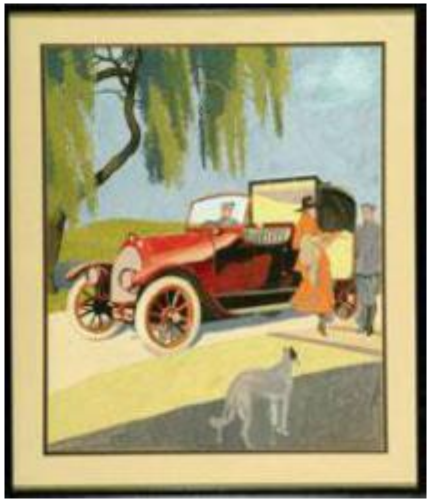
《Modeling and Analysis of system》
—review

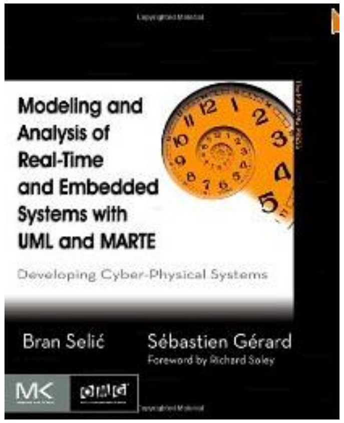
chenbo2008@ustc.edu.cn



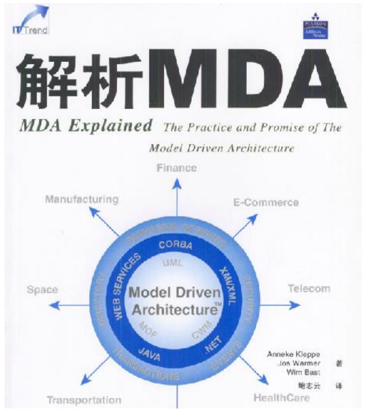
Course Slides

章节	内容
第一部分 建模基础	信息物理融合系统、建模、模型驱动框架
	安全攸关系统/实时与嵌入式系统
第二部分 需求建模	软件系统的需求建模（消息、用例等）
	软件系统的需求建模（静态结构）
	软件系统的需求建模（动态行为）
第三部分 系统设计	用于系统开发的面向对象快速开发过程
第四部分 高级建模概念	实时系统建模与分析语言（MARTE）
	实时系统与编程语言（Ada/JAVA等）

References


Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE
Developing Cyber-Physical Systems
Bran Selić
Sébastien Gérard
Foreword by Richard Soley

嵌入式系统导论
CPS方法
Introduction to Embedded Systems
A Cyber-Physical Systems Approach



Grading：

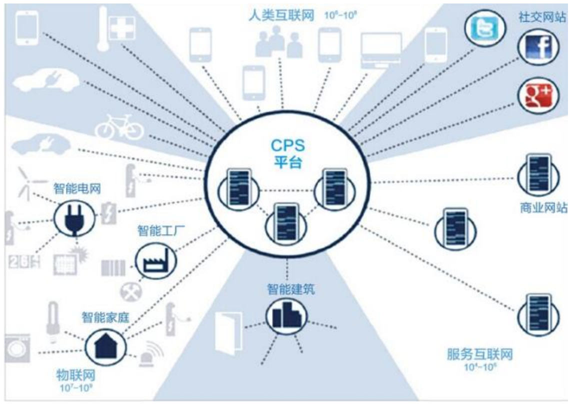
- Homework and Exercises 20%
- Mid-Term Exam...10%   in class.
- Final Exam………………………………….......... 70%

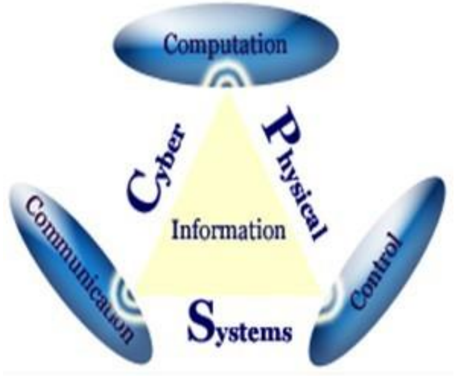
Introduction

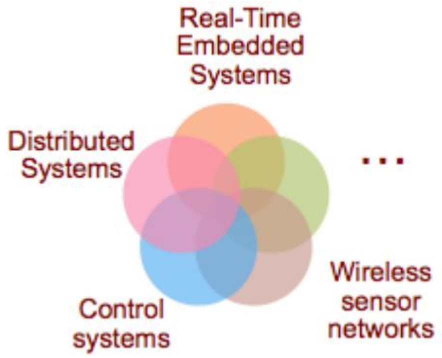
What are Cyber-Physical Systems?

- **Cyber** – computation, communication, and control that are discrete, logical, and switched;
- **Physical** – natural and human-made systems governed by the laws of physics and operating in continuous time;
- **Cyber-Physical Systems** – systems in which the cyber and physical systems are tightly integrated at all scales and levels;

“CPS will transform how we interact with the physical world just like the Internet transformed how we interact with one another.”







Characteristics of Cyber-Physical Systems

- Some defining characteristics:
  - Cyber – physical coupling driven by new demands and applications
    - Cyber capability in every physical component
    - Large scale wired and wireless networking
    - Networked at multiple and extreme scales
  - Systems of systems
  - New spatial-temporal constraints
    - Complex at multiple temporal and spatial scales
    - Dynamically reorganizing/reconfiguring
    - Unconventional computational and physical substrates (Bio? Nano?)
  - Novel interactions between communications/computing/control
    - High degrees of automation, control loops must close at all scales
    - Large numbers of non-technical savvy users in the control loop
  - Ubiquity drives unprecedented security and privacy needs
  - Operation must be dependable, certified in some cases

Characteristics of Cyber-Physical Systems

- What they are not:
  - Not desktop computing
  - Not traditional, post-hoc embedded/real-time systems
  - Not today's sensor nets
- Goals of a CPS research program
  - A new science for future engineered and monitored/controlled physical systems (10-20 year perspective)
  - Physical and cyber (computing, communication, control) design that is deeply integrated

Application Domains of Cyber-Physical Systems

- Healthcare
  - Medical devices
  - Health management networks
- Transportation
  - Automotive electronics
  - Vehicular networks and smart highways
  - Aviation and airspace management
  - Avionics
  - Railroad systems
- Process control
  - Large-scale Infrastructure
  - Physical infrastructure monitoring and control
  - Electricity generation and distribution

Application Domains of Cyber-Physical Systems

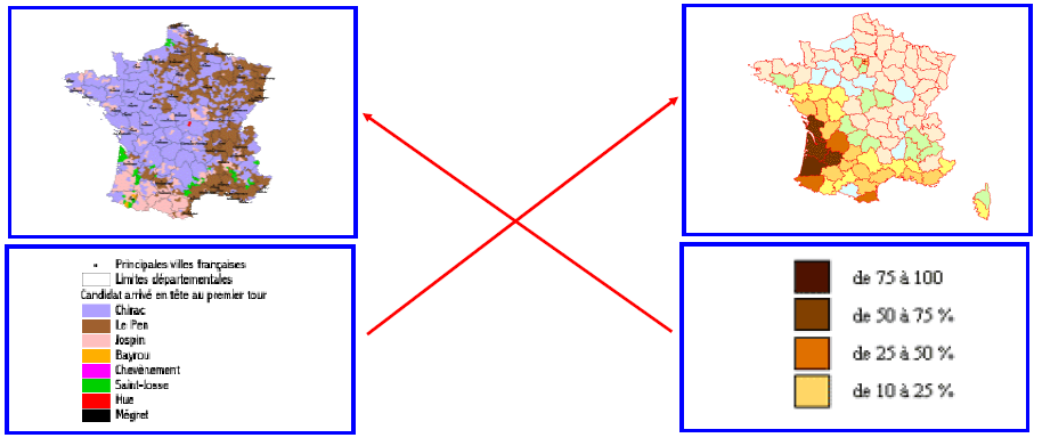
- Defense systems
- Tele-physical operations
  - Telemedicine
  - Tele-manipulation

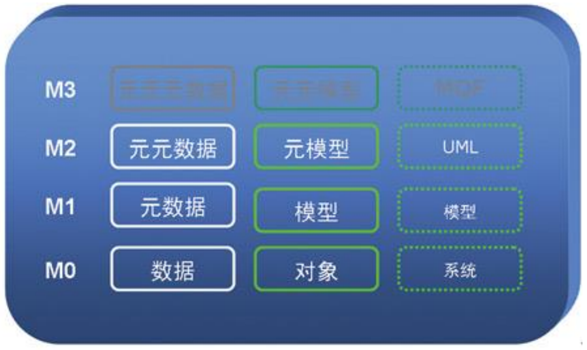
Model-Drive Development

Modeling

- A reduced/abstract representation of some system that highlights the properties of interest from a given point of view.
- The point of view defines concern and scope of the model.

Downtown Seattle
Bicycle Trail
Bicycle Lane
Arterial Street
Residential Street
Ride Free Area
Designates One Way Street
Monorail
SCALE IN 100 FEET
Elliott Bay
元模型





Modeling

- Phil Bernstein, "A Vision for Management of Complex Systems".
  - A model is a complex structure that represents a design artifcat such as a relateional schema,an interface definition(API),an XML schema, a semantic network,a UML model or a hypermedia document.
- OMG,"UML Superstructure".
  - A model captures a view of a physical system. It is an abstraction of the physical system,with a certain purpose. This purpose determines what is included in the model and what is relevant. thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail.
- OMG,"MDA Guide"
  - a formal specification of the function,structure and/or behavior of an application or system.
- Steve Mellor,et al.,"UML Distilled"
  - A model is a simplification of something so we can view.manipulate, and reason about it, and so help us understand the complexity inherent in the subject under study.
- Anneke Kleppe,et.al."MDA Explained"
  - A model is a description of(part of) a system written in a well-defined language. A well-definied language is a language with well-defined form(syntax), and meaning(semantics). which is suitable for automated interpretation by a computer.
- chris Raistrick et al.,"Model Driven Architecture with Executable UML"
  - A formal representation of the function, behavior, and structure of the system we are considering, expressed in an unambiguous language.

Model Driven Architecture

MDA 定义了三种模型：

- **计算独立模型（Computation-Independent Model，CIM）**
- 描述系统的需求和将在其中使用系统的业务上下文。此模型通常描述系统将用于做什么，而不描述如何实现系统。CIM 通常用业务语言或领域特定语言来表示。
- **平台独立模型（Platform-Independent Model，PIM）**
- 描述如何构造系统，而不涉及到用于实现模型的技术。此模型不描述用于为特定平台构建解决方案的机制。PIM 在由特定平台实现时可能是适当的，或者可能适合于多种平台上的实现。
- **平台特定模型（Platform-Specific Model，PSM）**
- 从特定平台的角度描述解决方案。其中包括如何实现 CIM 和如何在特定平台上完成该实现的细节。

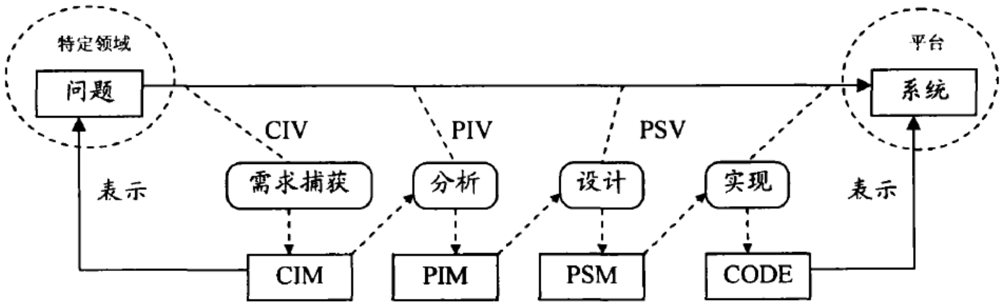


图 2-1 模型驱动开发途径

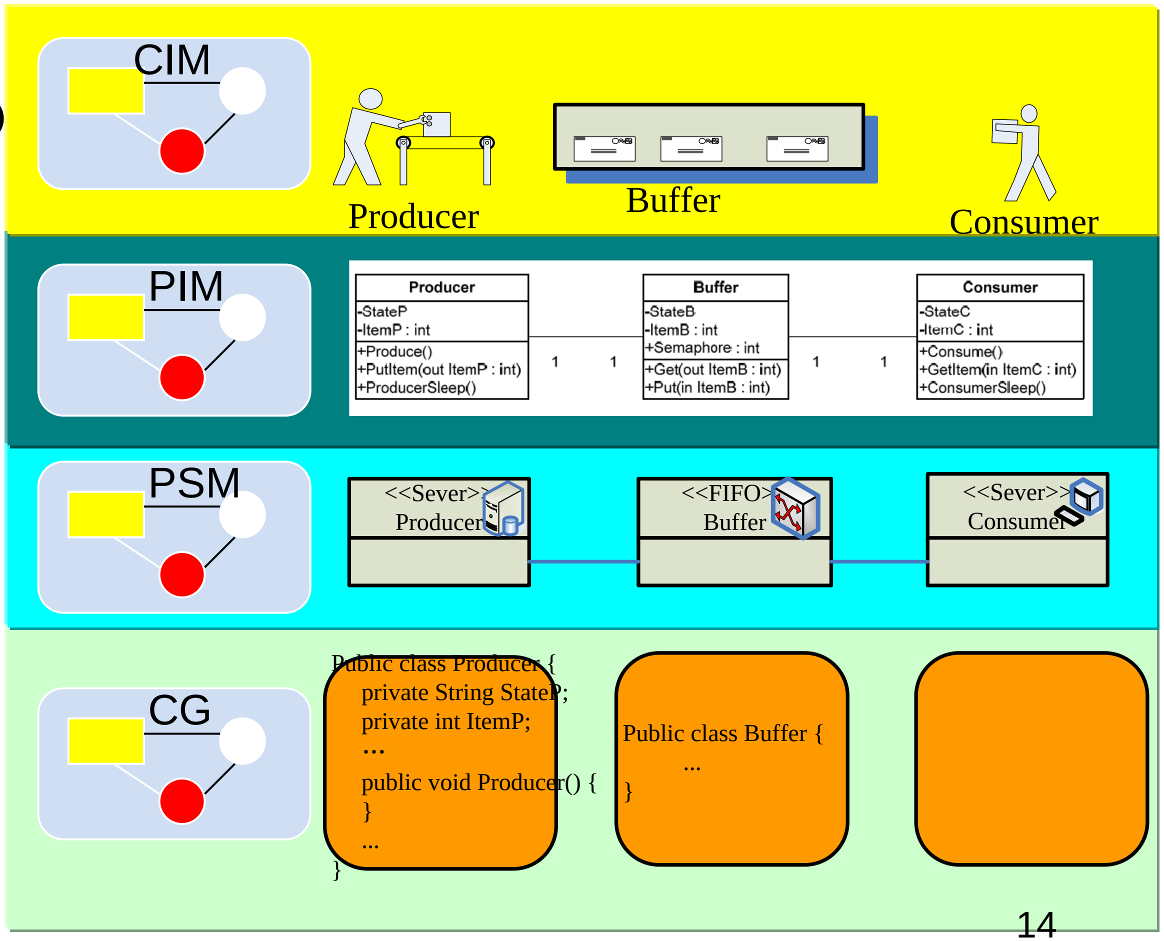
Model Driven Architecture

Computation Independent Model（CIM）

Platform Independent Model(PIM)

Platform Specific Model(PSM)

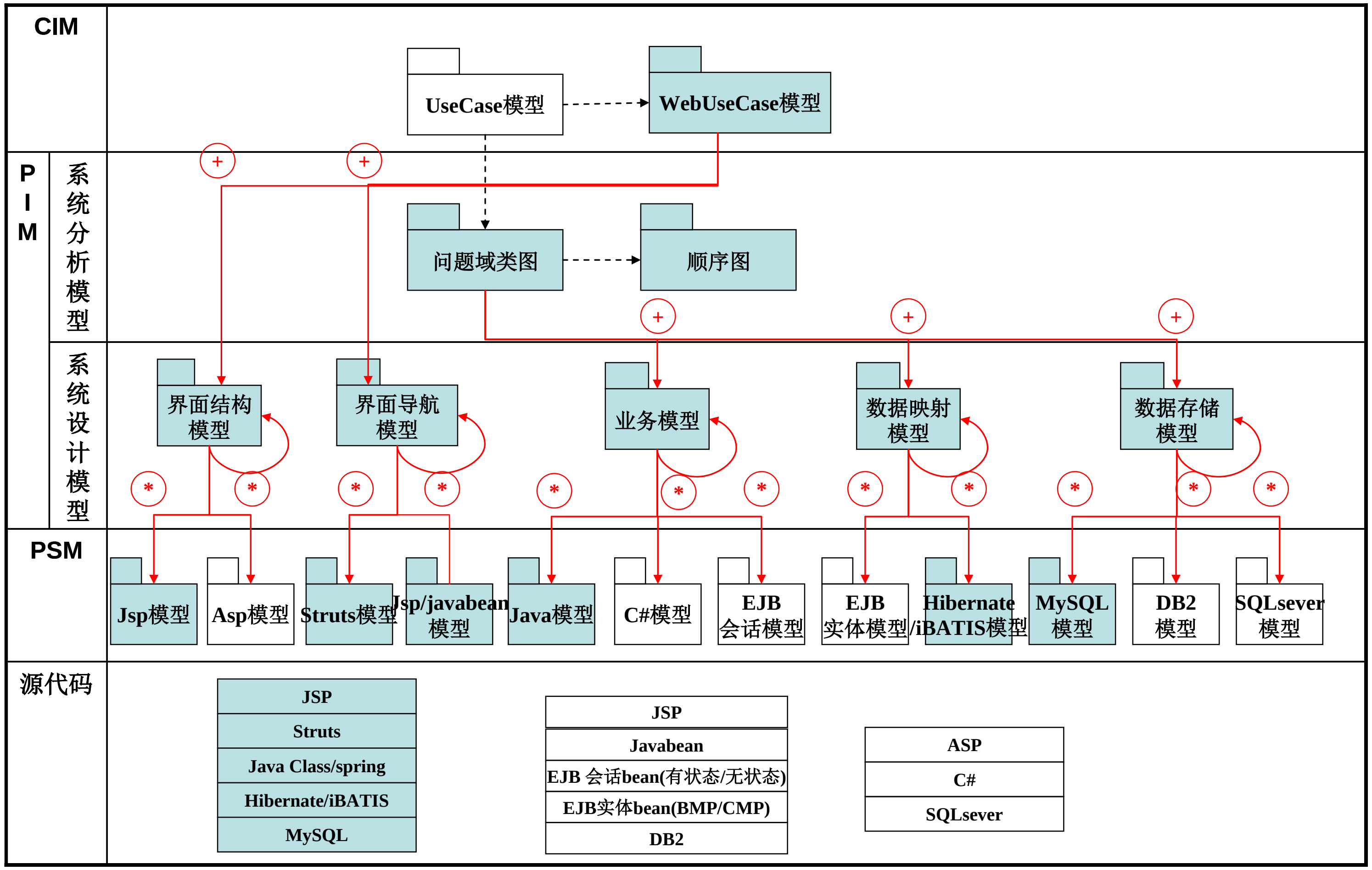
Code generation



```
Public class Producer {
  private String StateP;
  private int ItemP;
  …
  public void Producer() {
  }
  …
}
```

```
Public class Buffer {
    …
}
```

Model Driven Architecture



safety critical systems

What is a Real-Time System?

