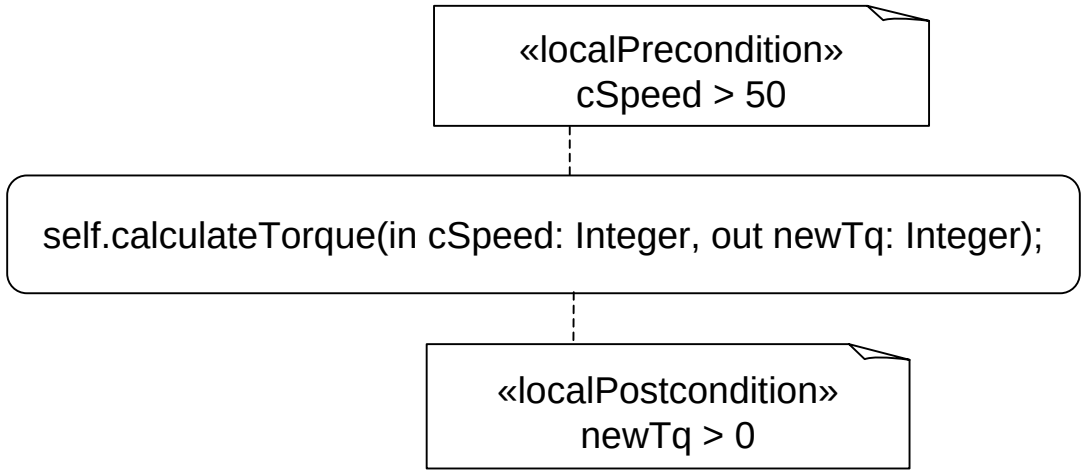
- **Owns incoming and outgoing activity edges that specify control flow and data flow from and to other nodes.**
- **Execution**
  - Start when all of its input conditions are satisfied.
  - Completion may enable the execution of a set of successor nodes and actions that take their inputs from the outputs of the action.
- **[CompleteActivities]**
  - Possible preconditions
    - ✓ Constraints that must be satisfied when execution is started.
  - Possible postconditions
    - ✓ Constraints that must be satisfied when executed is completed.
  - Semantics Variation Point
    - ✓ How local pre- and postconditions are enforced is determined by the implementation.
      - → e.g.
        - » Violation detection at runtime or compile time?
        - » Violation triggers error, warning… ?

Actions – notation

Action name or a sentence that describes the action.

Constraint description (pre/post-condition) or action label may be described using:
- A natural language for informal modelling.
- A tool-dependent action/constraint language for formal and executable modelling.





Superstrucutre presentation agenda

- Part I: Structure
  - Chapter 8. Components
  - Chapter 9. Deployments
- **Part II: Behavior**
  - Chapter 11. Actions
  - Chapter 12. Activities
  - **Chapter 14. Interactions**
  - Chapter 15. State Machines

Interaction overview

- **Interactions focus on the communications between instances, using message passing to express operation invocation and signal sending**

- **Interactions are used**
  - during analysis, to improve individual or group understanding of inter-object behavior
  - during design, to precisely describe inter-process communication
  - during testing, the traces can be compared with those described in the earlier phases.

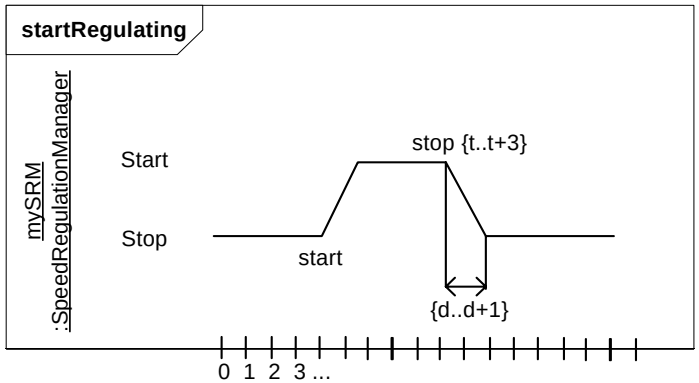
- **The core constructs in an Interaction include:**
  - lifeline, message, interaction occurrence, combined fragment, decomposition

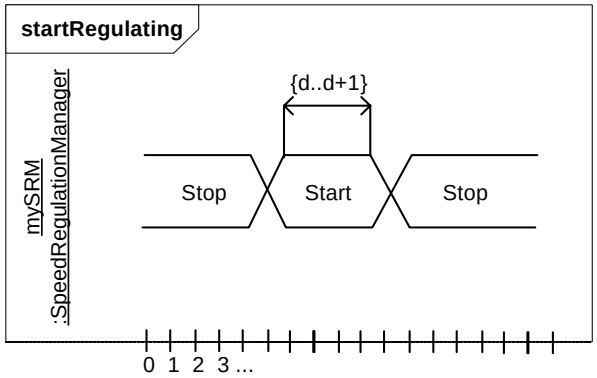
The kinds of Interaction diagrams

- **Sequence Diagrams (~MSC)**
  - most common kind of Interaction Diagram
  - focus on the Message interchange between a number of Lifelines under the point of view of time progression
  - note: May also be represented using a tabular natation
- **Communication Diagrams (previous collaboration diag. of UML1)**
  - focus on the interaction under the structural point of view. The sequencing of messages is given through a sequence numbering scheme.
- **Interaction Overview Diagrams (~HMSC)**
  - define Interactions through a variant of Activity Diagrams in a way that promotes overview of the control flow
- **Timing Diagrams**
  - primary purpose of the diagram is to reason about time

Timing Diagram

- Interactions with focus on time
- Timing vs. Sequence diagrams
  - **Very similar diagrams**
  - **More intuitive specification of state changes over time in timing diagrams**
- Examples:





- Main defaults
  - No implementation in current UML2 tools
  - Very weak specification!

Superstrucutre presentation agenda

- Part I: Structure
  - Chapter 8. Components
  - Chapter 9. Deployments
- **Part II: Behavior**
  - Chapter 11. Actions
  - Chapter 12. Activities
  - Chapter 14. Interactions
  - **Chapter 15. State Machines**

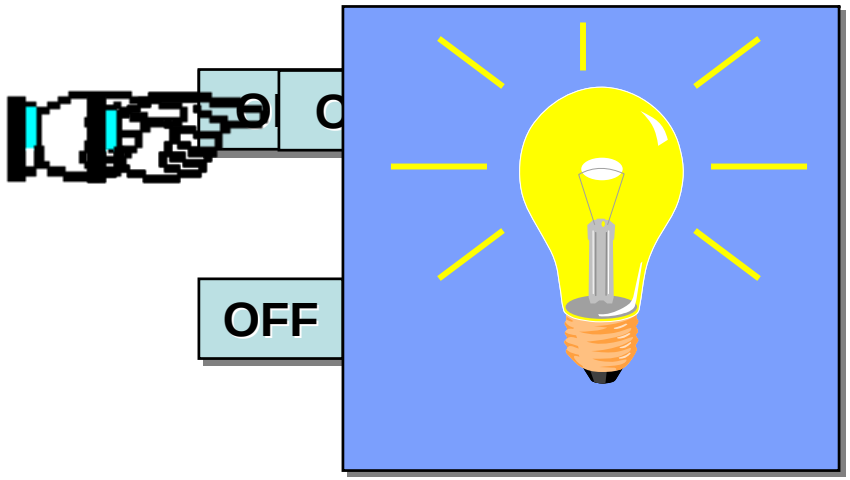
Overview of State Machines package

- **Define the set of concepts required to model discrete behavior through finite state-transition systems**
- **Two kinds of state machine defined**
  - Behavioral state machine
  - Protocol state machine
- **The core constructs in statemachines include:**
  - States, Regions, Transitions, Events
- **Typical applications of statemachines include:**
  - Object lifecycle (i.e. states an “order” object might be in)
  - Event-driven behaviors of embedded controllers
  - UI controllers
  - …