- Real-time systems have been defined as: "those systems in which the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced";
  - J. Stankovic, "Misconceptions About Real-Time Computing," *IEEE Computer,* 21(10), October 1988.

Example structure of a cyber-physical system

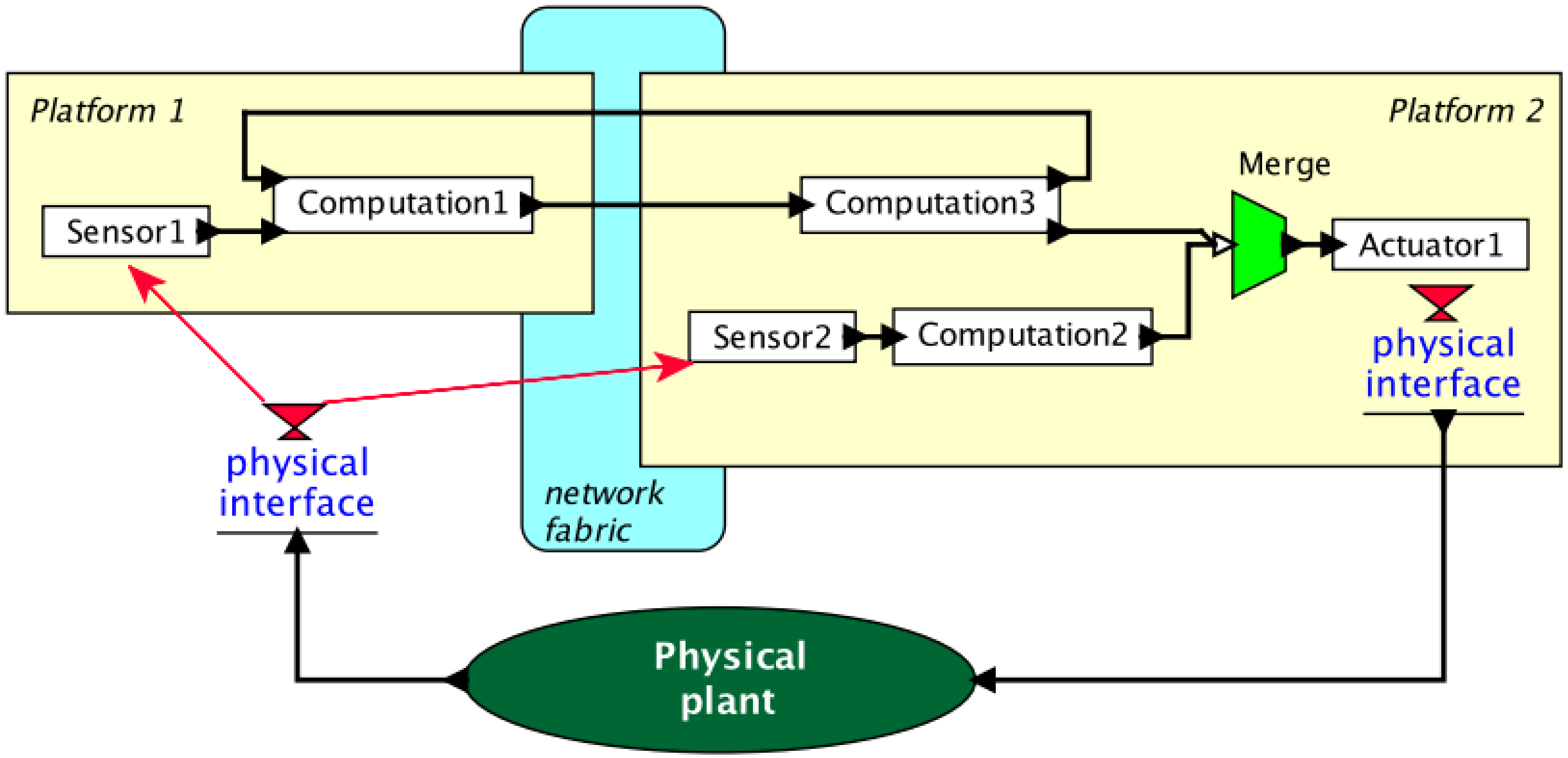
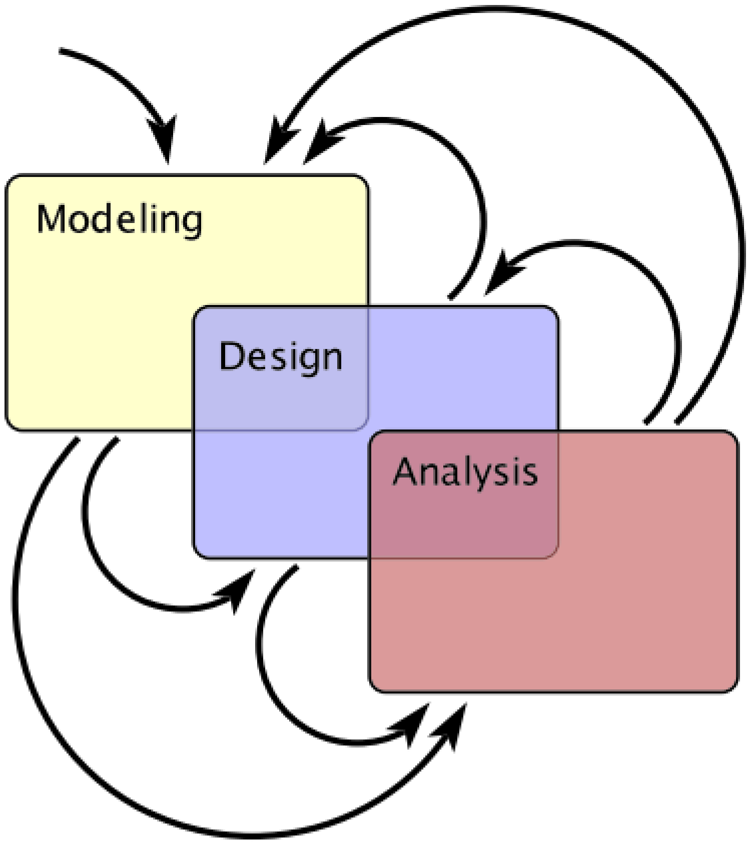


Figure 1.1: Example structure of a cyber-physical system.

The Design Process



Some Definitions

- ***Timing constraint:*** constraint imposed on timing behavior of a job: hard or soft.
- ***Release Time:*** Instant of time job becomes available for execution. If all jobs are released when the system begins execution, then there is said to be no release time
- ***Deadline:*** Instant of time a job's execution is required to be completed. If deadline is infinity, then job has no deadline. Absolute deadline is equal to release time plus relative deadline
- ***Response time:*** Length of time from release time to instant job completes.

Terms and concepts

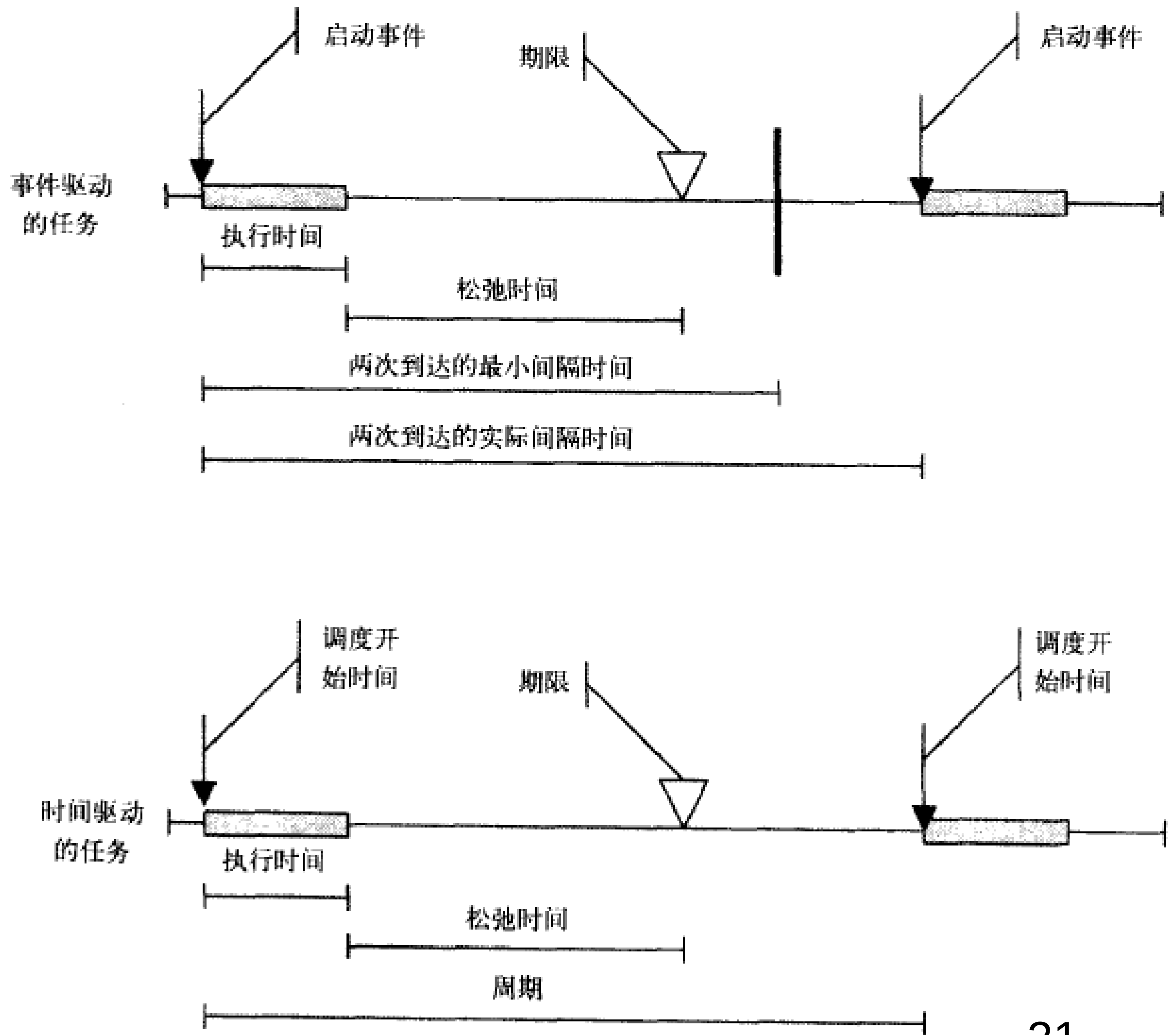


图2-1　期限

Timeliness

- The timeliness of an action has to do with the action meeting time constraints, such as a deadline. deadline maybe hard or soft. Missing a hard deadline constitutes a system failure of some kind, so great care must be taken to ensure that all such actions execute in a timely way. The important modeling concerns of timeliness are modeling execution time, deadlines, arrival patterns, synchronization pattern and time sources.

Timeliness

Remote RobotArm：

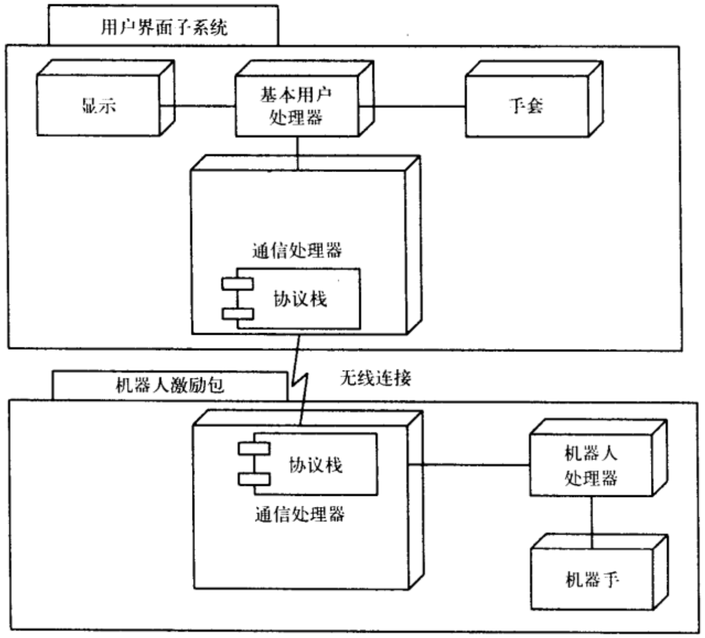
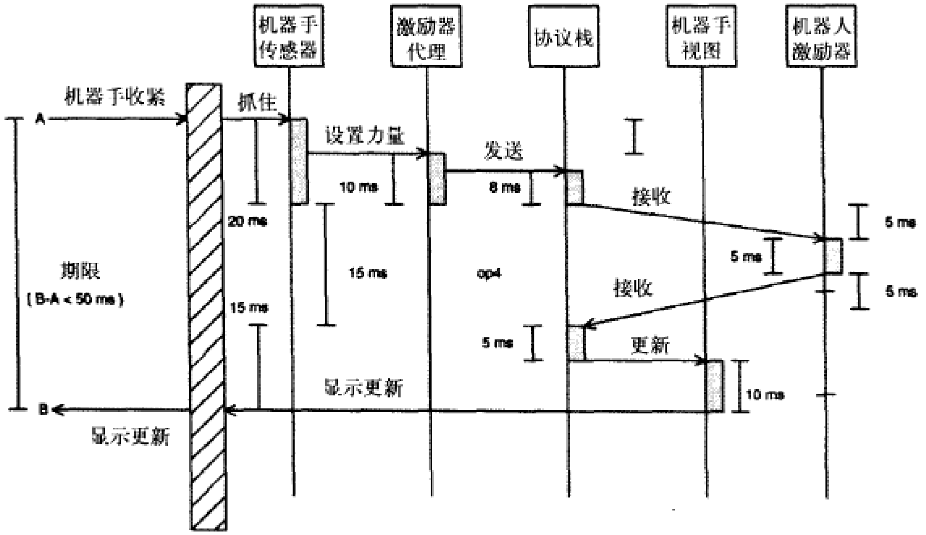


图2-3　远程机器人部署图



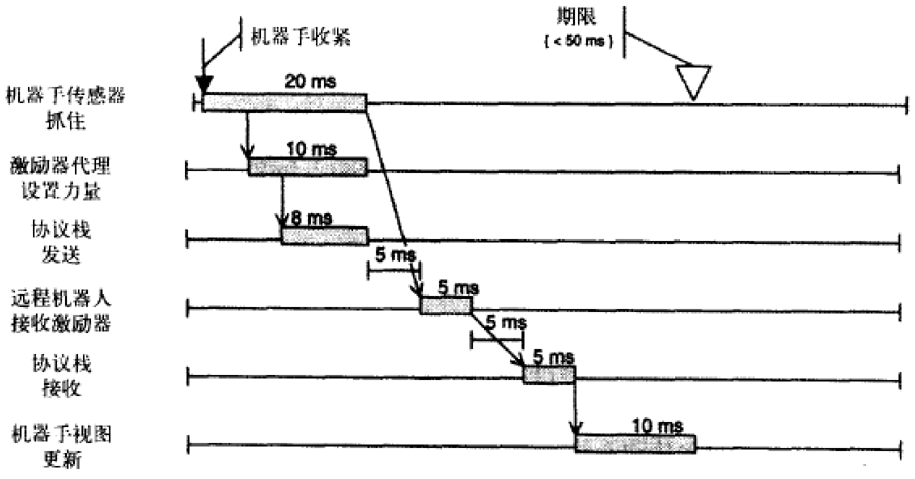


图2-4　远程机器人的时间预算

Concurrency

- Concurrency is the simultaneous execution of multiple sequential chains of actions. These action chains may execute on one processor(pseudo-concurrency) or multiple processors(true concurrency).Issues surrounding the exectuion of concurrent systems have to do with :

✓ the scheduling characteristics of the concurrent thread;

✓ the arrival patterns of incoming events;

✓ the rendezvous patterns used when threads must synchronize;

✓ and the methods of controlling access to shared resources.

Concurrency

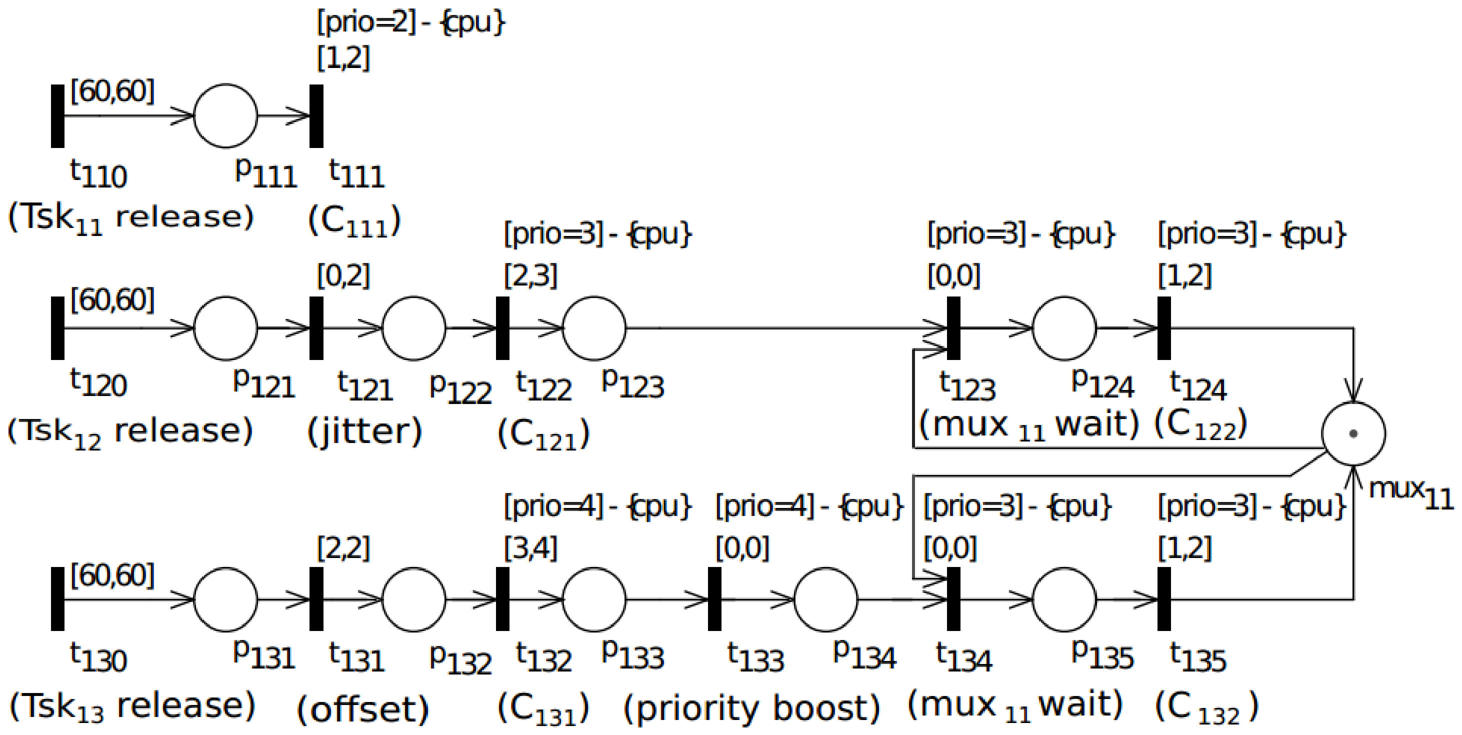


Fig. 2. The pTPN submodel of the Task-Set of application A_1 in the HS system of Table 1.

Predictability

- A key aspect of many real time systems is their predicatbility.

correctness and robustness

- A system is ***correct*** when it does the right thing all the time. Such a system is ***robust*** when it does the right thing under novel(unplanned) circumstances, even in the presence of unplanned failures of portions of the system. one must be on guard for *deadlock, race condition, and other exceptional conditions*.

correctness and robustness-deadlock

- there are four conditions that must be true for deadlock:

✓ task claim exclusive control over shared resources.

✓ task hold resources while waiting for other resource to be relesed.

✓ task connot be forced to relinquish resources.

✓ A circular waiting condition exists.

correctness and robustness-deadlock

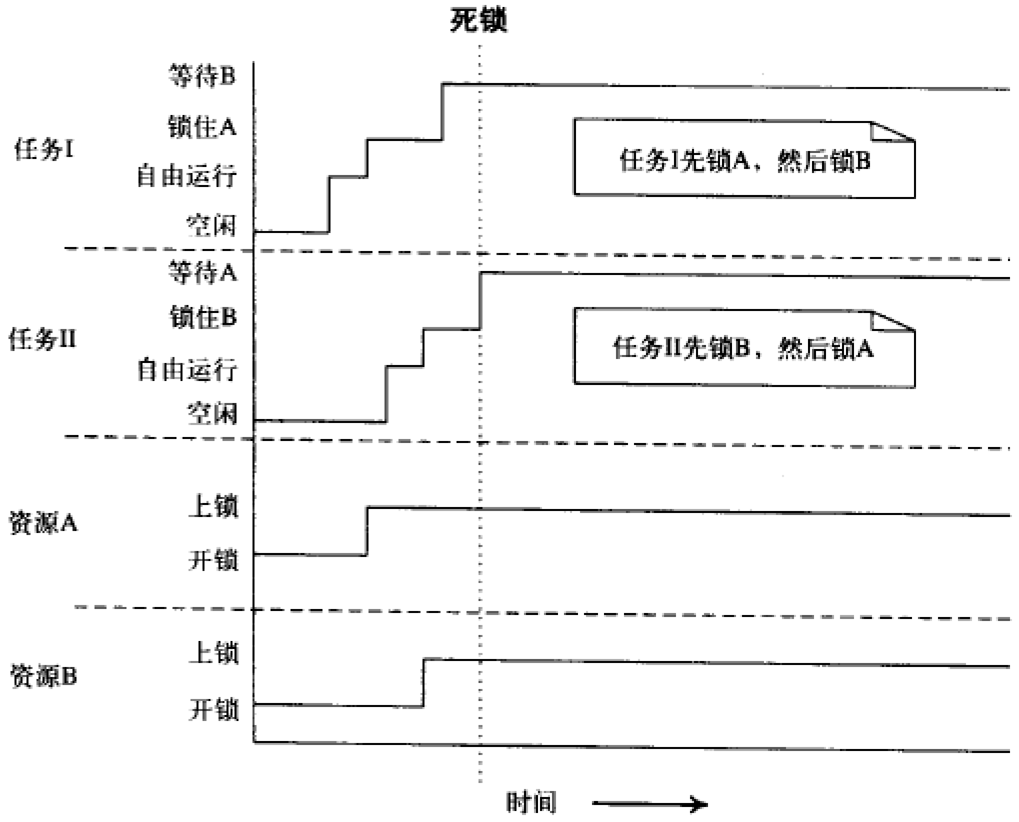
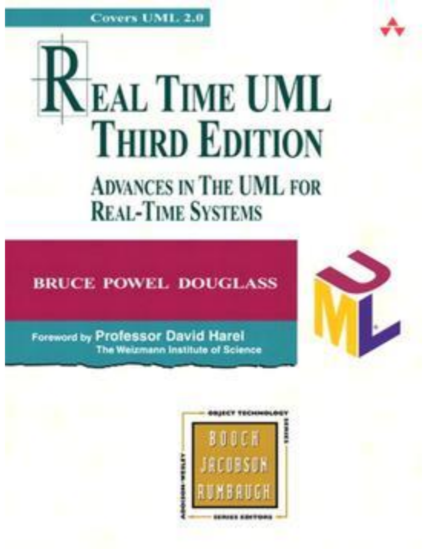


图2-5 死锁状态

- Real-time systems
- Real-time scheduling algorithms
  - Fixed-priority algorithm (RM)
  - Dynamic-priority algorithm (EDF)



Covers UML 2.0

REAL TIME UML THIRD EDITION

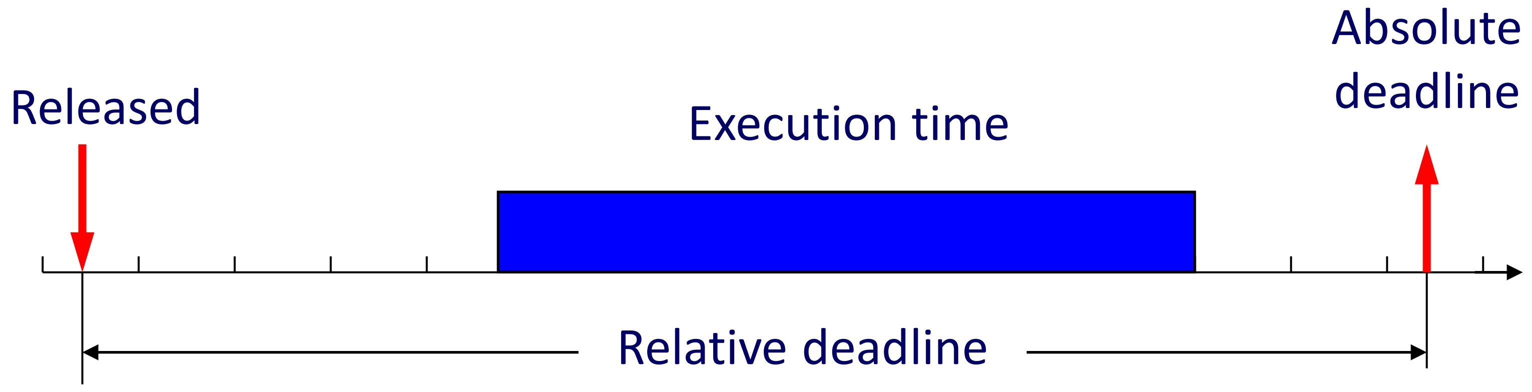
ADVANCES IN THE UML FOR REAL-TIME SYSTEMS

BRUCE POWEL DOUGLASS

Foreword by Professor David Harel
The Weizmann Institute of Science

Real-time workload

- Job (unit of work)
  - a computation, a file read, a message transmission, etc
- Attributes
  - Resources required to make progress
  - Timing parameters

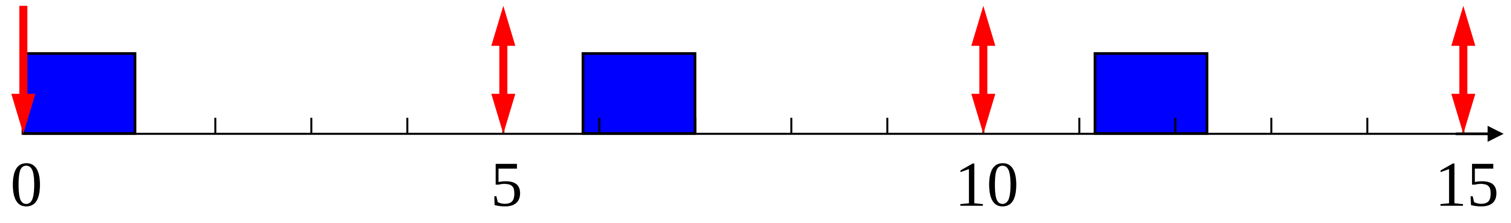


Real-time task

- Task : It starts with reading of input data and of its internal state and terminates with the production of the results and updates internal state.
- A task that does not have an internal state at its point of invocation is called a stateless task, otherwise, it is called a statefull task.

Real-time task

- Task : a sequence of similar jobs
  - Periodic task (p,e)
    - Its jobs repeat regularly
    - Period p = inter-release time (0 < p)
    - Execution time e = maximum execution time (0 < e < p)
    - Utilization U = e/p



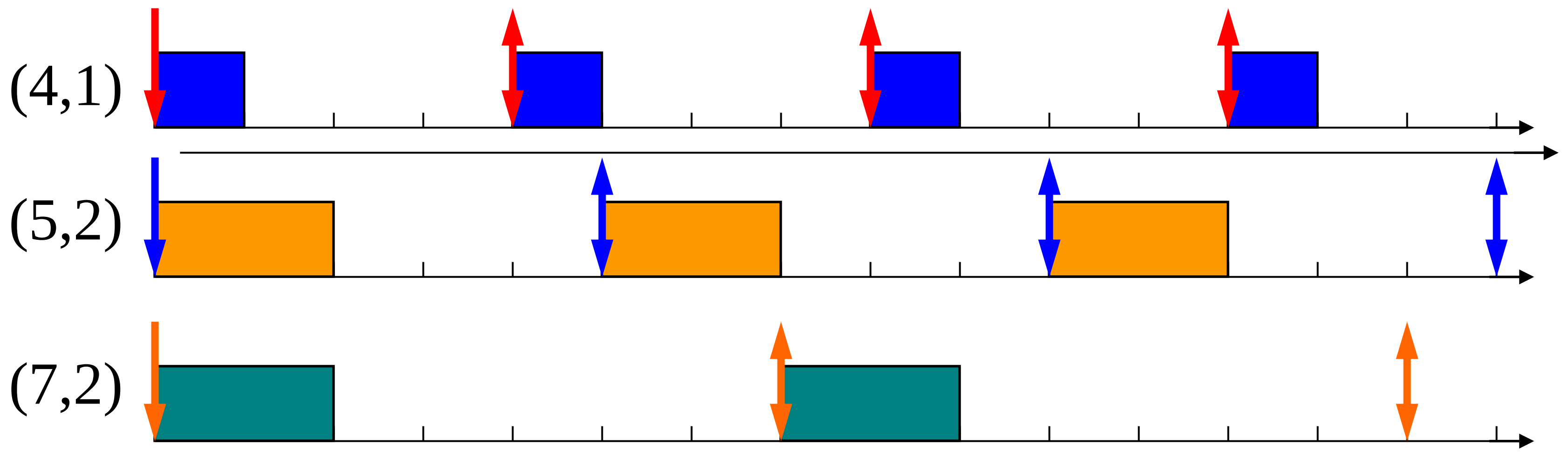
Deadlines: Hard vs. Soft

- Hard deadline
  - Disastrous or very serious consequences may occur if the deadline is missed
  - Validation is essential : can all the deadlines be met, even under worst-case scenario?
  - Deterministic guarantees

- Soft deadline
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
  - Best effort approaches / statistical guarantees

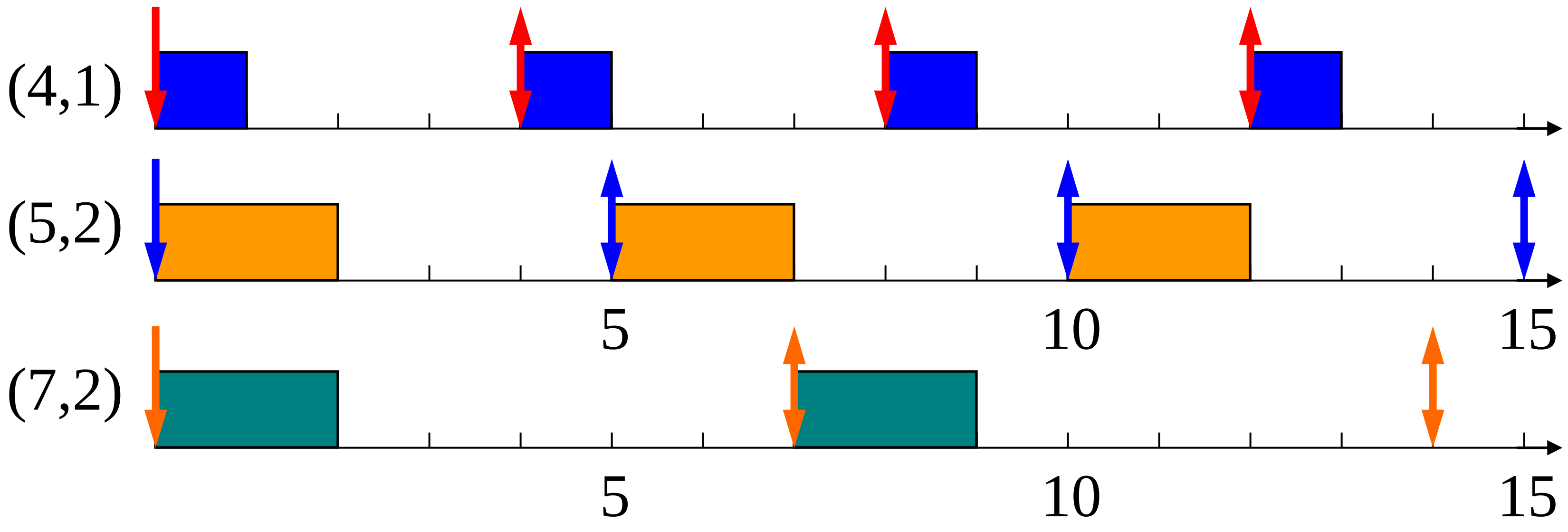
Schedulability

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines



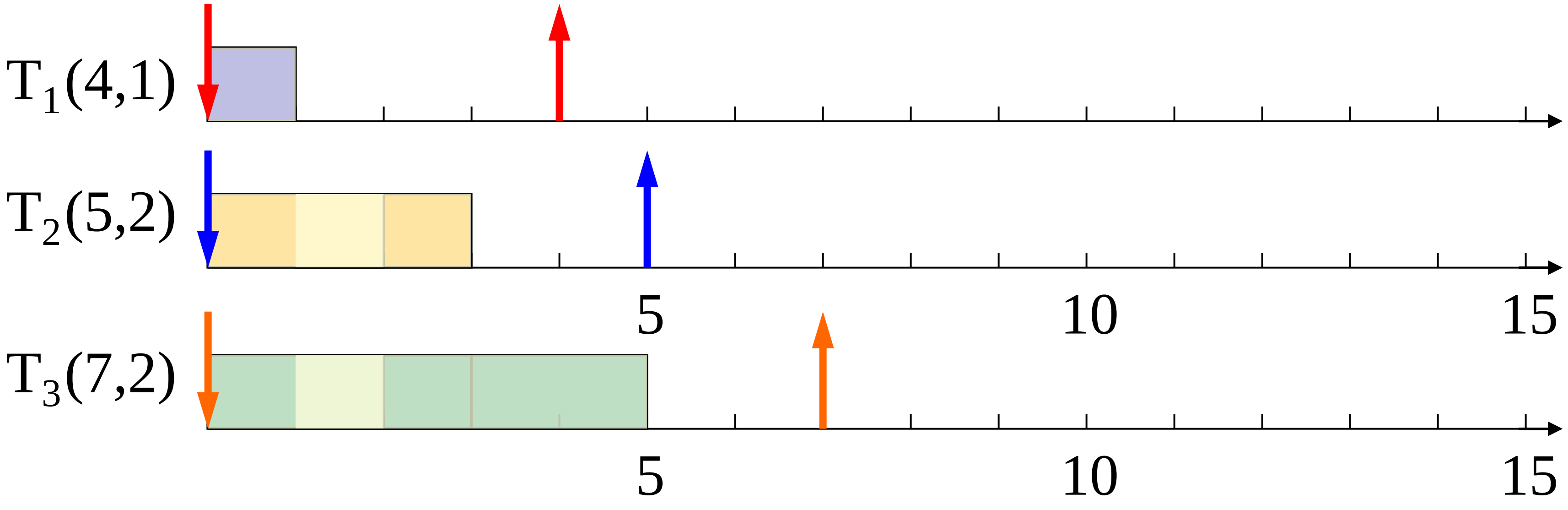
Real-Time Scheduling

- Determines the order of real-time task executions
- Static-priority scheduling
- Dynamic-priority scheduling