Automata

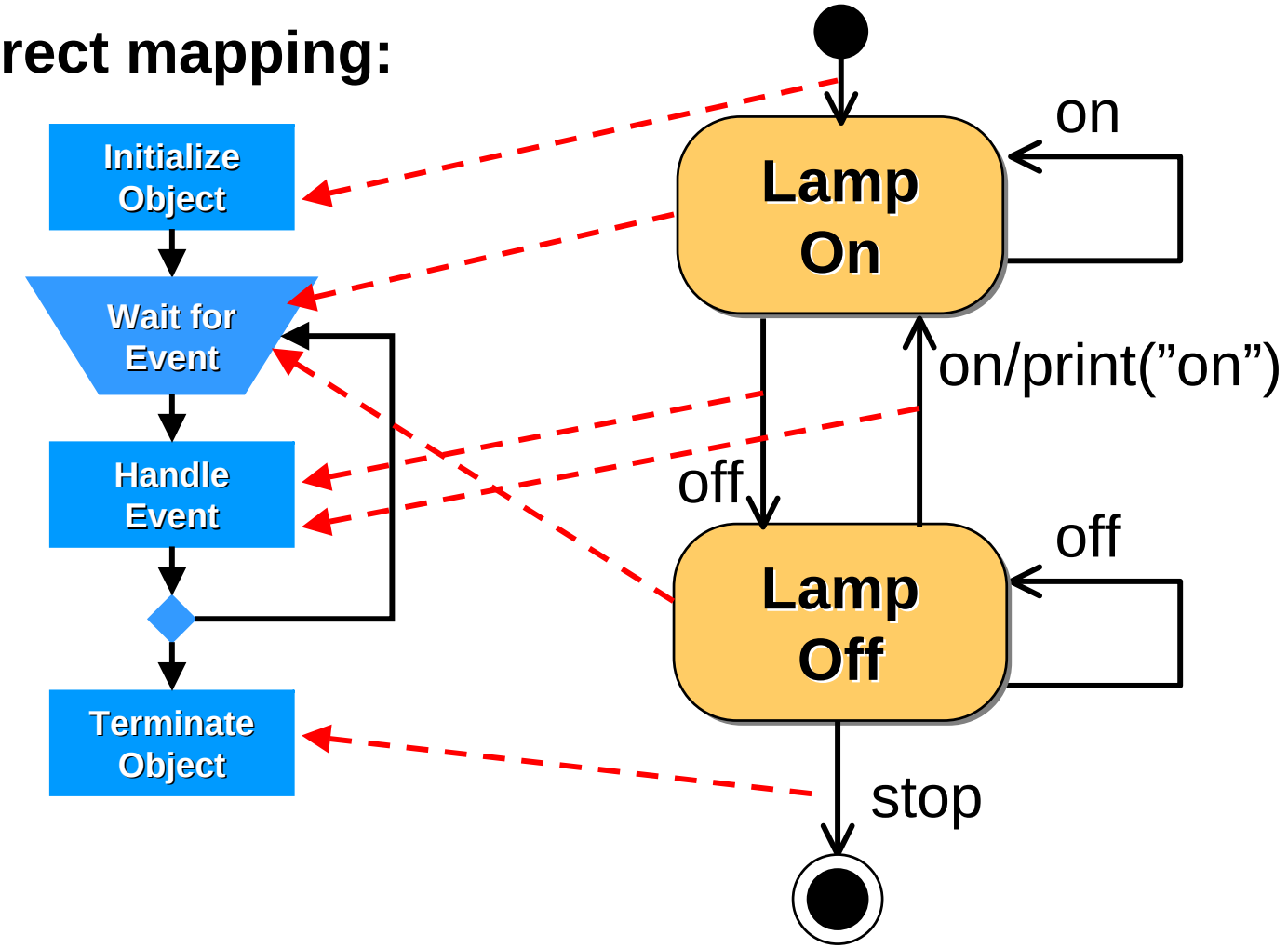
- **A machine whose output behavior is not only a direct consequence of the current input, but of some past history of its inputs**

- **Characterized by an internal state which represents this past experience**



Object Behavior and State Machines

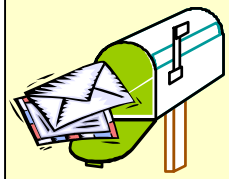
- **Direct mapping:**



State machine semantics

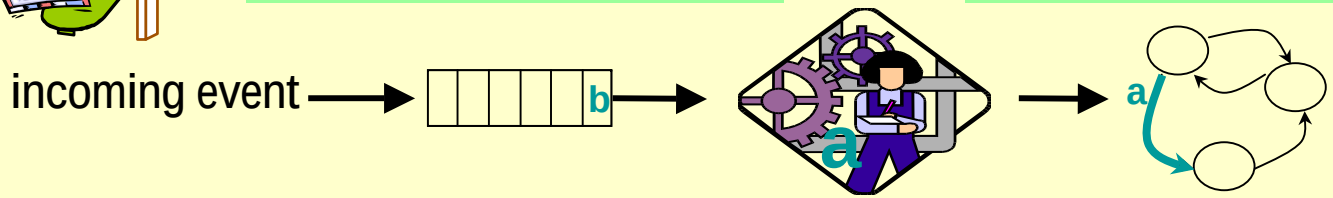
- **UML state machines = hypothetical machine that:**
  - Queues events
  - Dispatches events
  - Processes events

Run-To-Completion:
only one event at a time



Queue
=> Saves incoming events

Processor
=> Consumes selected event



Dispatcher
=> Selects and dequeues an event

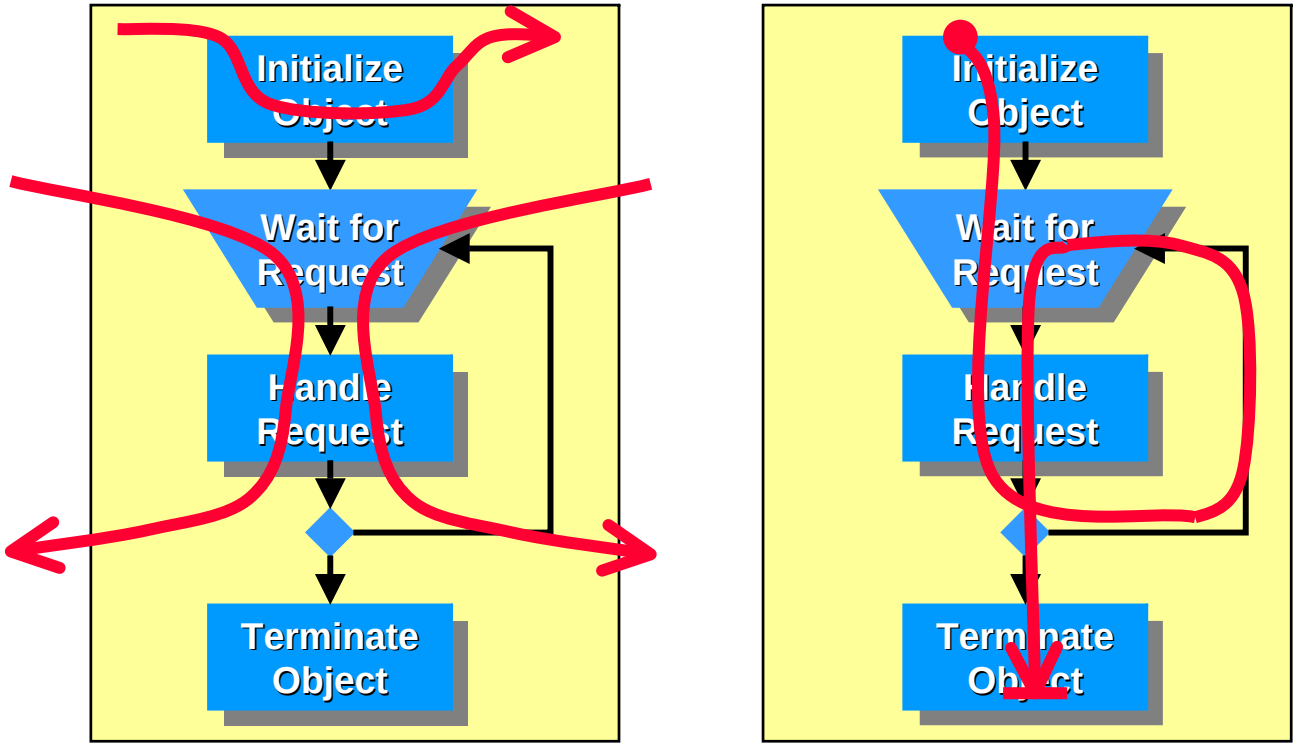
– FIFO dequeuing (commonly used by OO tools)

State machine semantics variation points

- **Event dequeuing policy has to be choosen**
  - Most widespread = FIFO
- **RTC granularity**
  - The only one defined = a single RTC for the whole state machine
  - But, it is possible to have several…
    - ✓ e.g.: one for each orthogonal region → but then « Just Do It !»
- **State machine inheritance issues: to be clarified**
  - There are no default or usual rules
    - ✓ only incomplete proposals (see later)