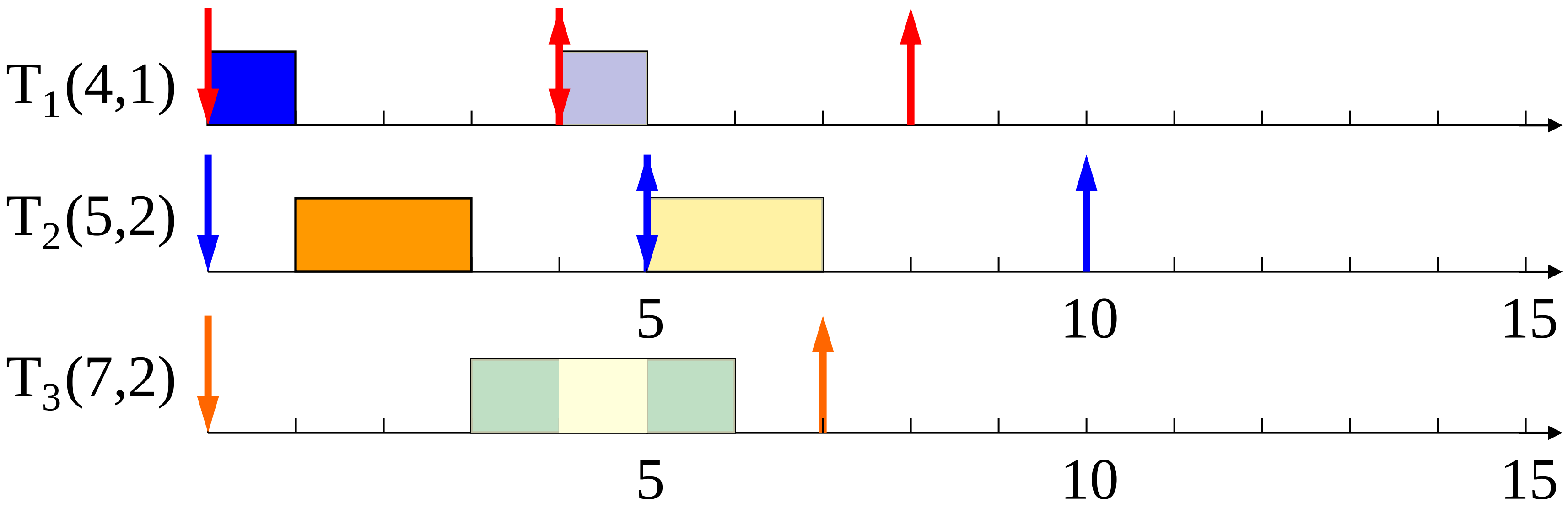
RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period



RM (Rate Monotonic)

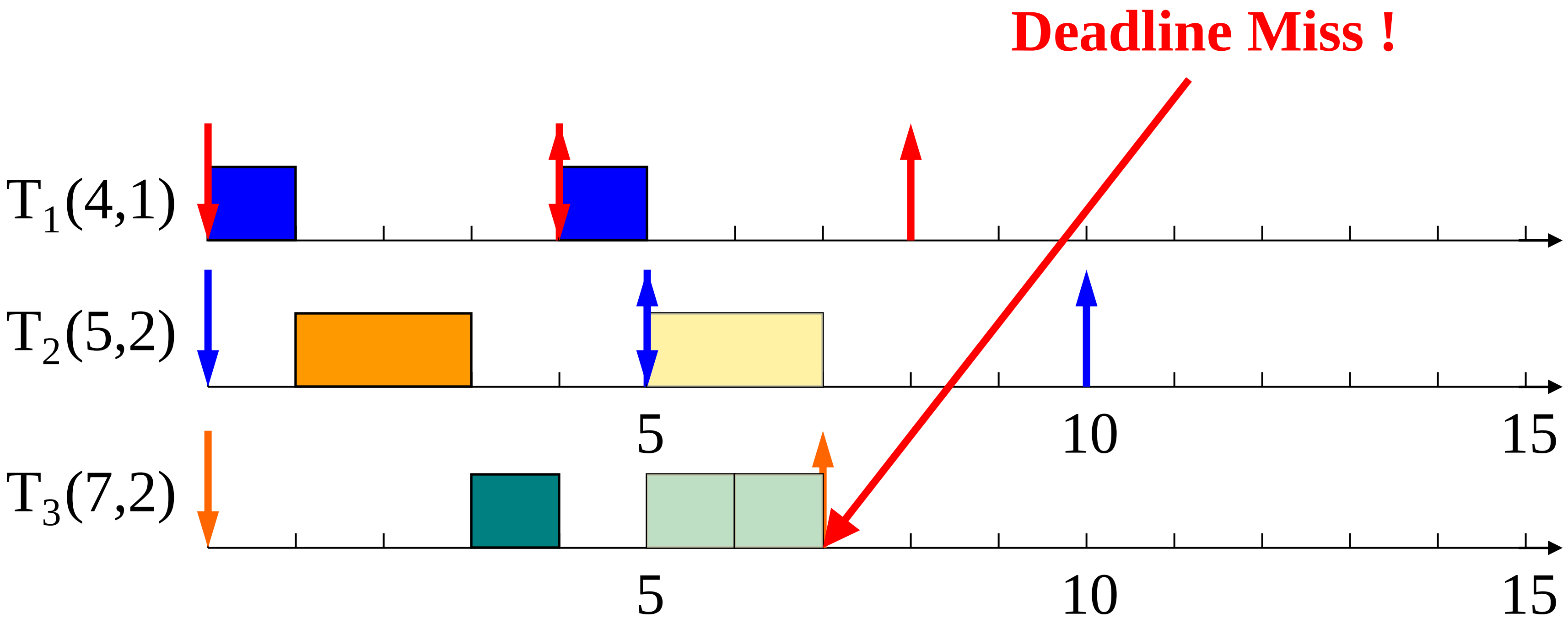
- Executes a job with the shortest period



RM (Rate Monotonic)

The original Rate Monotonic approach had several restrictions:
- ➢ all tasks are independent of each other(e.g. they do not interact)
- ➢ all tasks are periodic
- ➢ no task can block waiting for an external event.
- ➢ all tasks share a common release time(caled the critical instant)
- ➢ all tasks have a deadline equal to their period.

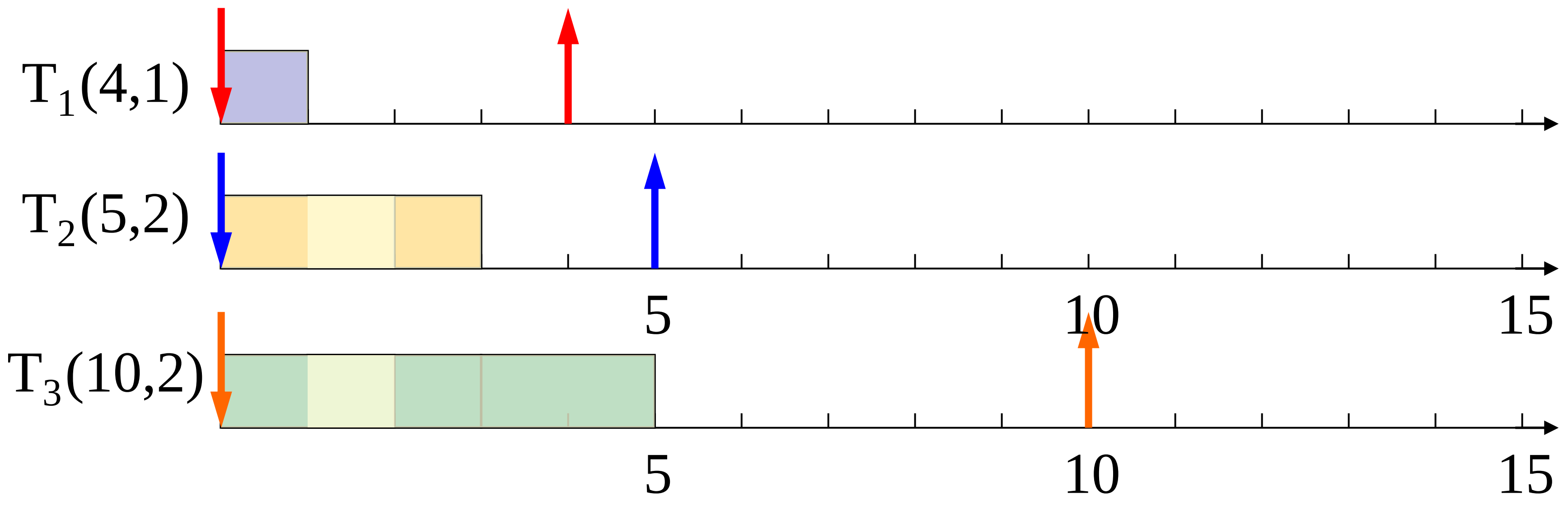
- Executes a job with the shortest period



Response Time

The original Rate Monotonic approach had several restrictions:
- all tasks are independent of each other(e.g. they do not interact)
- all tasks are periodic
- no task can block waiting for an external event.
- all tasks share a common release time(caled the critical instant)
- all tasks have a deadline equal to their period.

- Response time
  - Duration from released time to finish time

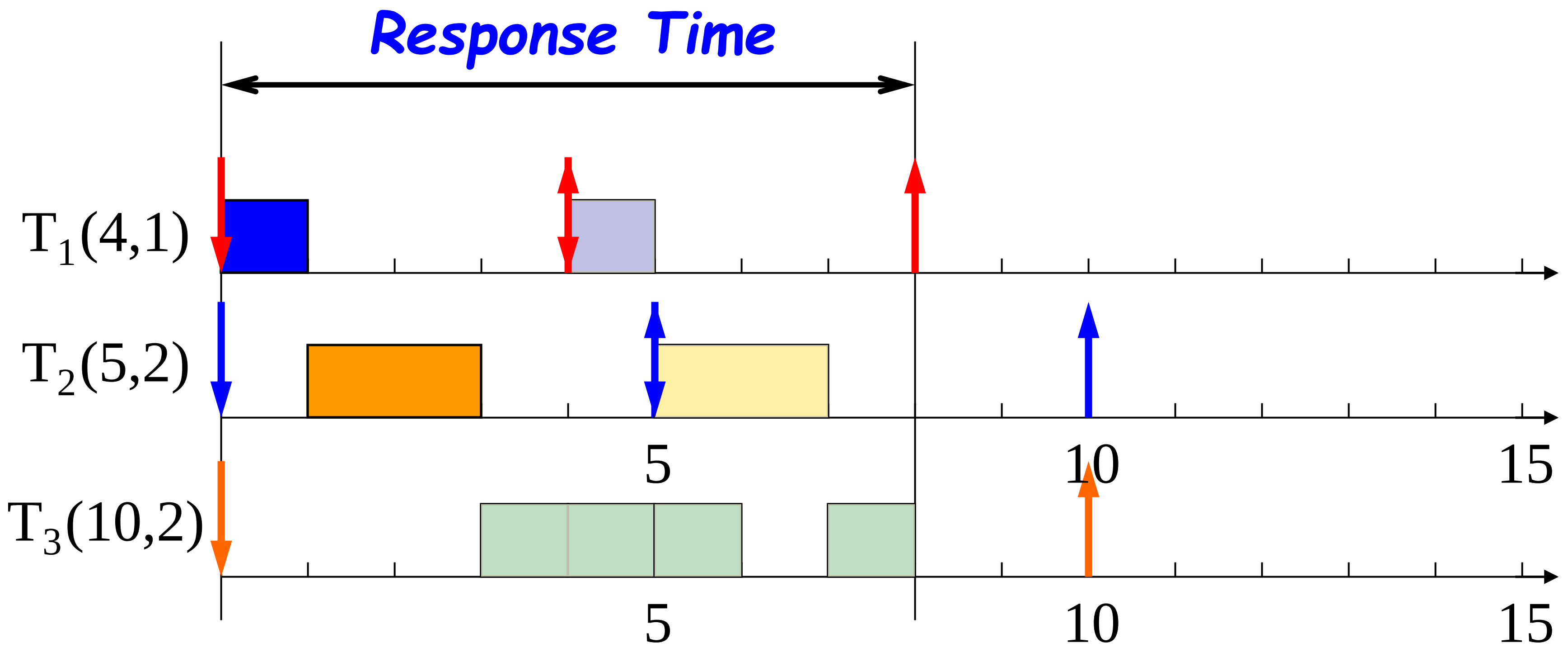


Response Time

The original Rate Monotonic approach had several restrictions:

- all tasks are independent of each other(e.g. they do not interact)
- all tasks are periodic
- no task can block waiting for an external event.
- all tasks share a common release time(caled the critical instant)
- all tasks have a deadline equal to their period.

- Response time
  - Duration from released time to finish time



RM – Utilization Bound

- Real-time system is schedulable under RM if

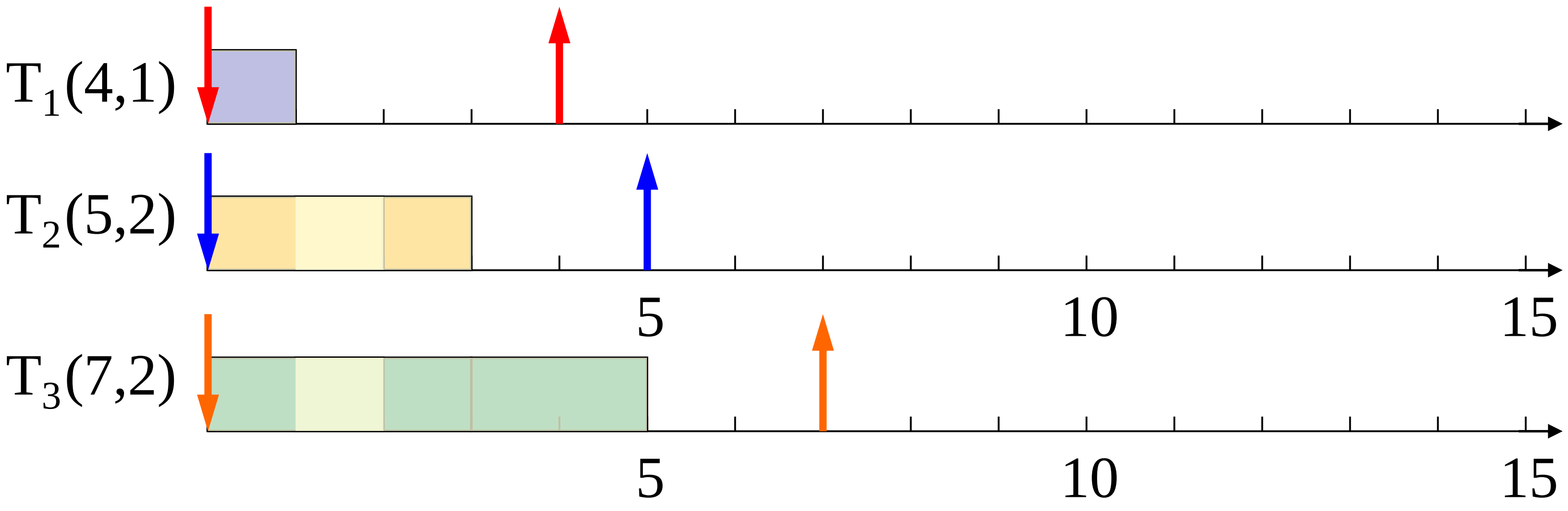
$$\sum U_i \leq n\,(2^{1/n}-1)$$

- Example: T1(4,1), T2(5,1), T3(10,1),
- $\sum U_i = 1/4 + 1/5 + 1/10$
- $= 0.55$
- $3\,(2^{1/3}-1) \approx 0.78$

- Thus, $\{T_1, T_2, T_3\}$ is schedulable under RM.