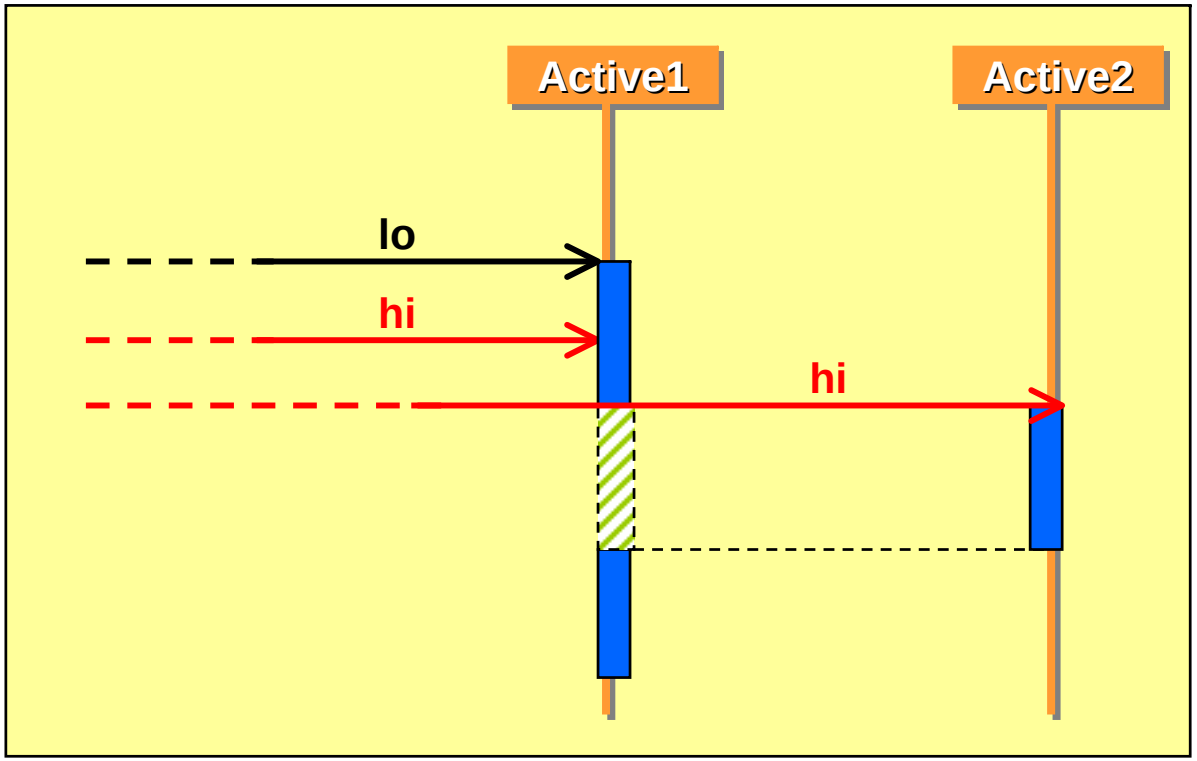
Object and Threads

- **Passive objects: depend on external power (i.e. execution resource)**
- **Active objects: self-powered (have their own execution resource)**

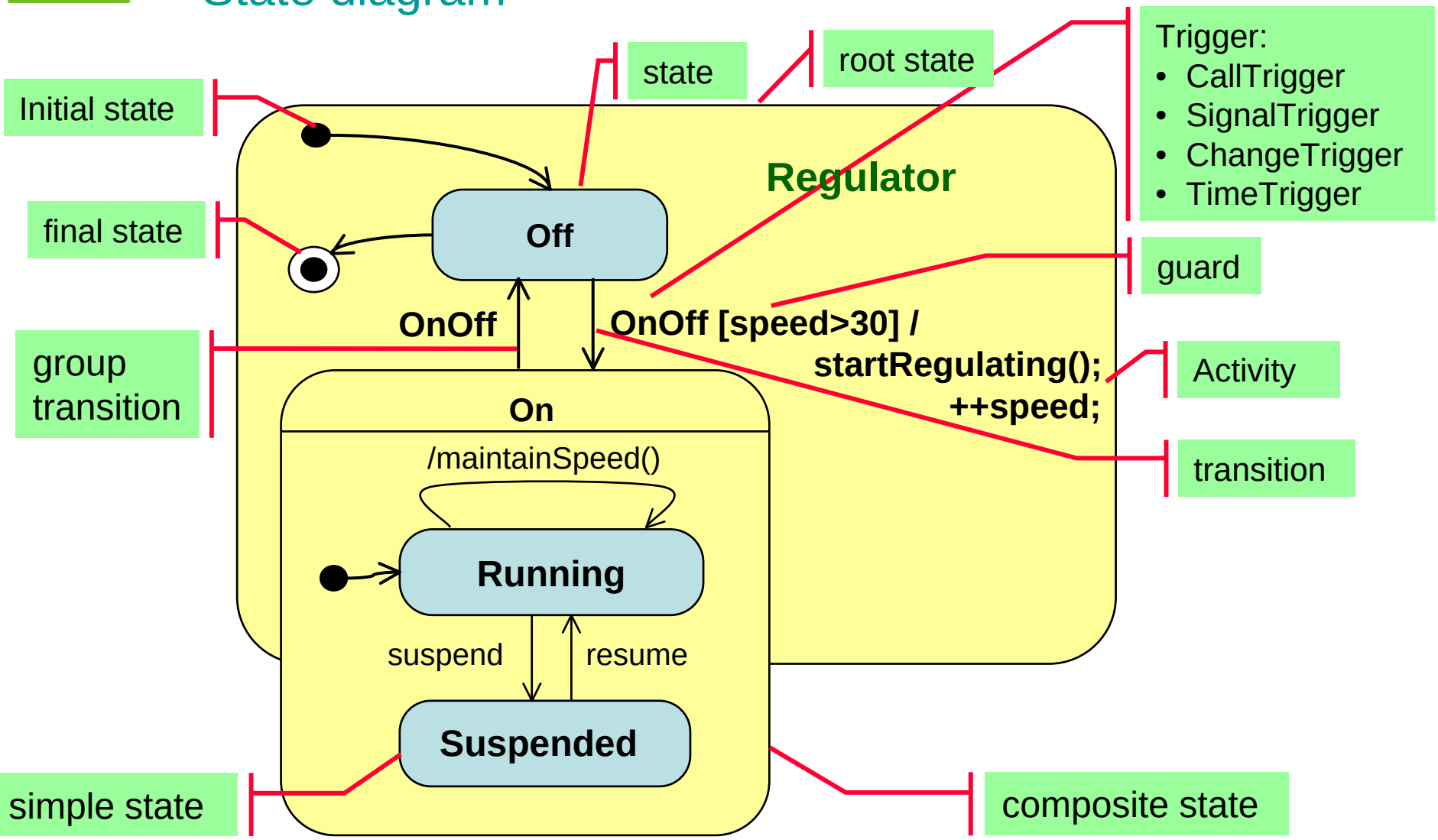


The Run-to-Completion Model

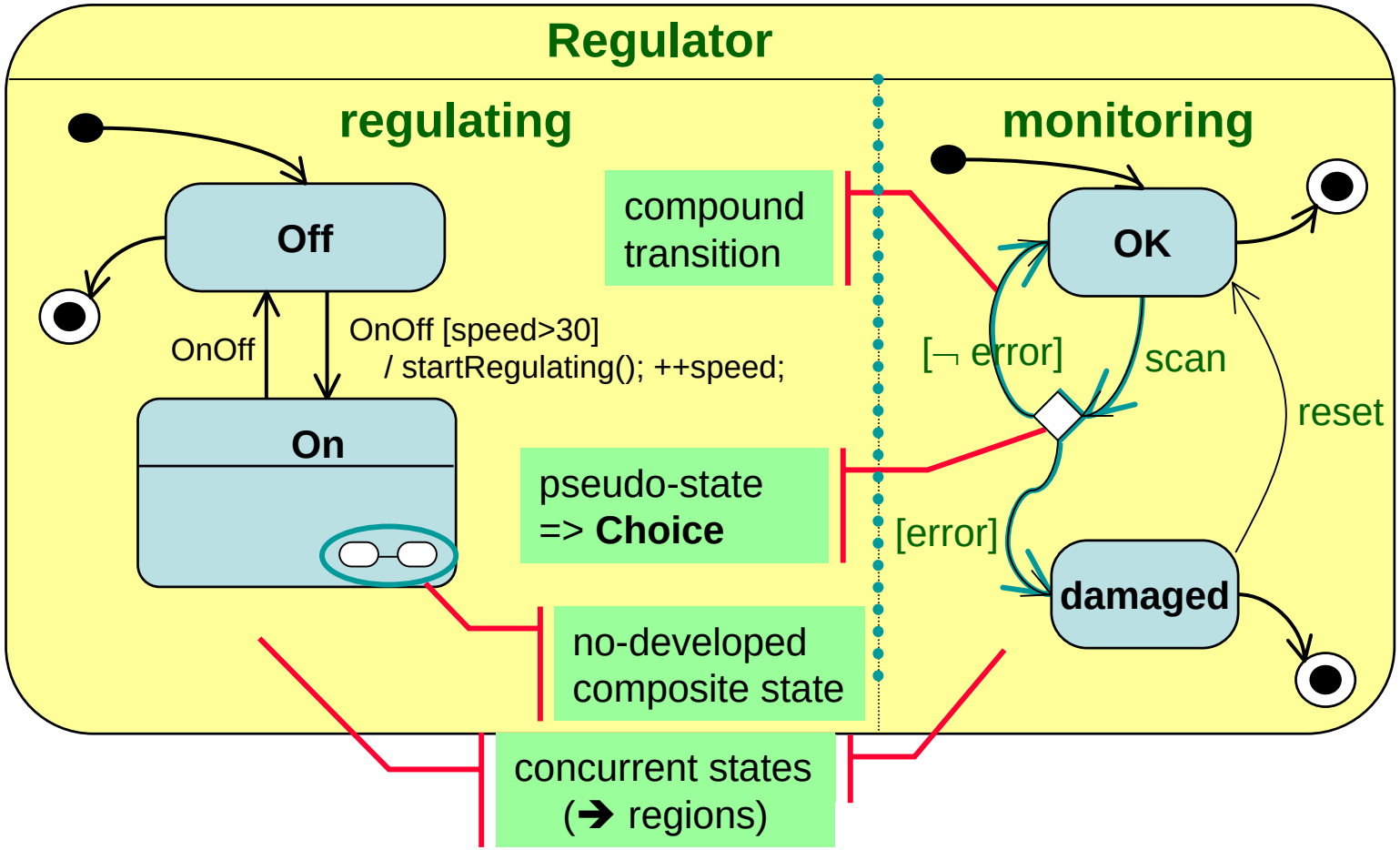
- **A high priority event for (another) active object will preempt an active object that is handling a low-priority event**



State diagram



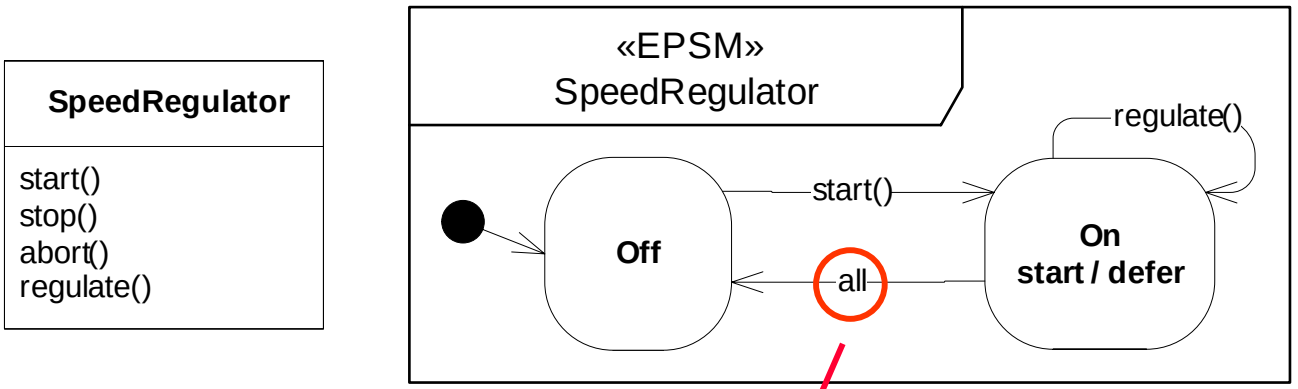
State diagram (seq.)



The AnyReceiveEvent

- **Any receipt of event trigger the transition firing expect those triggering explicitly transitions with same source vertex.**

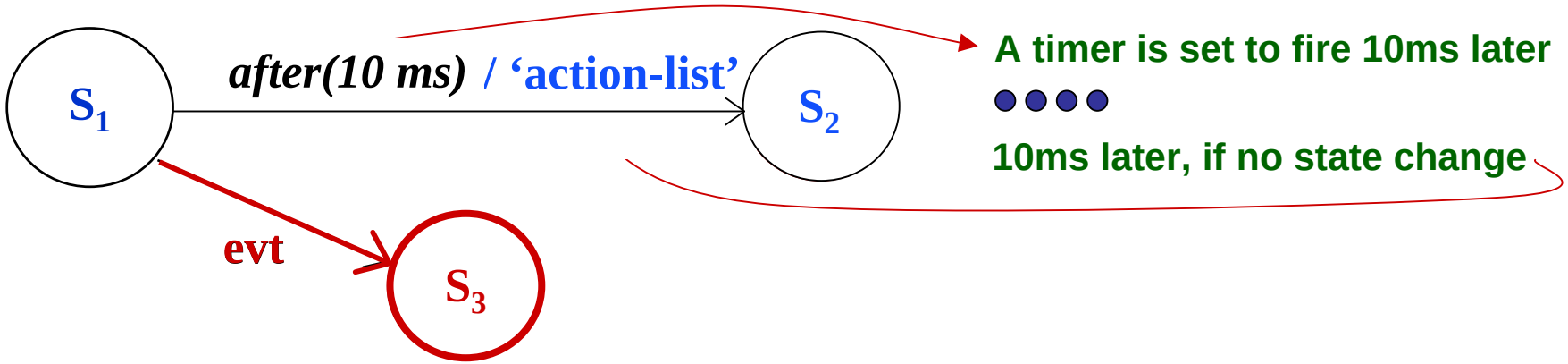
- **Example:**



"all" means for AnyReceivEvent
➔ In this case, either *stop* or *abort* event receipt may trigger the transition

Timing specification within state machine

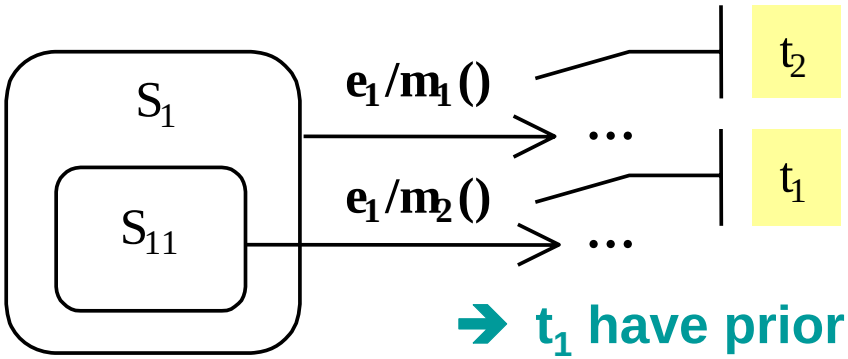
- **Timer setting on transitions of state machines**
  - TimeEvent
    - ✓ **after** ⇒ **relative moment in time**
    - ✓ **when** ⇒ **absolute moment in time**



- **Drawbacks**
  - Specification is performed through implementation mechanisms
  - TimeEvent handled as other events ⇒ determinism issues !

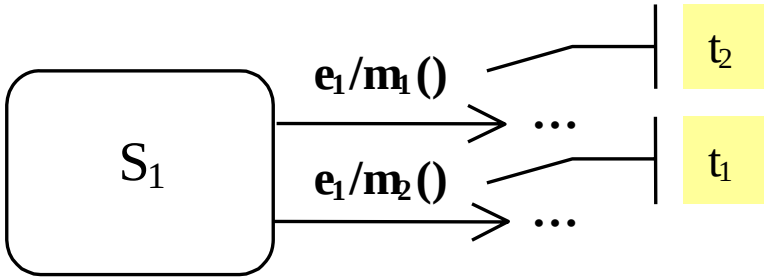
Timing specification within state machine (seq.)