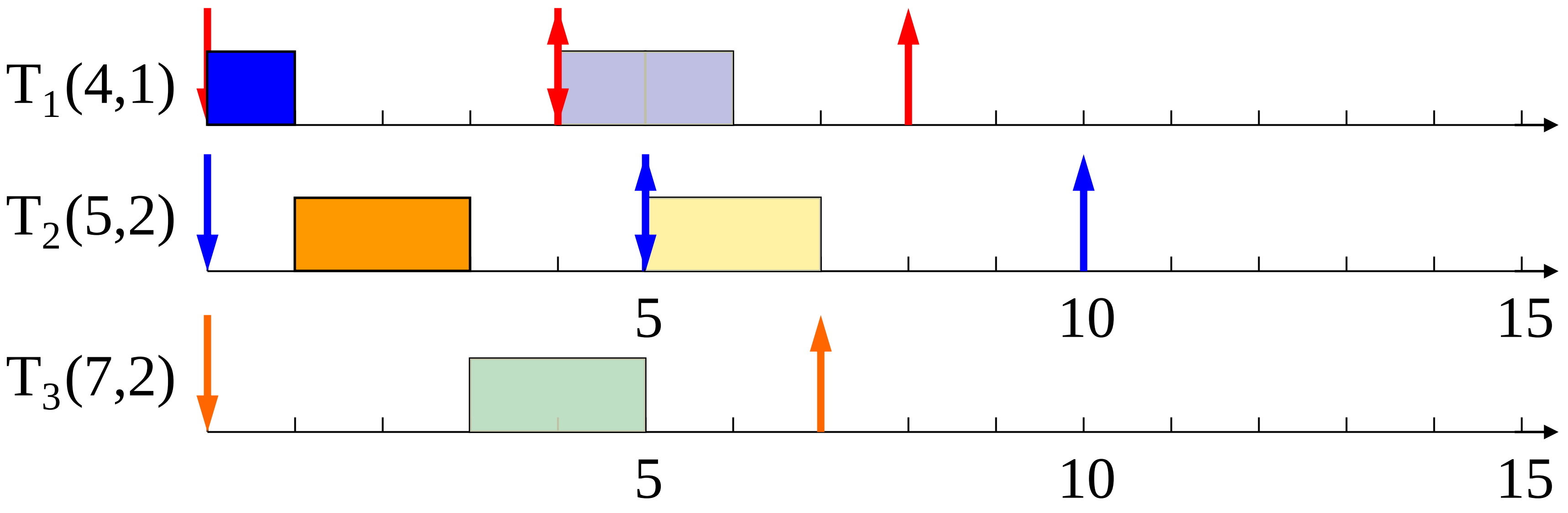
EDF (Earliest Deadline First)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



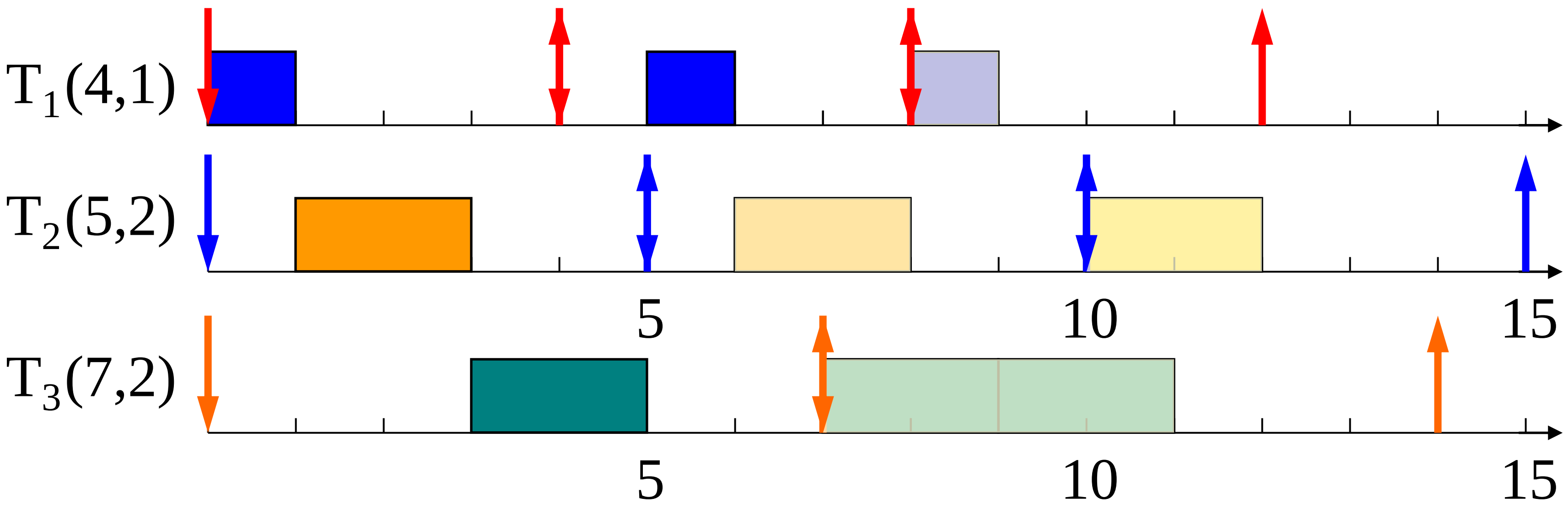
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



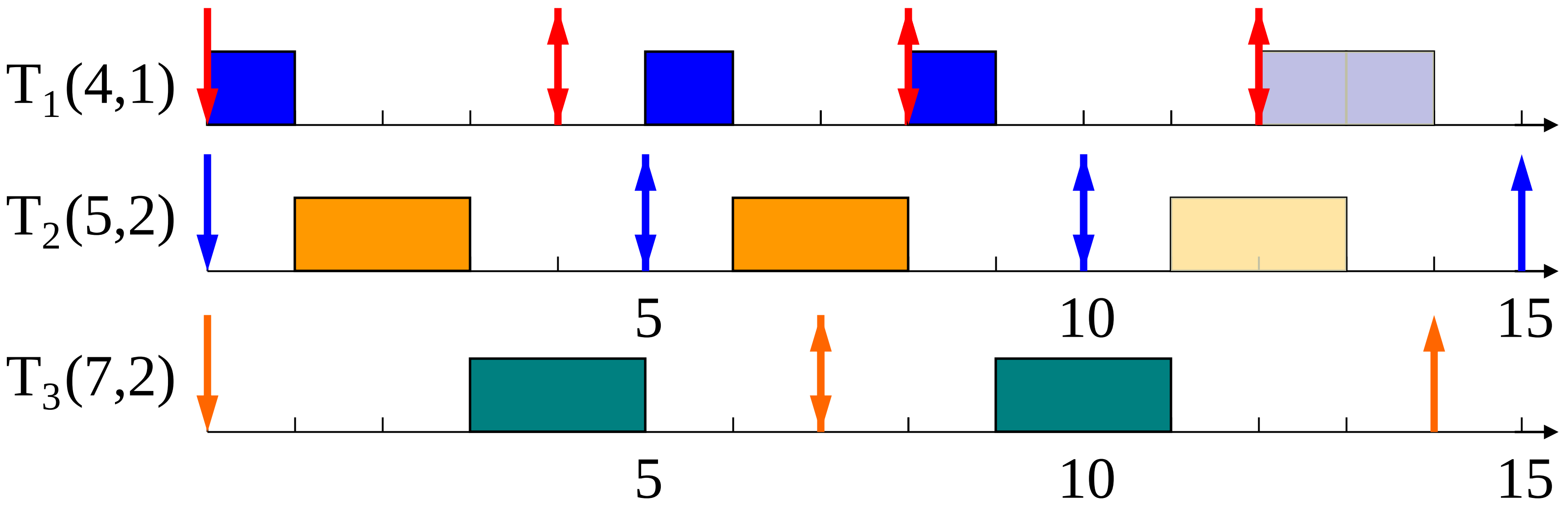
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



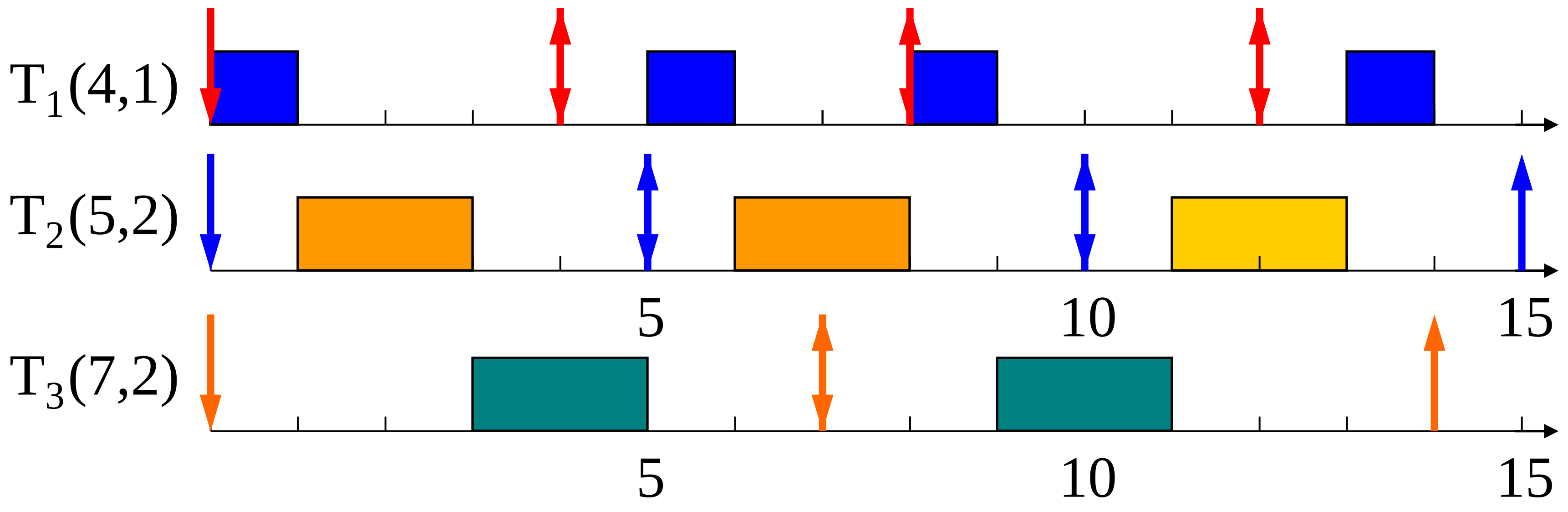
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



EDF (Earliest Deadline First)

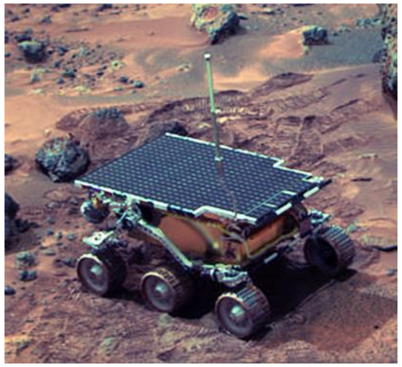
- Optimal scheduling algorithm
  - if there is a schedule for a set of real-time tasks, EDF can schedule it.



实时操作系统特点

提高内核实时性的方法—任务互斥、同步

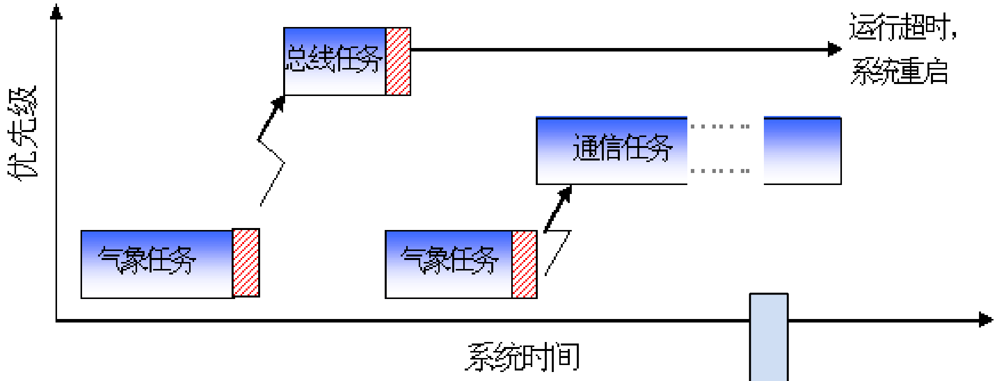
- 资源有限等待：任务没能获得需要的资源会被阻塞。如果资源不是任务继续运行必备的，任务可选择有限等待该资源。
- 优先级反转问题解决—抢占式任务调度中的资源竞争：
  - 1997年7月4日，火星探路者在火星表面成功着陆并进行观测，发回了各种火星表面全景图，被宣称为“完美”。但是在着陆后的第10天，也就是开始采集气象资料后不久，探路者开始出错，反复无规律地重启，每次重启都造成了数据丢失.JPL（美国国家航空航天局喷气推进实验室）的工程师们花了相当多的时间在实验室仿真，希望能够再现引起重启的情况。几天过去了，一个清晨，火星上那台探路者兄弟身上发生的重启情况终于被再现了。经过数据分析，得出了原因——优先级逆转。



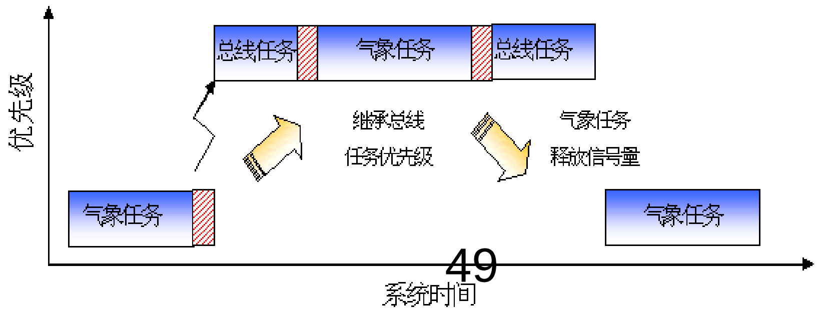
实时操作系统特点

提高内核实时性的方法——优先级反转问题

- 实时操作系统——优先级逆转现象



- 多任务内核应允许动态改变任务的优先级以避免发生优先级反转现象。为防止发生优先级反转,内核能自动变换任务的优先级,这叫做优先级继承(Priority inheritance)



采用**优先级继承**协议消除

modeling with UML

Overview: modeling with UML

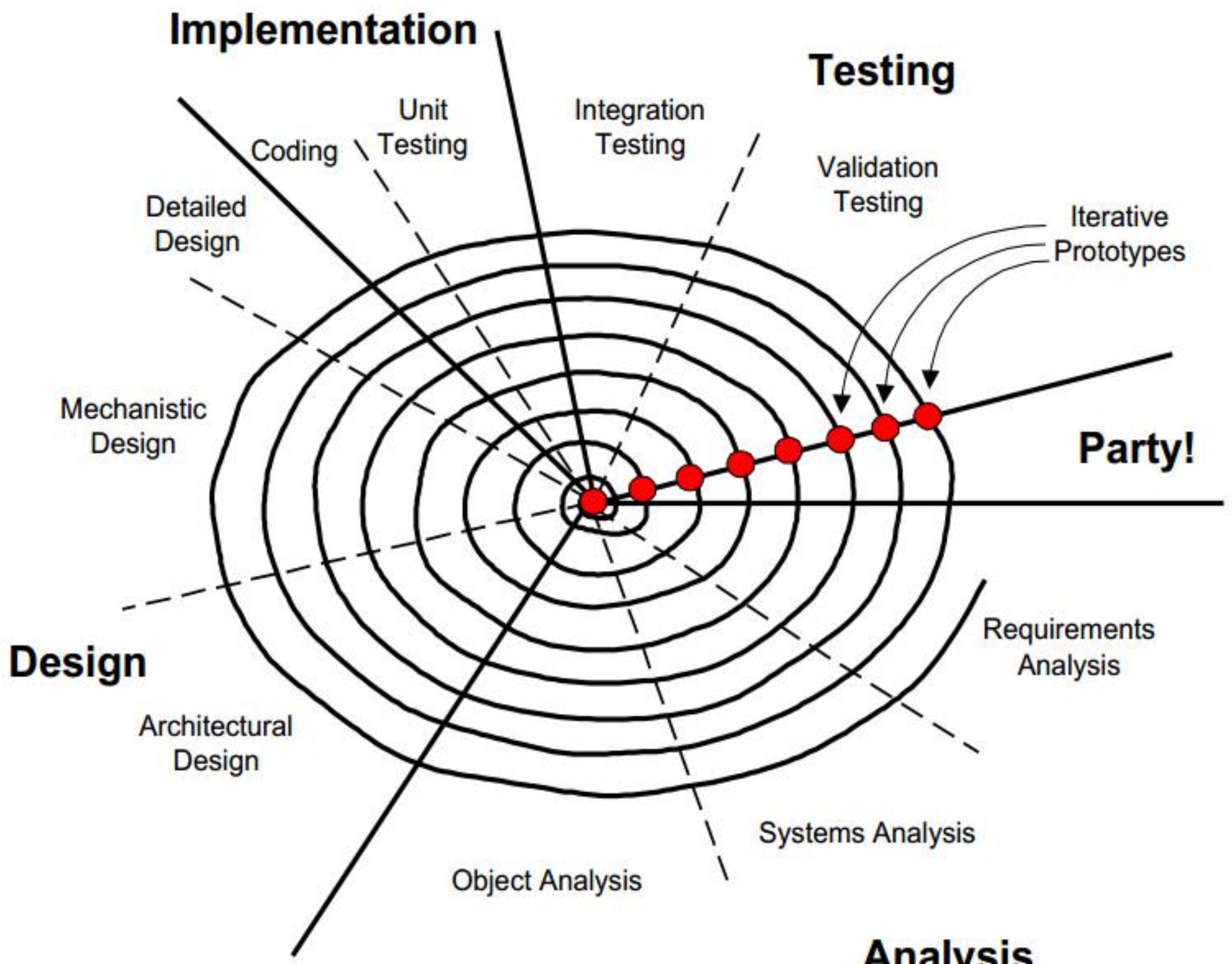
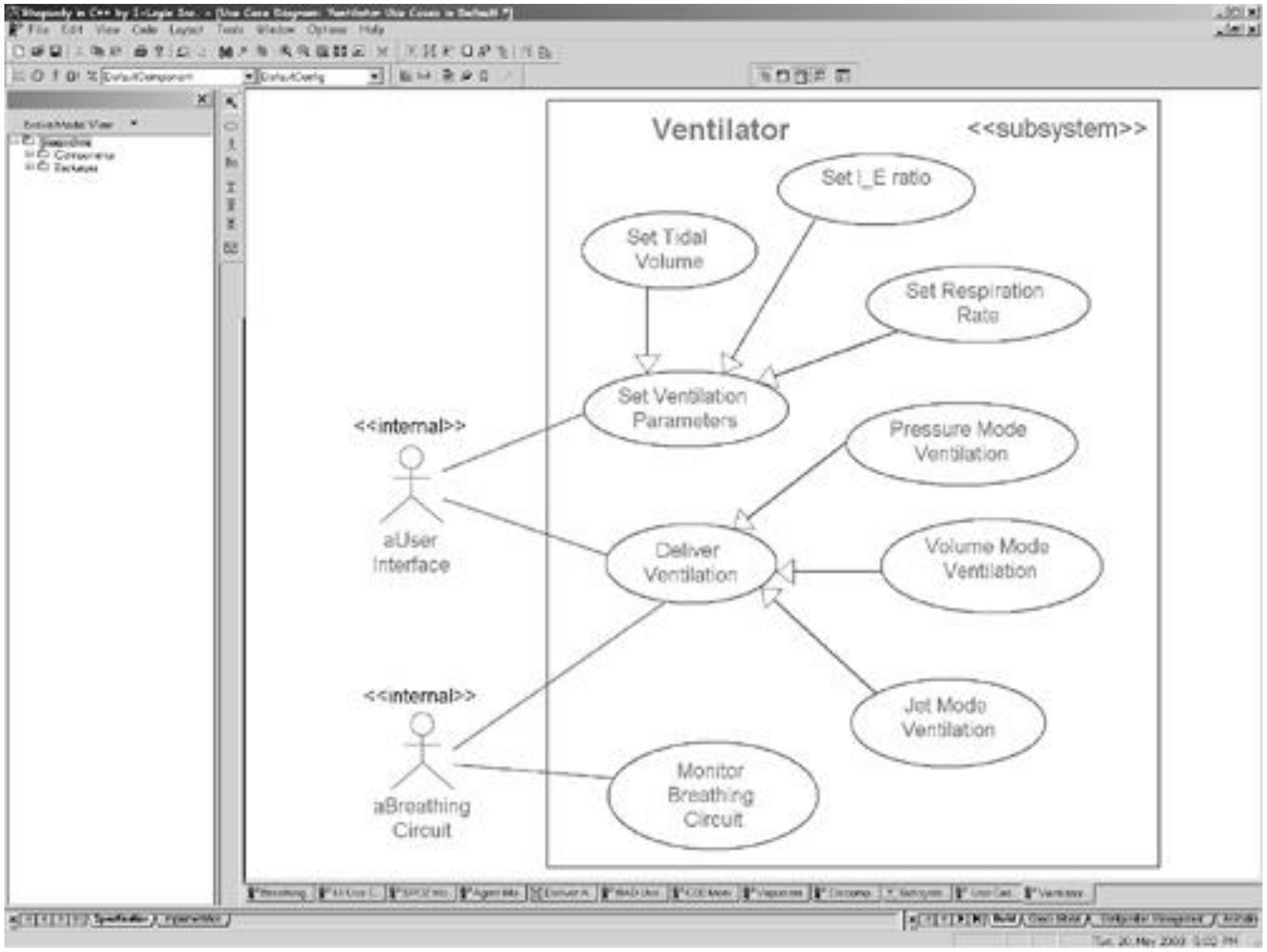