- **Implicit priority rules between transitions …**



➔ t_1 have priority over t_2 because S_{11} is a substate of S_1

- **… but possible non determinism models**



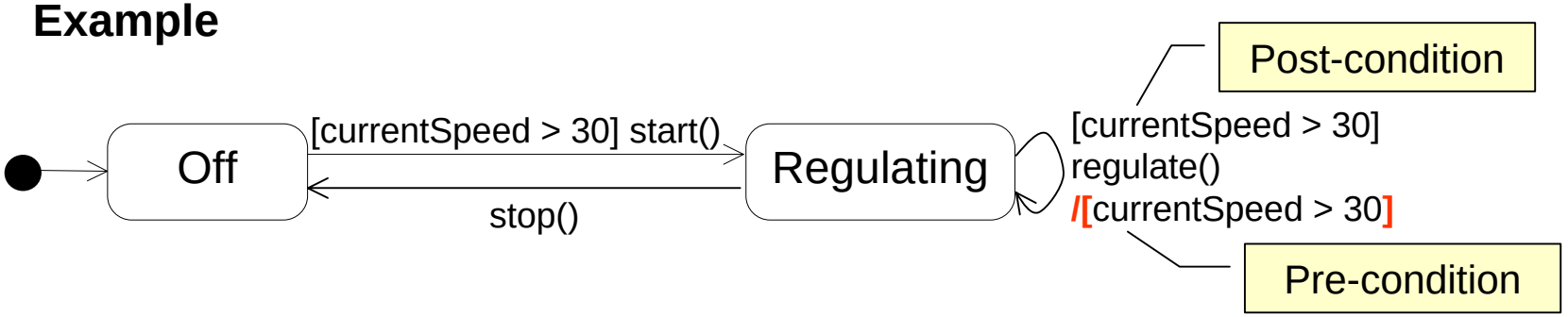
➔ non determinism situation !

Details of the Protocol Statemachines

- **Specialization of Statemachine concept**
- **Used to express usage protocols**
  - life cycle of objects, legal usage of operation…
- **May be linked to interface/port to specify behavioral contract**
- **Protocol transition**
  - Specifies legal transitions for an operation
    - ✓ "It specifies that the referred operation can be called for an instance in the origin state under the initial condition (guard), and that at the end of the transition, the destination state will be reached under the final condition (post)."
  - May have pre- (i.e. guard) and post conditions
  - No "effect" action
    - ✓ implicit operation linked with the call trigger

Details of the Protocol Statemachines (seq.)

- **Unexpected event reception**
  - Semantic Variation Point: "the event can be ignored, rejected or deferred, an exception can be raised, or the application can stop on an error. (i.e. pre-condition violation)"
- **Equivalences to pre and post conditions of operations**
- **Unreferred Operations**
  - May be executed whatever the state of the object
  - Do not change statemachine state
- **Possible to use other triggers on protocol transition**
  - No action effects specified, but post-condition
- **Example**

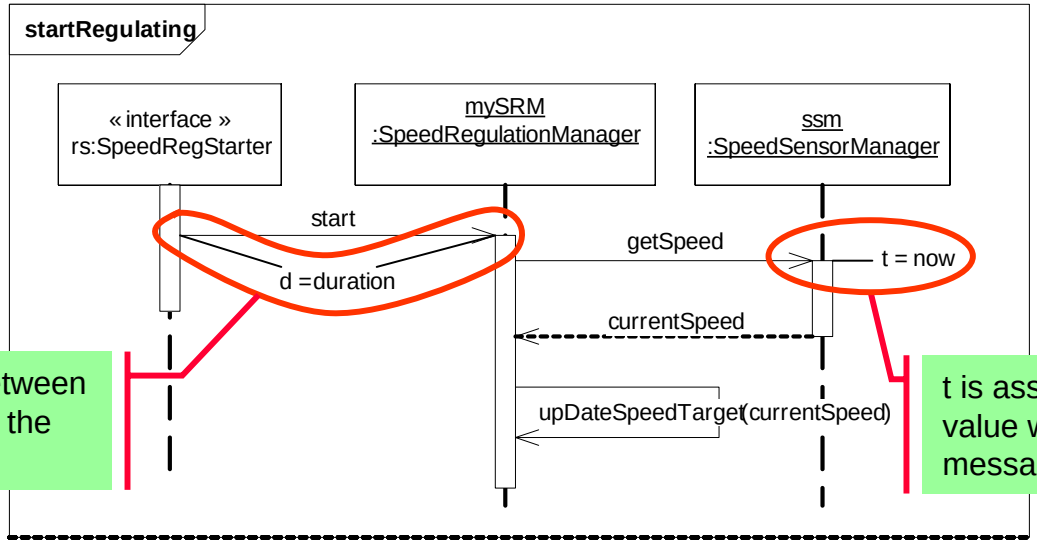


Time domain definition

- **TimeExpression**
  - Denote a time value
  - Notation
    - ✓ Implementation specific textual format
    - ✓ Example: RTtimeVaulue in SPT (see later)
- **Duration**
  - Denote the difference between two points in time
  - Notation: implementation specific textual format
- **Interval**
  - Range between two value specifications
  - Notation: <interval> ::= <min-value> '..' <max-value>
  - **TimeInterval**
    - ✓ Denote a range between two TimeExpression element
    - ✓ Notation: interval notation with values typed TimeExpression
  - **DurationInterval**
    - ✓ Denote a range between two Duration elements
    - ✓ Notation: interval notation with values typed Duration

Specific actions related time characteristics

- **DurationObservationAction**
  - Action that measures an identified duration in the context in which it is executing and writes the obtained value to the given structural feature
  - Notation: <durationobservation> ::= <write-once-attribute> '= duration'
- **TimeObservationAction**
  - Action that obtains the current value of time in the context in which it is executing and writes this value to the given structural feature
  - Notation: <timeobservation> ::= <write-once-attribute> '= now'
- **Examples**

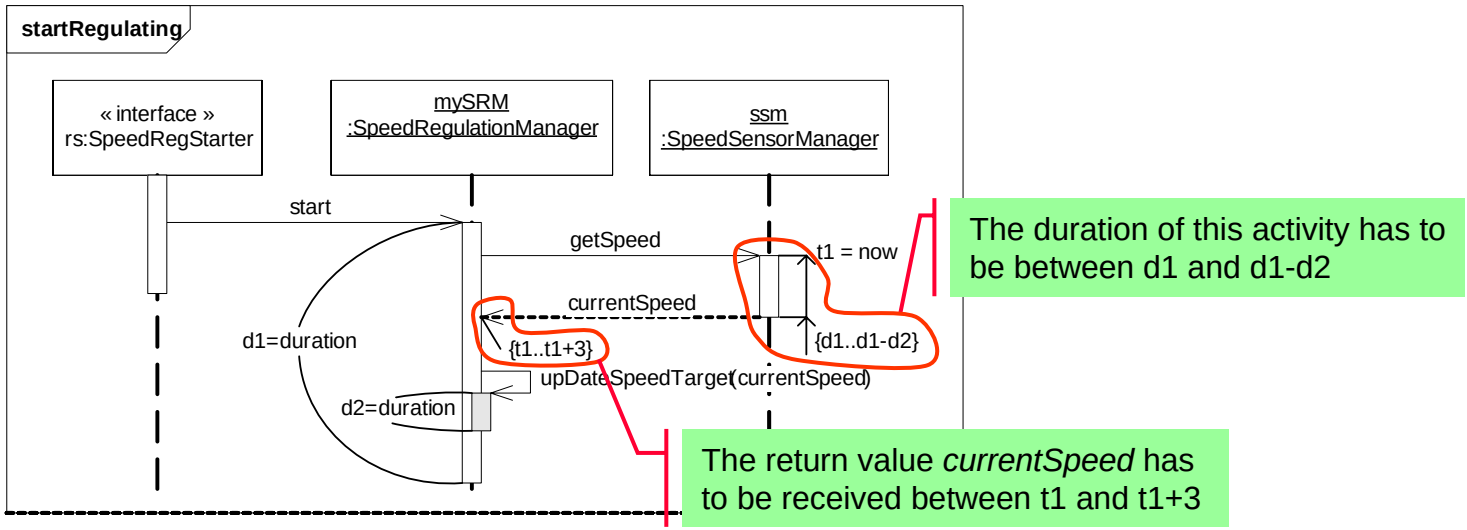


d = difference of time between reception and posting of the *start* message.

t is assigned with the current time value when the getSpeed message is received.