Figure 4: ROPES Process

Overview: modeling with UML



Overview: modeling with Dynamic Behavior

- Types of behaviors
- states Diagrams
- Activity Diagrams
- sequence diagrams

Behavior and the UML

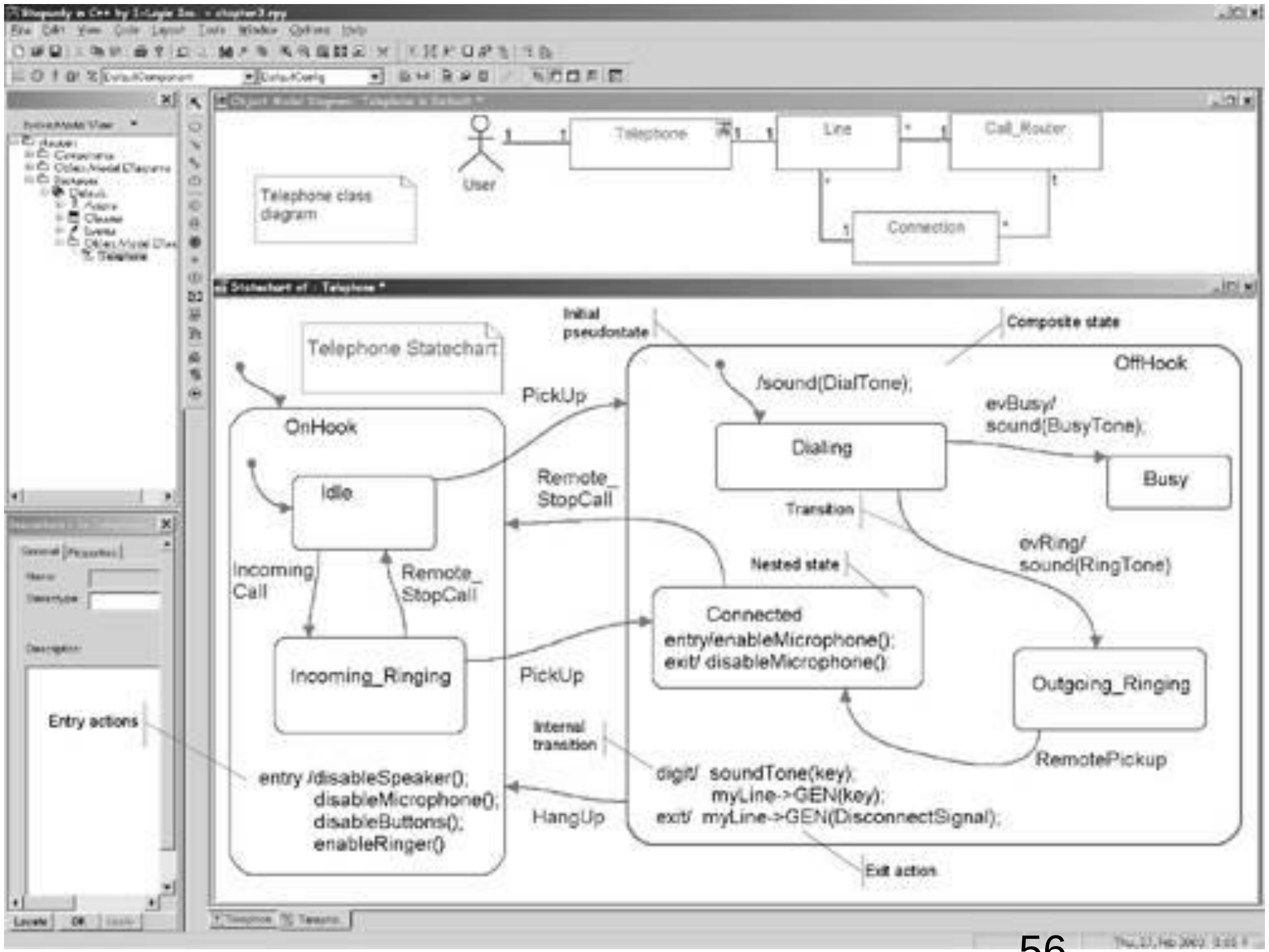
- other Non-functional constraints frequently apply, as well. such constraints are refered to as QOS constraints. and quality how that behavior is accomplished. For example: how long should an action take?Is two-digit accuracy enough?

- QOS constraints are generally modeled using a constraint language. (OCL)

Behavior and the Single Object

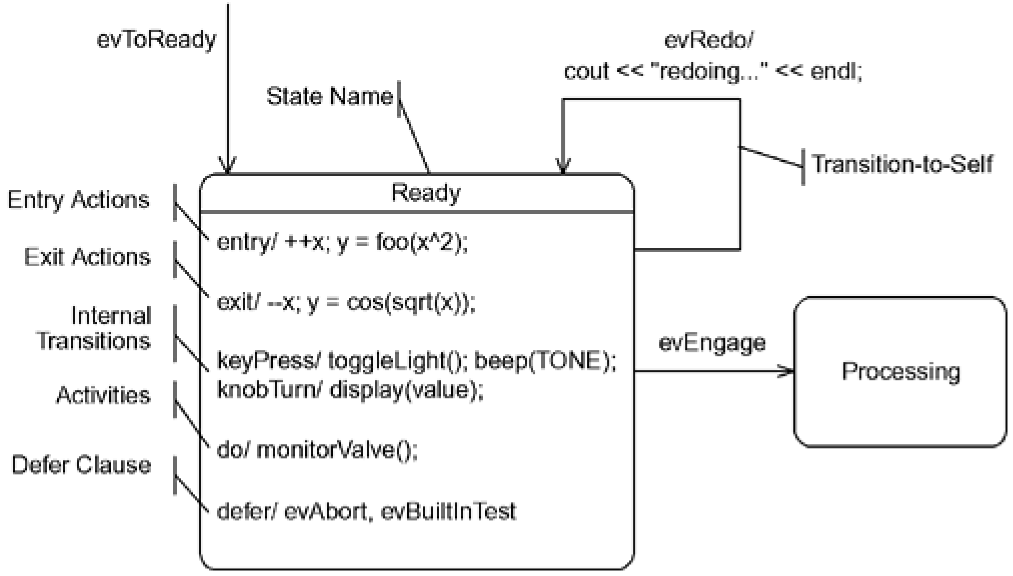
- The primary feature of the UML is its support for finite state machines.
- The two fundamental concepts of statecharts are *State* and *Transition*.
- A State is a condition of existence of an instance that is distinguishable from other such conditions.
- States that are disjoint are called or-states because the object must be in one such state or another.

Basic Statechart Elements



State Features

- The object may execute actions when a state is entered or exited, when an event is received (although a transition isn't taken), or when a transition is taken.


evToReady
State Name
evRedo/
cout << "redoing..." << endl;
Transition-to-Self
Entry Actions
Exit Actions
Internal Transitions
Activities
Defer Clause
Ready
entry/ ++x; y = foo(x^2);
exit/ --x; y = cos(sqrt(x));
keyPress/ toggleLight(); beep(TONE);
knobTurn/ display(value);
do/ monitorValve();
defer/ evAbort, evBuiltInTest
evEngage
Processing

Events

- The UML defines four kinds of events:
  - ***SignalEvent***: An event associated with a Signal. A Signal is a specification of an asynchronous communication, so a SignalEvent is an event associated with an asynchronously received signal.
  - ***CallEvent***: An event associated with a Call. A Call is a specification of a synchronous communication, so a CallEvent allows one object to affect the state of another by invoking one of its methods directly.
  - ***TimeEvent***: An event associated with the passage of time, usually indicated with either tm(<duration>) or after(<duration>). Almost all TimeEvents are relative time; that is, they specify that the event will occur after the object has been in the specified state for at least <duration> time units. If the object leaves that state before the timeout has elapsed, then the logical timer associated with that duration disappears without creating the timeout event.
  - ***ChangeEvent***: An event associated with the change in value for an attribute. It is rarely used in software applications; however, when a state attribute is memory mapped onto a piece of hardware, it can be used to indicate that the memory address changed value.

Transitions

- Transitions are directed arcs beginning at the starting state and finishing at the target state. Transitions usually have named event triggers optionally followed by actions (i.e., executable statements or operations) that are executed when the transition is taken. The event signature for a transition is of the form:
- event-name '(' parameter-list ')' '[' guard-expression ']' '/' action-list
  - Events may specify formal parameter lists, meaning that events may carry actual parameters.
  - digit(key: keyType)/ show(key)

Guards & Action Execution Order

- The guard expression is a Boolean expression, enclosed between square brackets, that must evaluate to either true or false.

- The order of execution of actions is important. The basic rule is *exit-transition-entry*. That is, the exit action of the predecessor state is executed first, followed by the actions on the transition, followed by the entry actions of the subsequent state.