Specific constraints related time characteristics

- **Constraint (from Kernel)**
  - Additional semantic information attached to an element
  - Condition or restriction expressed in natural language text (or in a machine readable language) put under brackets
    - ✓ e.g. OCL is often used for constraint description
- **TimeConstraint**
  - Constraint referring to one TimeInterval
- **DurationConstraint**
  - Constraint referring to one DurationInterval

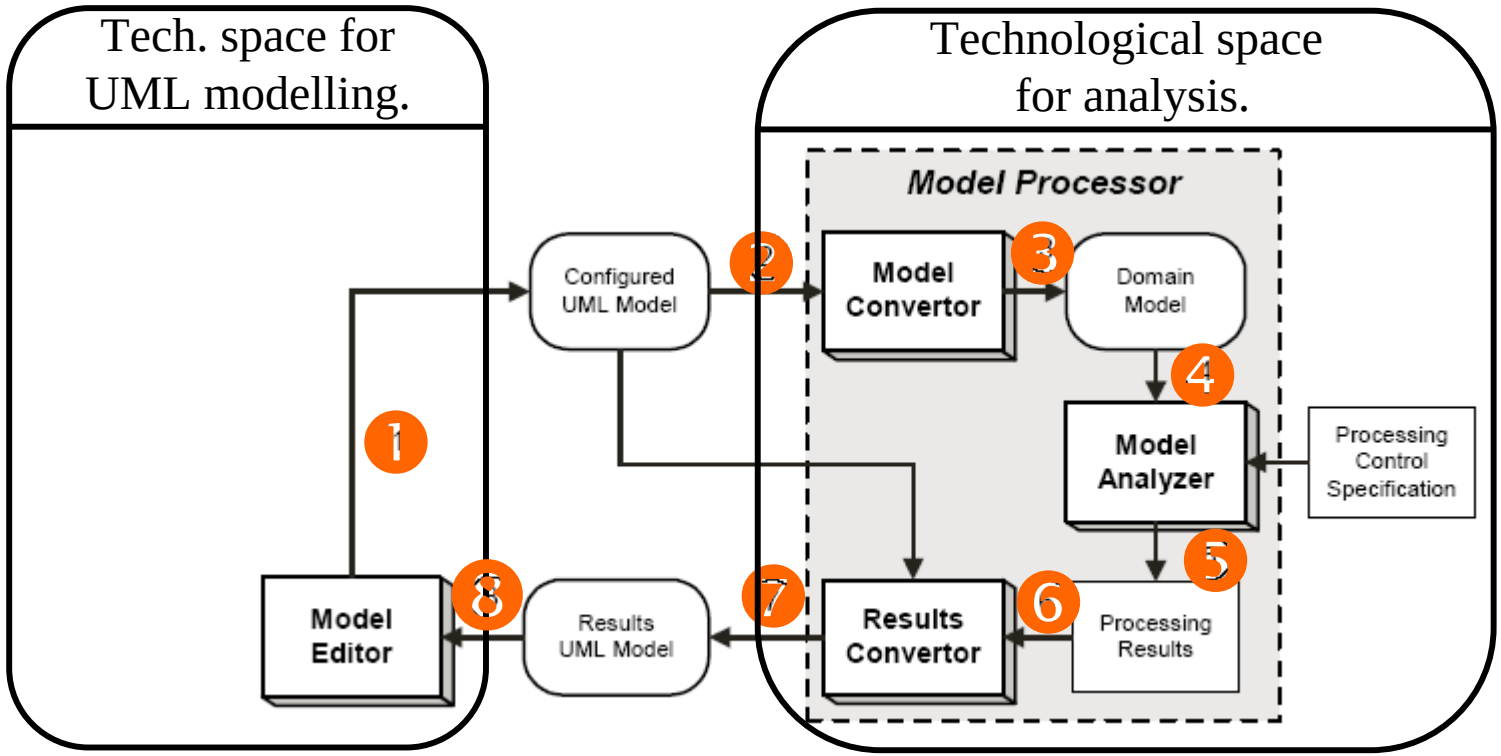


Agenda

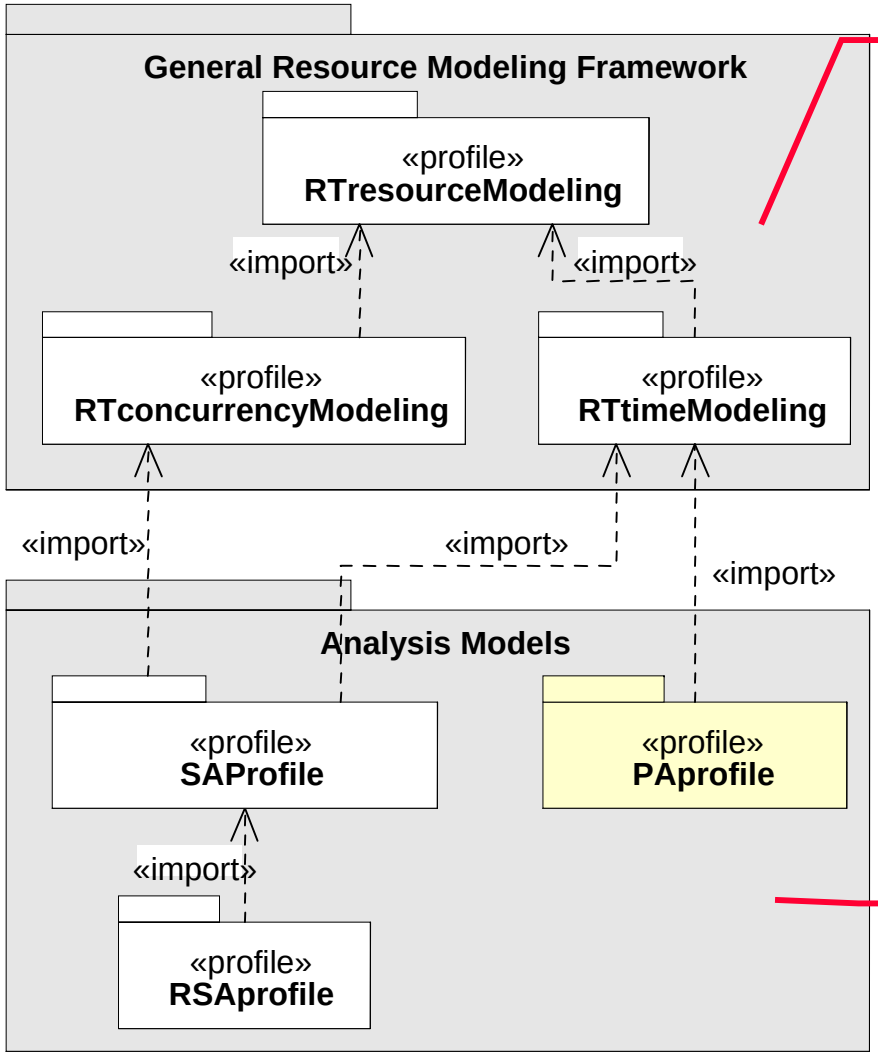
- Introduction on MDE and UML2
- Native concepts for RT in UML2
- **The UML profile for SPT specification**
- **Conclusions and perspectives**

Motivations

- **Required modelling information for analysis**
  - Add specific annotations to models for analysing
    - ✓ E.g. The UML profile for SPT
- **Model processing**

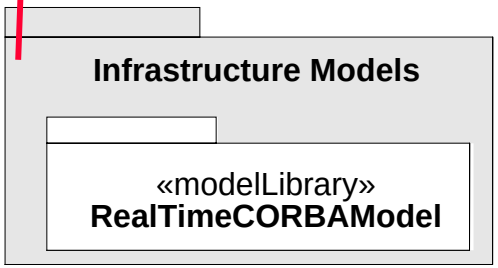


Structure of the SPT



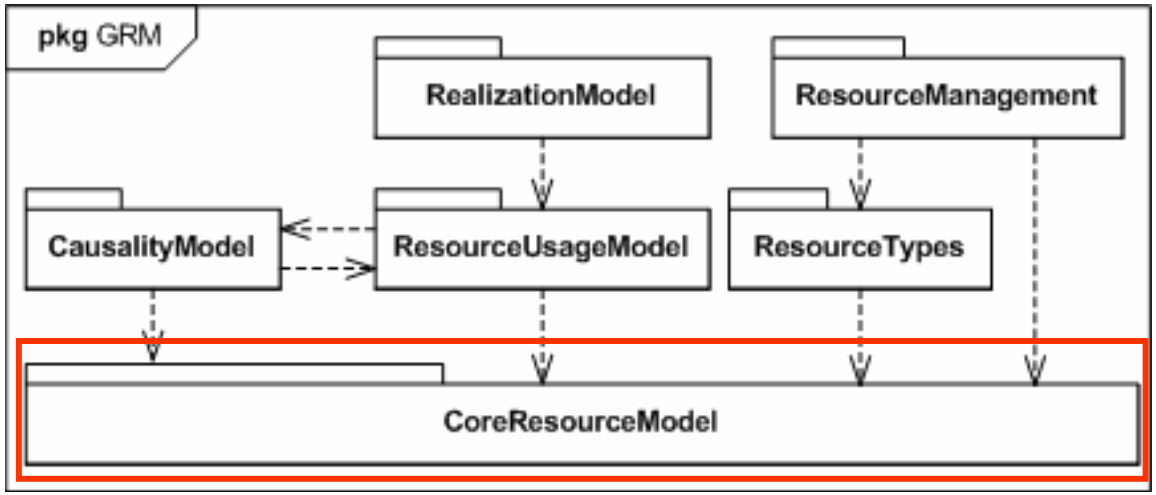
Generic framework for any kind of analysis: General Resource Modeling, General Concurrency Modeling and General Time Modeling

Support for modeling a RT-CORBA platform for schedulability analysis



Support for schedulability and performance analysis

General Resource Modeling



- **Main issue to solve by analysis methods:**
  - Is offer enough for the demand ?

➔**Principle of GRM relies on the client-server model**

- **Basics of the core resource model**
  - Required and offered QoS on clients and servers
  - QoS contract between clients and servers

GRM – Causality & resource usage models

- **Causality model**
  - Basics for all dynamic descriptions of SPT
  - Details
    - ✓ Based on 2 kinds of event occurrences
      - → Stimulus generated by caller objects
      - → Stimulus received by called objects
        - » Trigger a set of actions called **scenario**
        - » Action executions may then generate stimulus…
- **Resource usage model**
  - Specifies how clients use resources
  - Two types
    - ✓ Static
      - → Structural relationships between clients and resources
    - ✓ Dynamic
      - → Scenario (i.e. set of actions) following the predecessor-successor model with possibly multiple and concurrent pred./succ.

GRM – Resource types

- **3 taxonomies for resource types depending on:**
  - Their goal
    - ✓ Processor resources (physical or virtual): *can save & execute codes*
    - ✓ Communication resources: *ensure information sharing*
    - ✓ Device resources: *others*
  - Their nature
    - ✓ Active resources: *may manage stimuli on their own*
    - ✓ Passive resources: *others*
  - Their associated protection mechanism
    - ✓ Protected resource instances offer at least one exclusive service specifying one access policy.
    - ✓ Non protected resources: Others

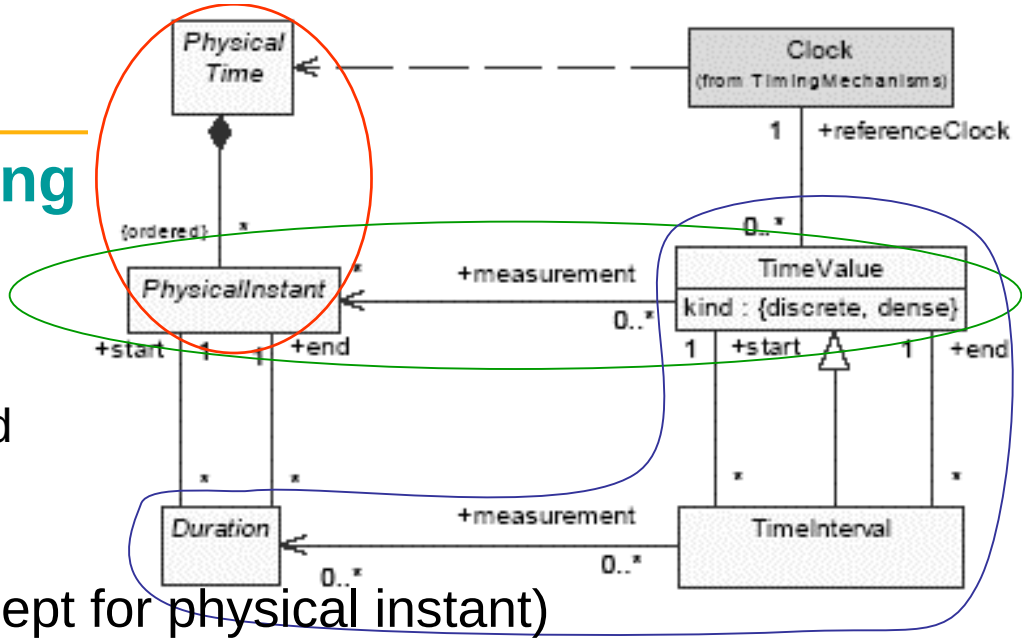
GRM – Resource management

- **Resource broker**
  - Ensure allocation and release of a set of resource instances depending of an access control policy
- **Resource manager**
  - Manage life cycle of resource instances
    - ✓ Creation, delete…
  - Based on a resource control policy
- **In OS, resource broker and manager are a same entity**

GRM – Realization model

- **Related to the deployment issues**
- **Based on two layering models**
  - Refinement-type layering
    - ✓ Sequel of models obtained by refinements
      - → Models are more and more detailed
    - ✓ e.g. a UML model refined in a C++ model
  - Realization layering
    - ✓ An abstract layer is realized by a more concrete one
    - ✓ e.g. application layer relying on an OS layer