Guards & Action Execution Order

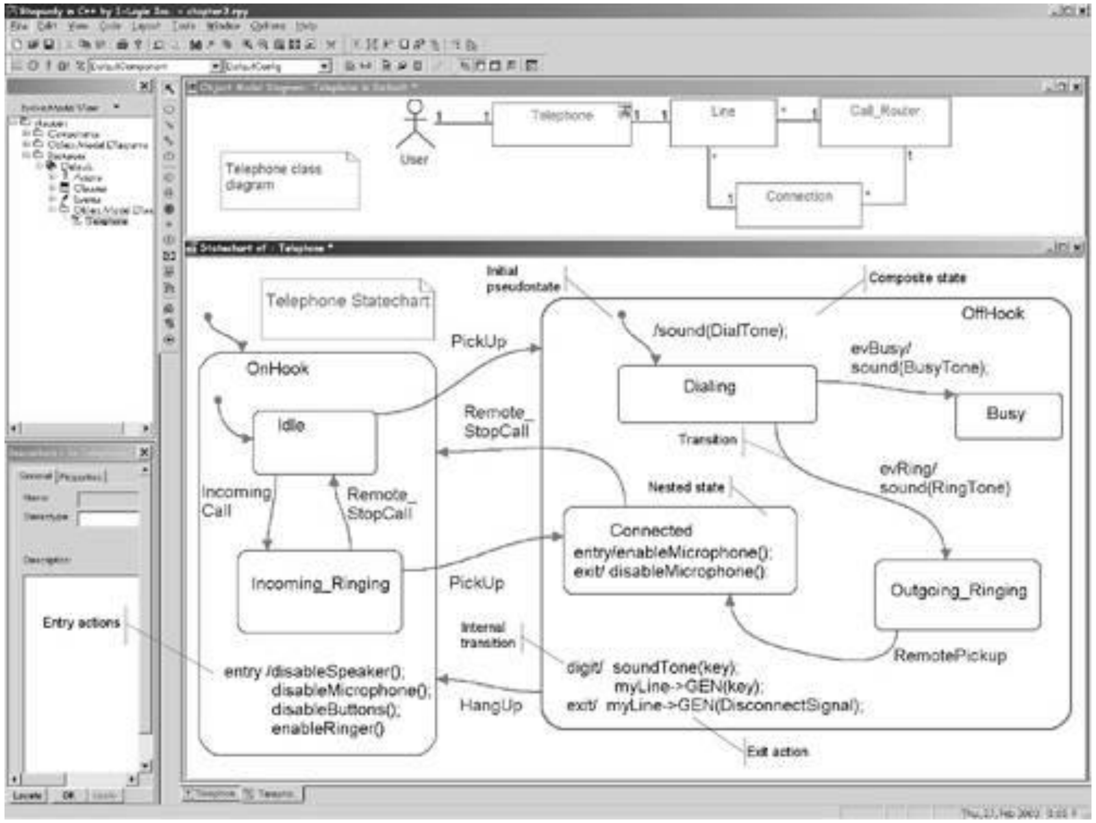
1、IncomingRinging exit actions

2、On Hook exit actions

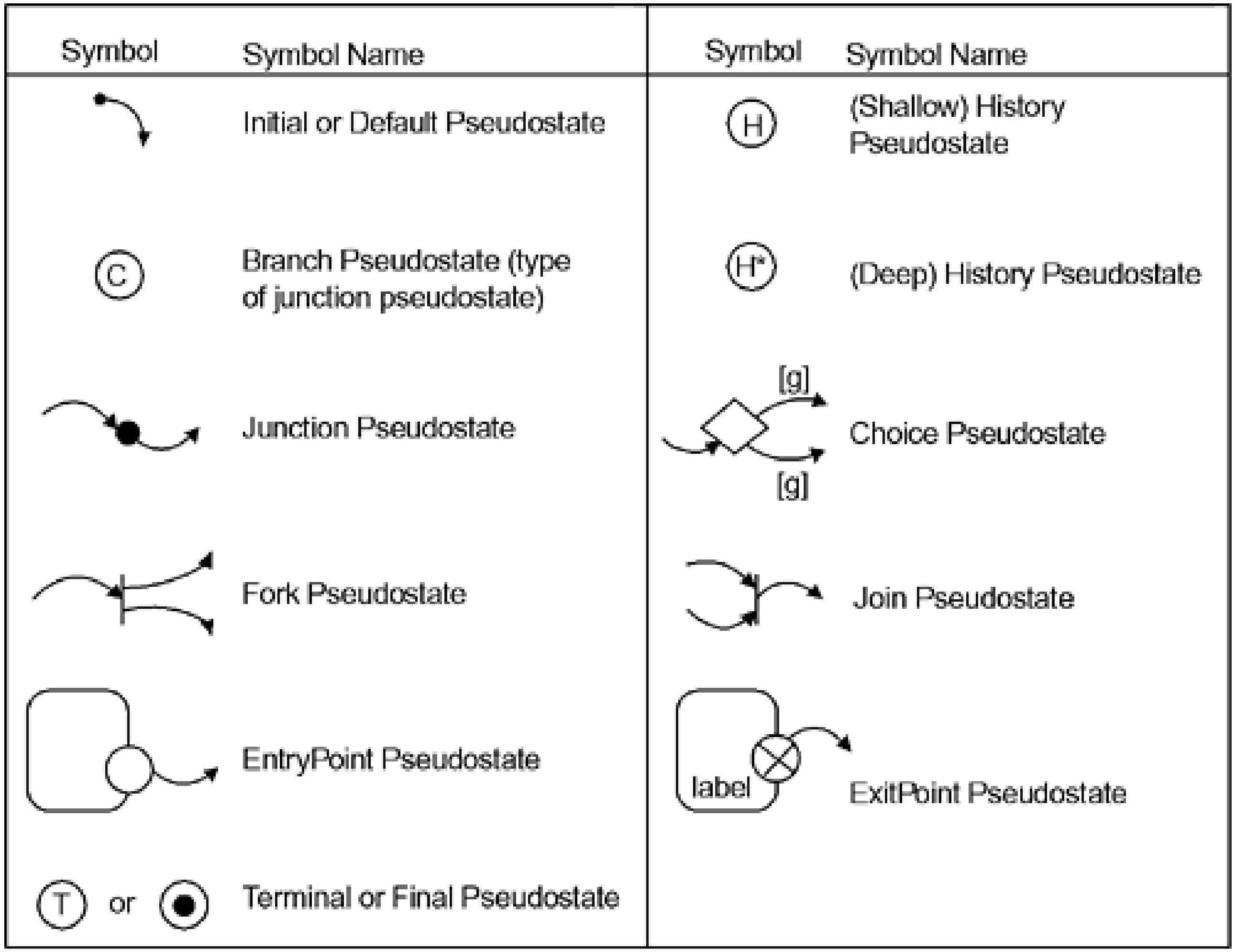
3、Pick up transition actions

4、Off Hook entry actions

5、Connected entry actions

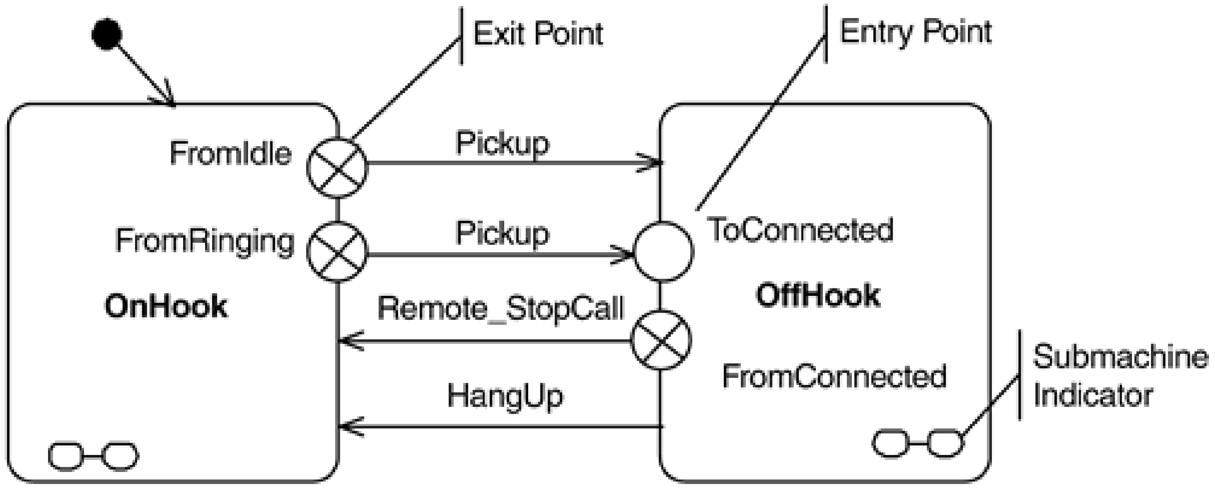


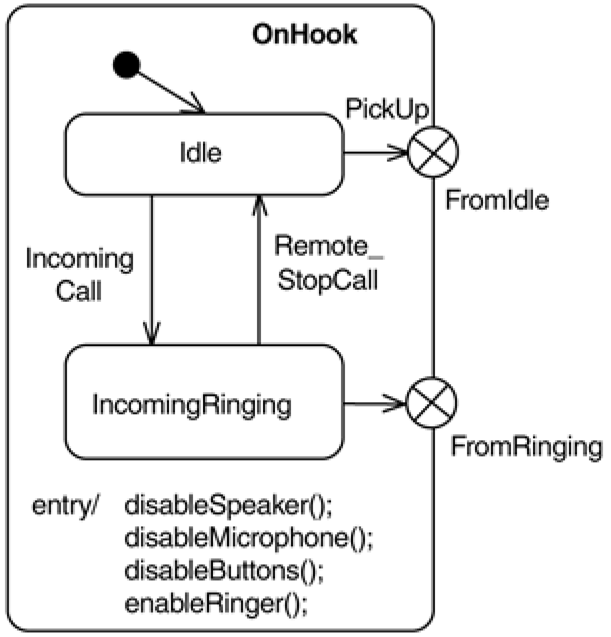
Pseudostates

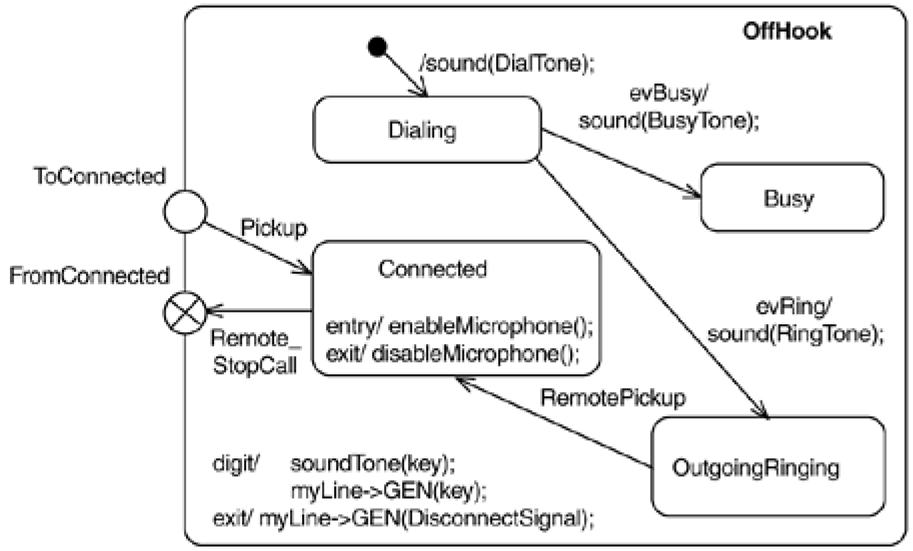


Symbol	Symbol Name	Symbol	Symbol Name
	Initial or Default Pseudostate	H	(Shallow) History Pseudostate
C	Branch Pseudostate (type of junction pseudostate)	H*	(Deep) History Pseudostate
	Junction Pseudostate	[g] [g]	Choice Pseudostate
	Fork Pseudostate		Join Pseudostate
	EntryPoint Pseudostate	label	ExitPoint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

Submachines, Entry, and Exit Points



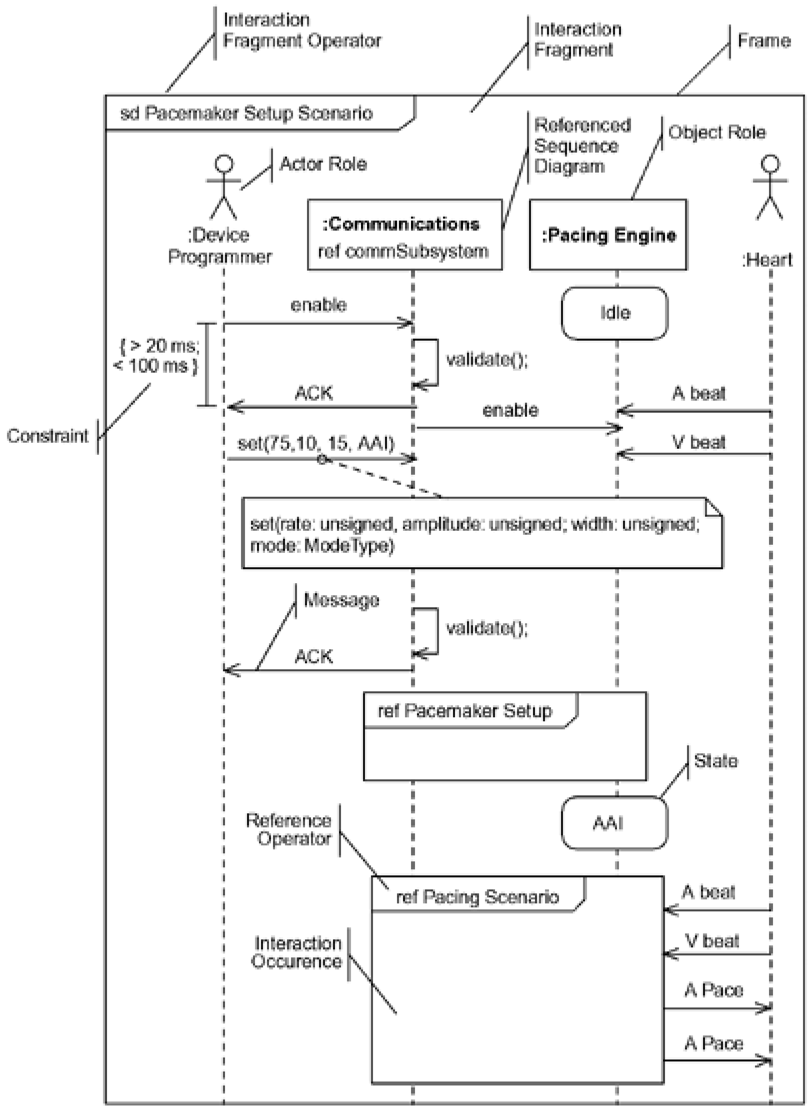


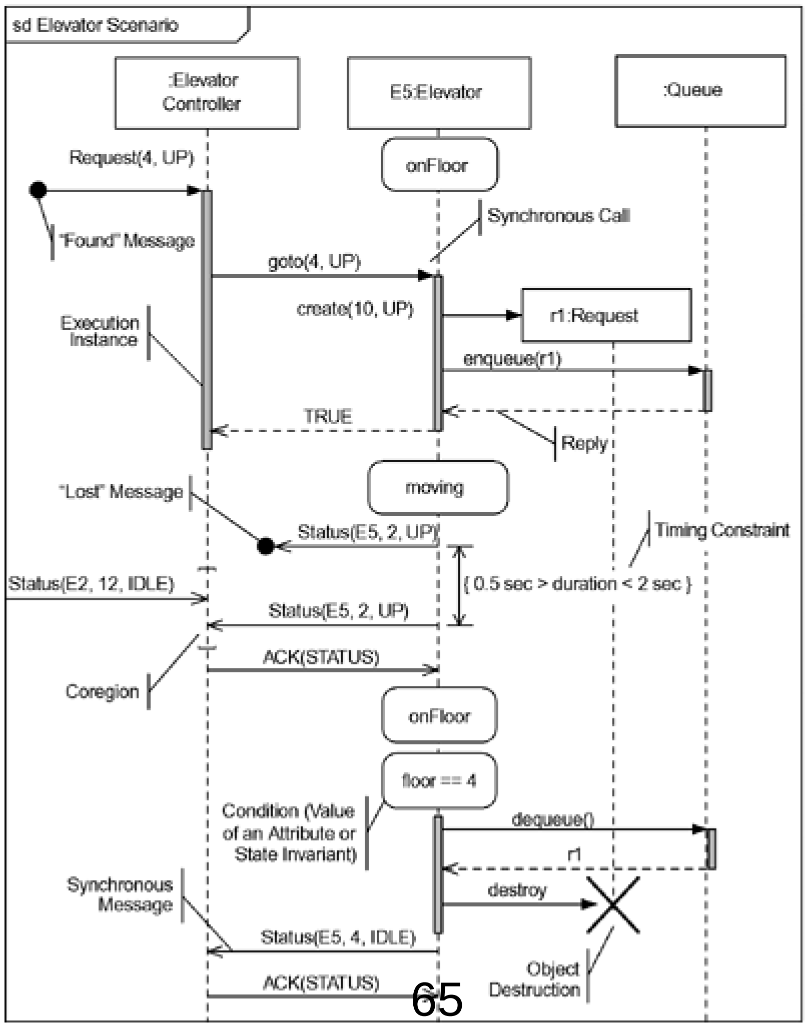


Interactions

- There are three primary diagrammatic forms in the UML for depicting interaction scenarios—
  - communication diagrams,
  - sequence diagrams
  - timing diagrams.

Sequence Diagrams





Sequence Diagrams-Partial Ordering

m0.send()->m0.receive()

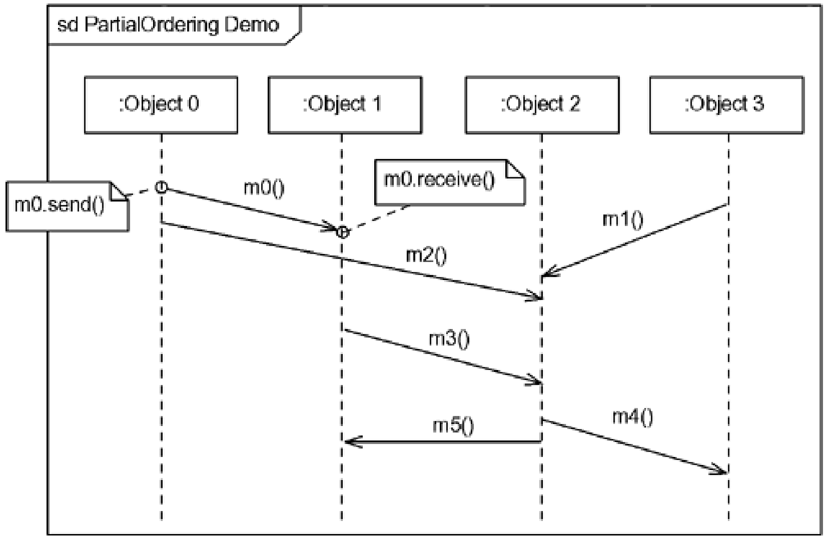
m1.send()->m1.receive()

m2.send()->m2.receive()

m3.send()->m3.receive()

m4.send()->m4.receive()

m5.send()->m5.receive()

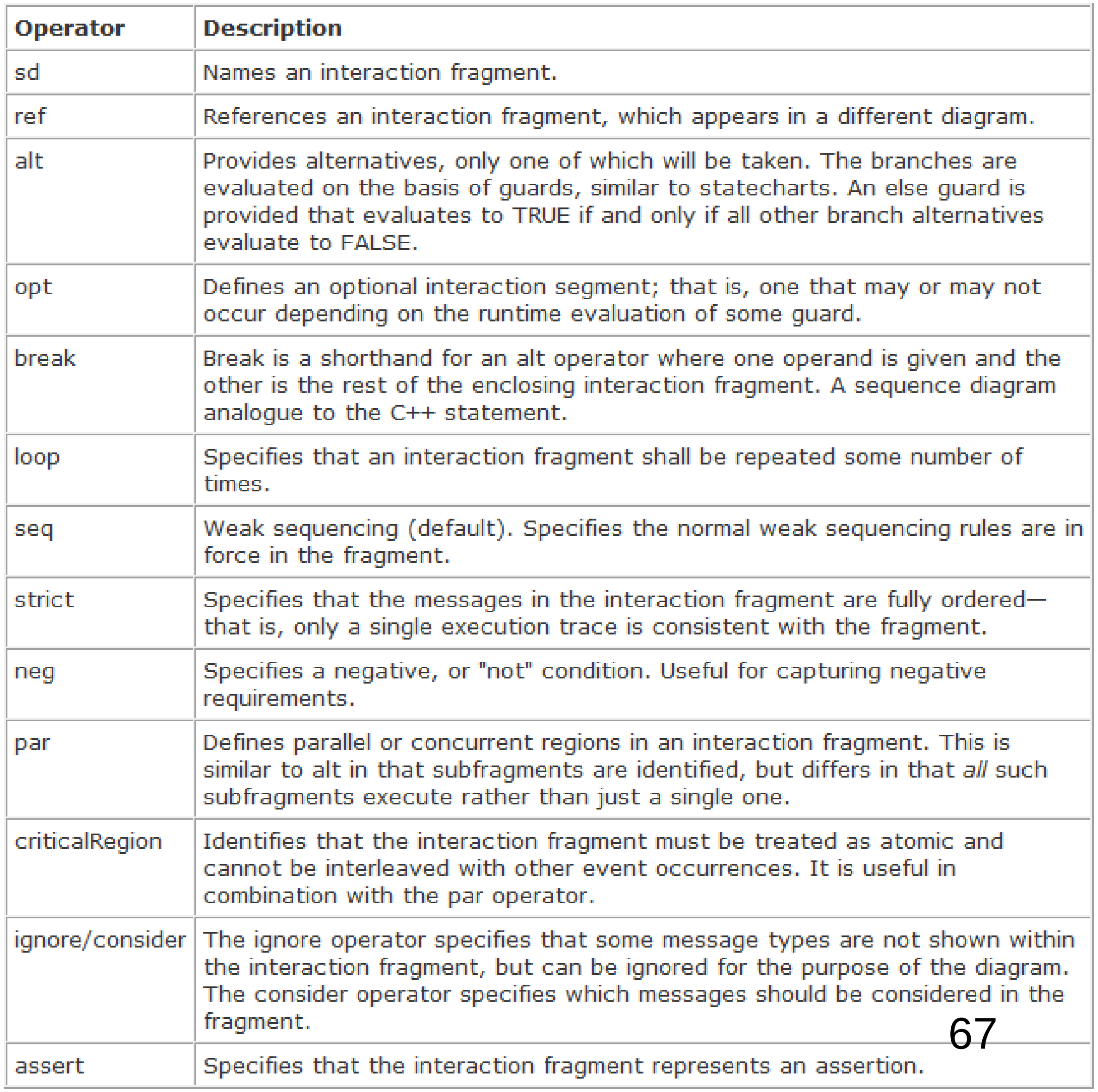


m1.receive->m2.receive()->m3.receive()->m4.send()->m5.send()

m0.receive()->m3.receive()->m5.receive()

If m2.receive() arrives prior to m0.receive(), this is called message *overtaking*.