General Time Modeling

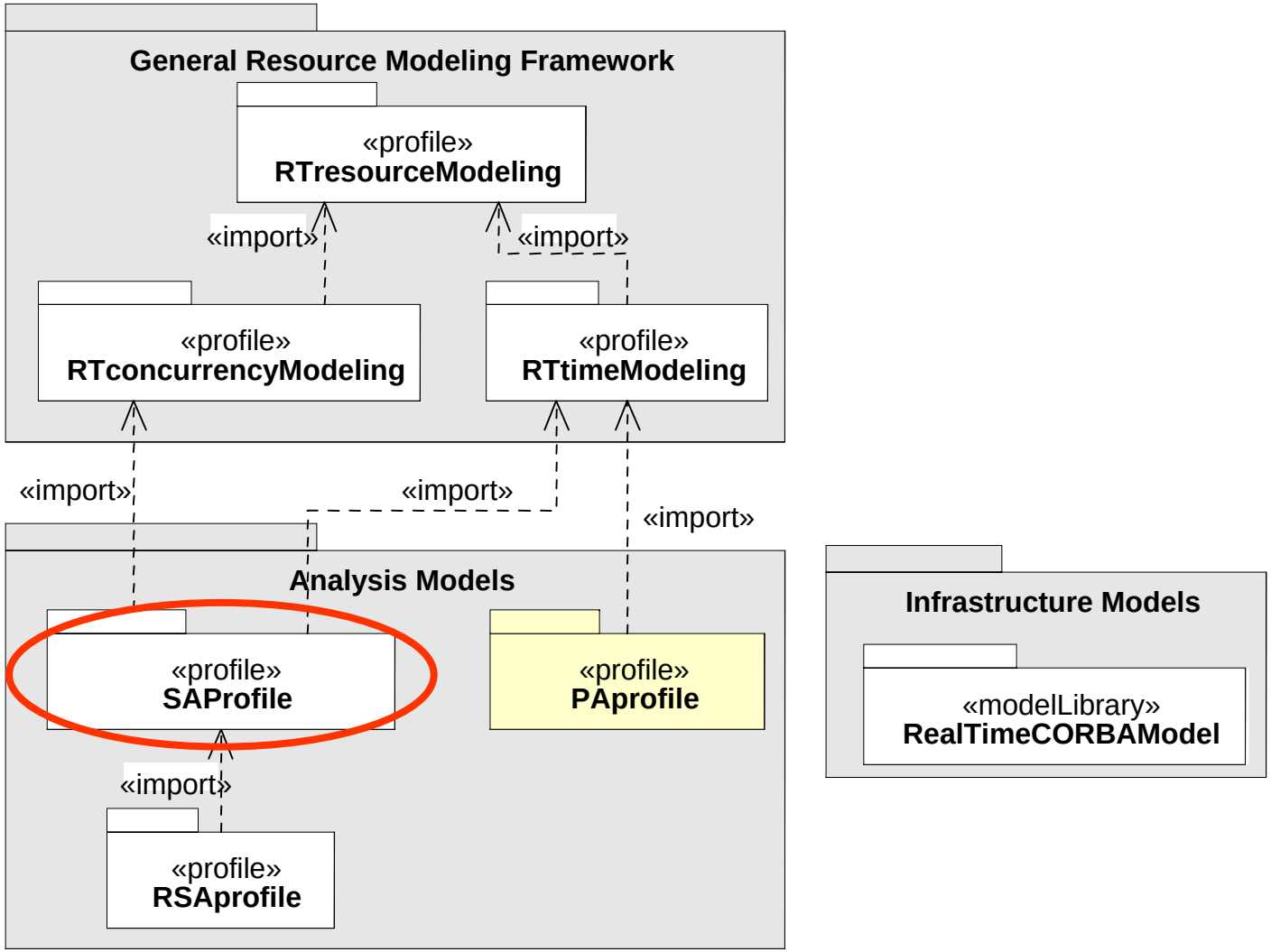


- **PhysicalTime**
  - Set of physical instants
    - ✓ Continuous and unbound
    - ✓ Ordered and dense
- **Concret concepts**
  - TimeValue (concrete concept for physical instant)
    - ✓ Discret: represented as integers
    - ✓ Dense: represented as floats
  - TimeInterval (concrete concept for duration)
    - ✓ Specialize TimeValue ➔ discret or dense
    - ✓ Owns both start and end time values
- **Two mechanisms for time measurement**
  - Clock
    - ✓ Generate periodically a specific stimulus
  - Timer
    - ✓ Generate a stimulus at a given time instant
    - ✓ Relative or absolute times
    - ✓ May be periodic

Concurrency model

- **ConccurentUnit (CU)**
  - Entity that may execute concurrently to others
- **CU instantiation involves main scenario execution**
  - It executes until instance delete
  - It may call stimulus reception action to accept them
  - Accepted stimulus may trigger appropriate service execution
- **During execution of services:**
  - Either the main scenario is blocked until service execution end
    - ✓ “Run-to-Completion” assumption
  - Or it may continue and for example accept other stimuli
    - ✓ Possible intra-concurrency
- **CU may have one or plus queues for saving incoming stimuli**

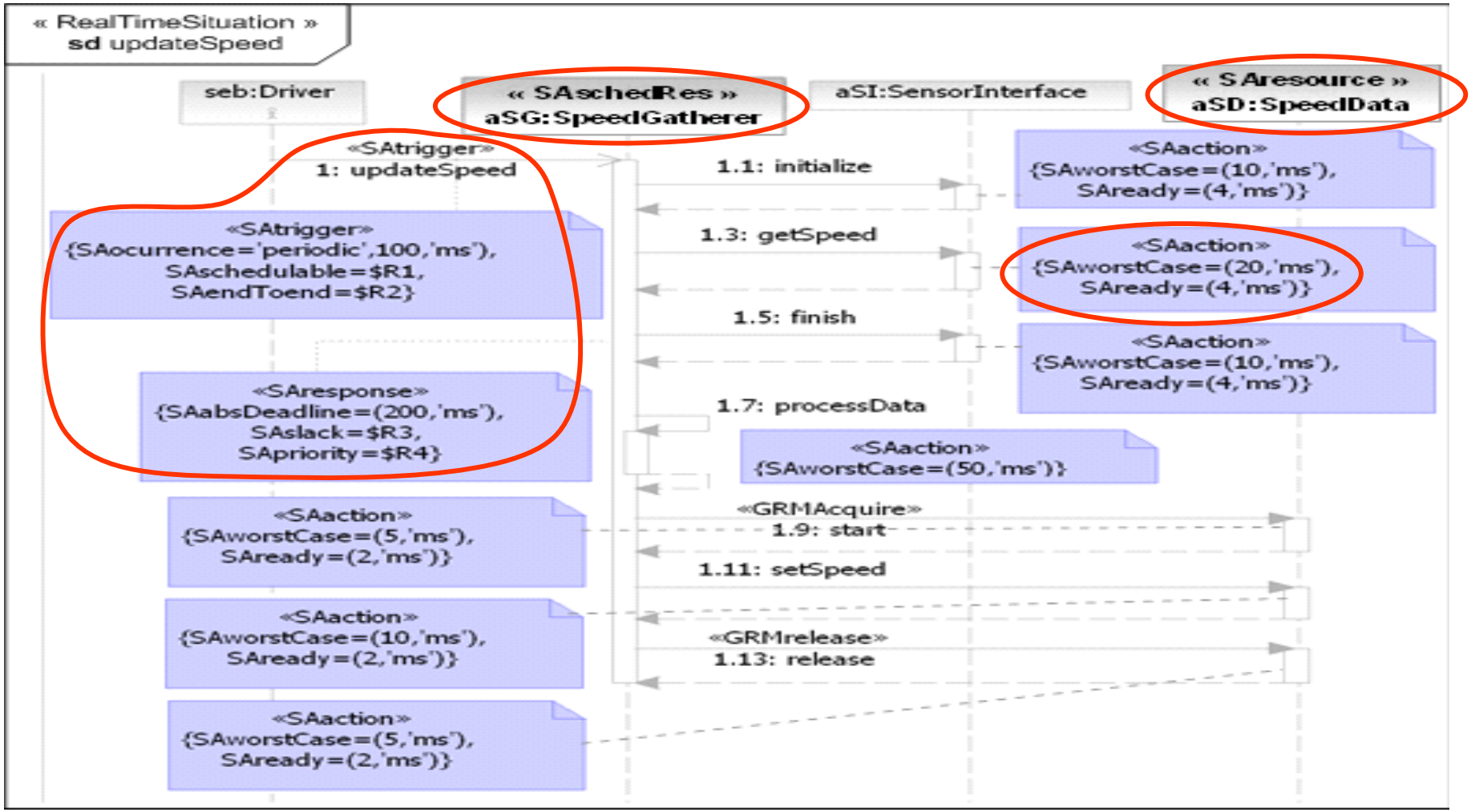
Structure of the SPT



Main concepts of the SA sub-profile

- **SchedulableRessource**
  - Support for task
  - Set of SAaction
    - ✓ Priority, WCET, Delay time…
- **Scheduler**
  - Specify a scheduling policy
    - ✓ Rate monotonic, fixed priority…
- **ExecutionEngine**
  - Allocated to the execution of a set of schedulable res.
  - Owns features
    - ✓ Priority range, context switch time…

Usage example in sequence diagrams



Agenda

- Introduction on MDE and UML2
- Native concepts for RT in UML2
- The UML profile for SPT specification
- **Conclusions and perspectives**

Pro and con of the UML Profile for SPT

- **The SPT has carried out key challenges for RT modeling:**
  - A generic framework for modeling Resources and their QoS was proposed (general notions were adopted by the QoS & FT profile).
  - A powerful mean to model metric Time and general Concurrence.
  - Two temporal analysis modeling frameworks biased to RMA-based (Schedulability) & Queuing and Petri Nets (Performance) techniques.

- **But, some lacks were reported since its adoption:**
  - Incoherencies between GRM, Schedulability, and Performance modeling elements show a need for a better top-level architecture.
  - Certain drawbacks exist to model more complex systems (e.g. distributed systems, hierarchical schedulers, etc.).
  - Specific semantics must be better defined. For instance, different kinds of deployment, properties of communication engines, etc.
  - SPT does not support state machine-based modeling.

➔ A new version: the UML profile for MARTE
(**M**odeling and **A**nalysis of **R**eal-**T**ime and **E**mbedded systems)

A Broader Scope of UML for MARTE

- **Furthermore, upcoming MARTE profile must support:**
  - Integrated modeling of both software and hardware aspects.
  - Modeling of platform, platform-independent, and their allocation viewpoints in a MDA approach.
  - Specification of not only RT constraints but also embedded QoS characteristics as power consumption and memory size.
  - Modeling of embedded, reactive, control/command, and intensive data flow computational systems.
  - Component-based architectures modeling and analysis.
  - Capability to Asynchronous/Causal, Synchronous/Clocked, and Real/Continuous time modeling.
  - Compliance with UML 2 and the QoS & FT profile.

Schedule for the MARTE standard

- **Initial submission**
  - Dec. 2005
- **Final version**
  - End of 2006

www.promarte.org

Questions ?