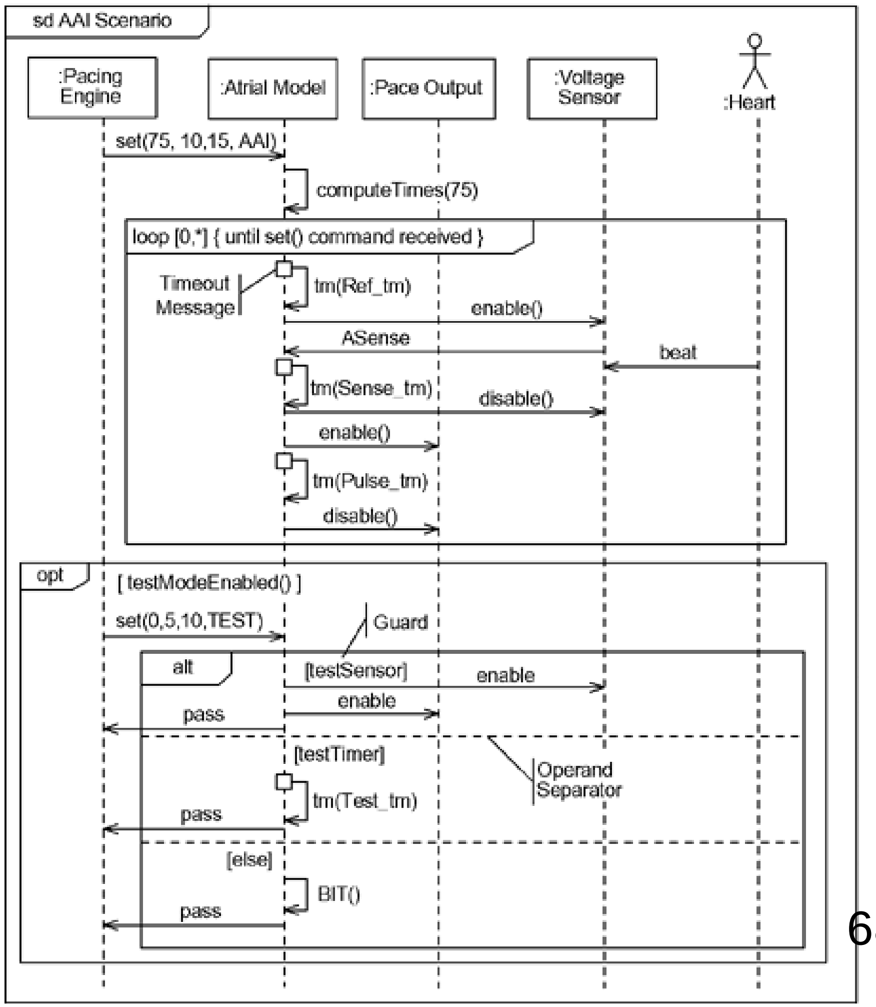
Sequence Diagrams-Partial Ordering



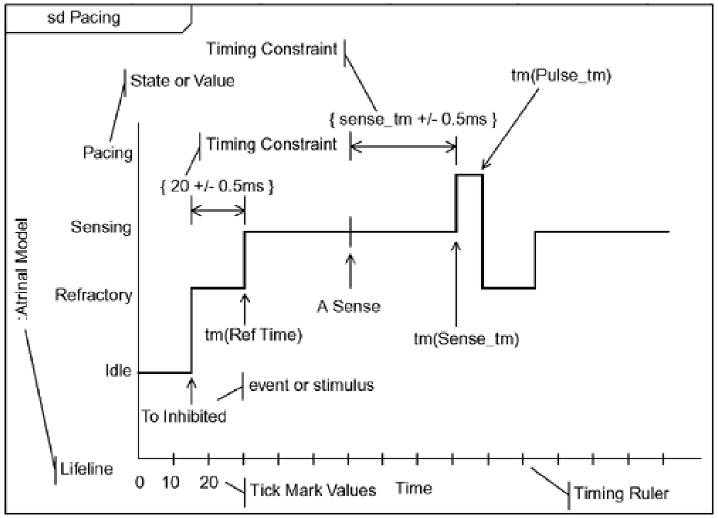
Operator	Description
sd	Names an interaction fragment.
ref	References an interaction fragment, which appears in a different diagram.
alt	Provides alternatives, only one of which will be taken. The branches are evaluated on the basis of guards, similar to statecharts. An else guard is provided that evaluates to TRUE if and only if all other branch alternatives evaluate to FALSE.
opt	Defines an optional interaction segment; that is, one that may or may not occur depending on the runtime evaluation of some guard.
break	Break is a shorthand for an alt operator where one operand is given and the other is the rest of the enclosing interaction fragment. A sequence diagram analogue to the C++ statement.
loop	Specifies that an interaction fragment shall be repeated some number of times.
seq	Weak sequencing (default). Specifies the normal weak sequencing rules are in force in the fragment.
strict	Specifies that the messages in the interaction fragment are fully ordered—that is, only a single execution trace is consistent with the fragment.
neg	Specifies a negative, or "not" condition. Useful for capturing negative requirements.
par	Defines parallel or concurrent regions in an interaction fragment. This is similar to alt in that subfragments are identified, but differs in that *all* such subfragments execute rather than just a single one.
criticalRegion	Identifies that the interaction fragment must be treated as atomic and cannot be interleaved with other event occurrences. It is useful in combination with the par operator.
ignore/consider	The ignore operator specifies that some message types are not shown within the interaction fragment, but can be ignored for the purpose of the diagram. The consider operator specifies which messages should be considered in the fragment.
assert	Specifies that the interaction fragment represents an assertion.

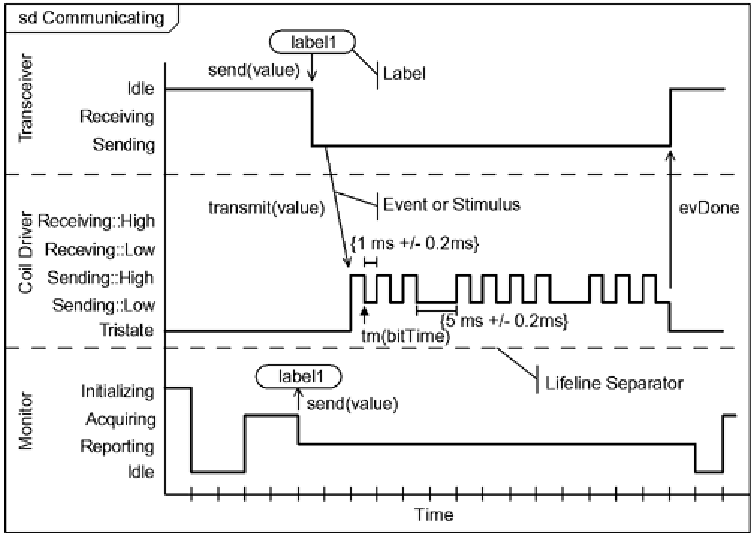
Sequence Diagrams-Alternatives, Branches, Options, and Loops

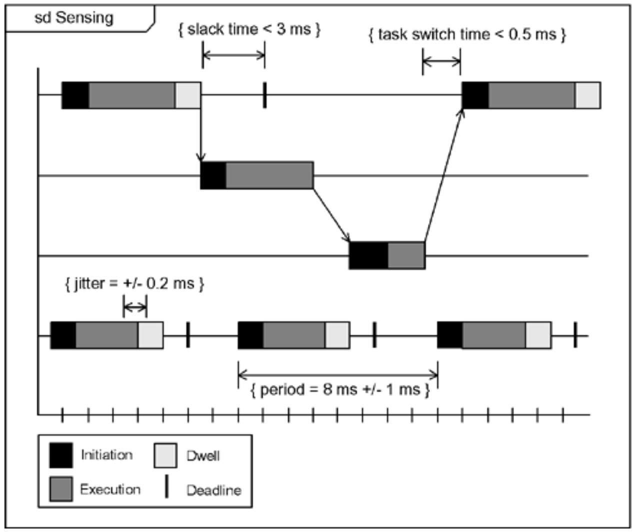
- There are several operators devoted to branching: alt, break, opt, and loop.

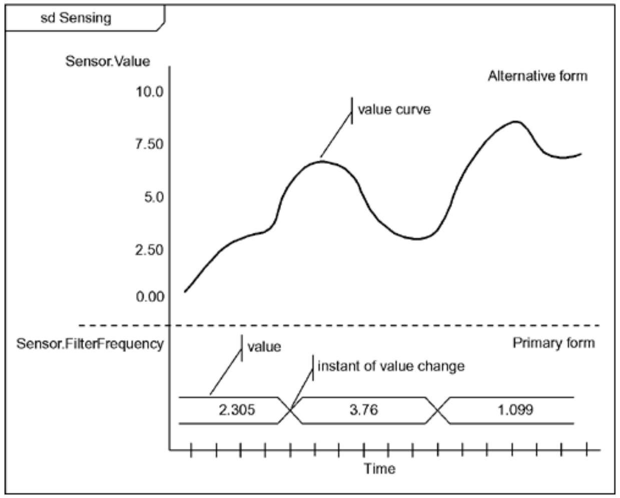


Timing Diagrams









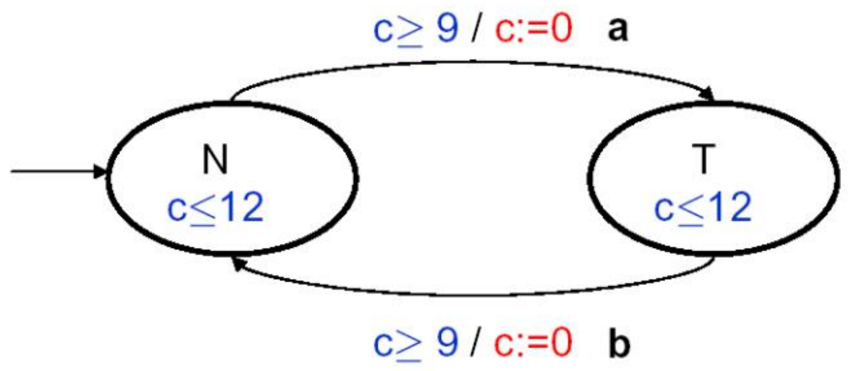
Timed automata

- **Timed automata** is a theory for modeling and verification of real time systems. Examples of other formalisms with the same purpose, are *timed Petri Nets*, *timed process algebras*, and *real time logics*.
- several model checkers have been developed with timed automata being the core of their input languages e.g.. It is fair to say that they have been the driving force for the application and development of the theory.

Timed automata

- A timed automaton is essentially a finite automaton (that is a graph containing a finite set of nodes or locations and a finite set of labeled edges) extended with real-valued variables(Timed Systems).
  - The **variables** model the logical clocks in the system, that are initialized with zero when the system is started, and then increase synchronously with the same rate.
  - **Clock constraints** i.e. guards on edges are used to restrict the behavior of the automaton.
  - A **transition** represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge. Clocks may be reset to zero when a transition is taken.

Timed automata

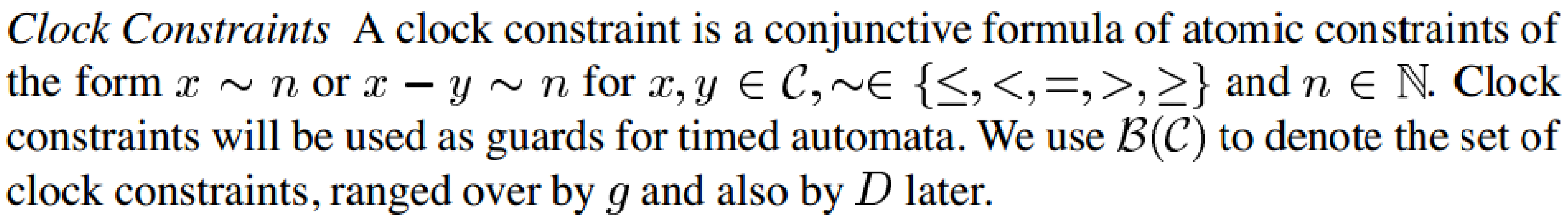


Locations N, T
Initial location N
Actions a,b
Clock c

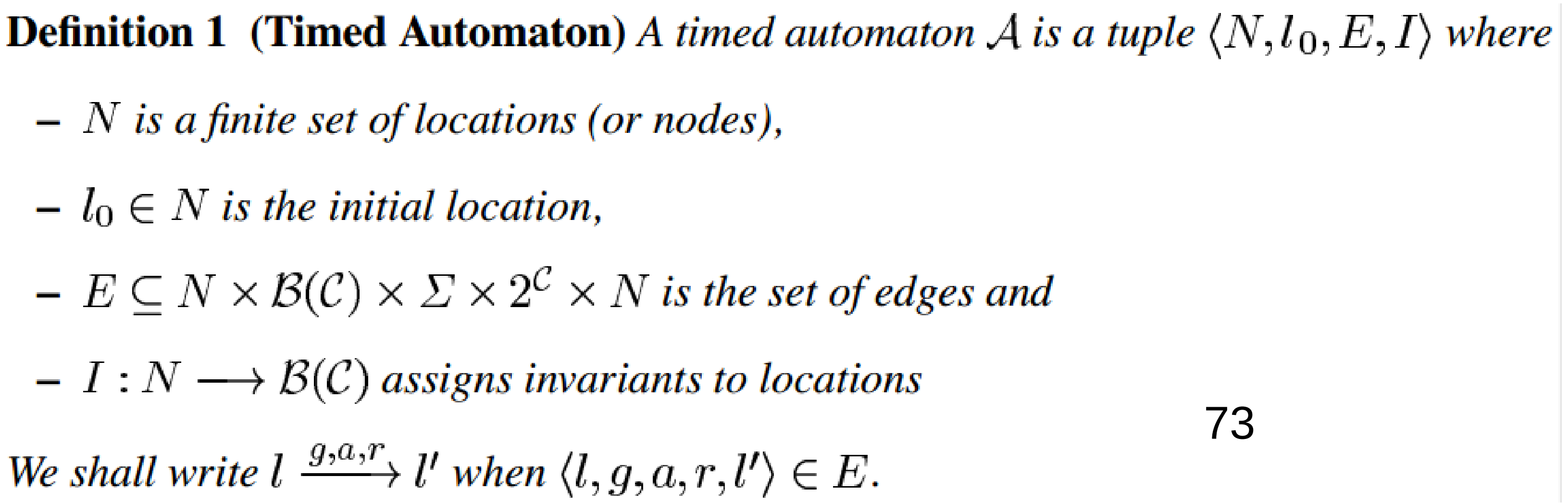
Guard c≥9
Invariant c≤12
Clock reset c:=0

Timed automata-Formal Syntax

- Assume a finite set of real-valued variables ranged over by etc. standing for clocks and a finite alphab Σ ranged over by etc.standing for actions.



Clock Constraints A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in \mathcal{C}, \sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. Clock constraints will be used as guards for timed automata. We use $\mathcal{B}(\mathcal{C})$ to denote the set of clock constraints, ranged over by g and also by D later.



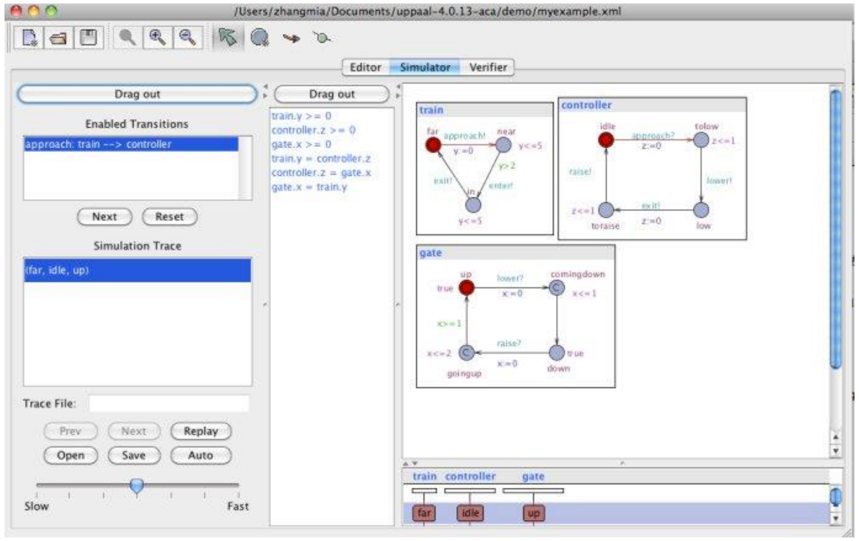
Definition 1 (Timed Automaton) *A timed automaton $\mathcal{A}$ is a tuple $\langle N, l_0, E, I\rangle$ where*

- *N is a finite set of locations (or nodes),*
- *$l_0 \in N$ is the initial location,*
- *$E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ is the set of edges and*
- *$I : N \longrightarrow \mathcal{B}(\mathcal{C})$ assigns invariants to locations*

We shall write $l \xrightarrow{g,a,r} l'$ when $\langle l, g, a, r, l'\rangle \in E$.

Timed automata-tools-Uppaal-Modeling with UPPAAL

Networks of Timed Automata A *network of timed automata* is the parallel composition $A_1|\cdots|A_n$ of a set of timed automata $A_1,\ldots,A_n$, called processes, combined into a single system by the CCS parallel composition operator with all external actions hidden. Synchronous communication between the processes is by hand-shake synchronization using input and output actions; asynchronous communication is by shared variables as described later. To model hand-shake synchronization, the action alphabet Σ is assumed to consist of symbols for input actions denoted $a?$, output actions denoted $a!$, and internal actions represented by the distinct symbol τ.



approach: train --> controller

(far, idle, up)

```
train.y >= 0
controller.z >= 0
gate.x >= 0
train.y = controller.z
controller.z = gate.x
gate.x = train.y
```

Timed automata-tools-Uppaal-Modeling with UPPAAL

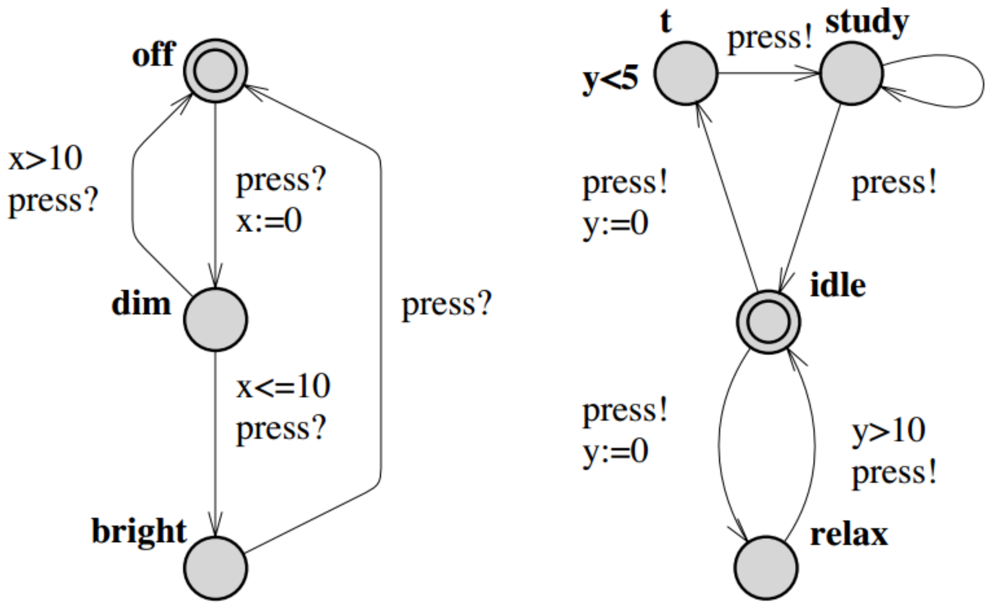


Fig. 14. Network of Timed Automata

Timed automata-tools-Uppaal-Modeling with UPPAAL

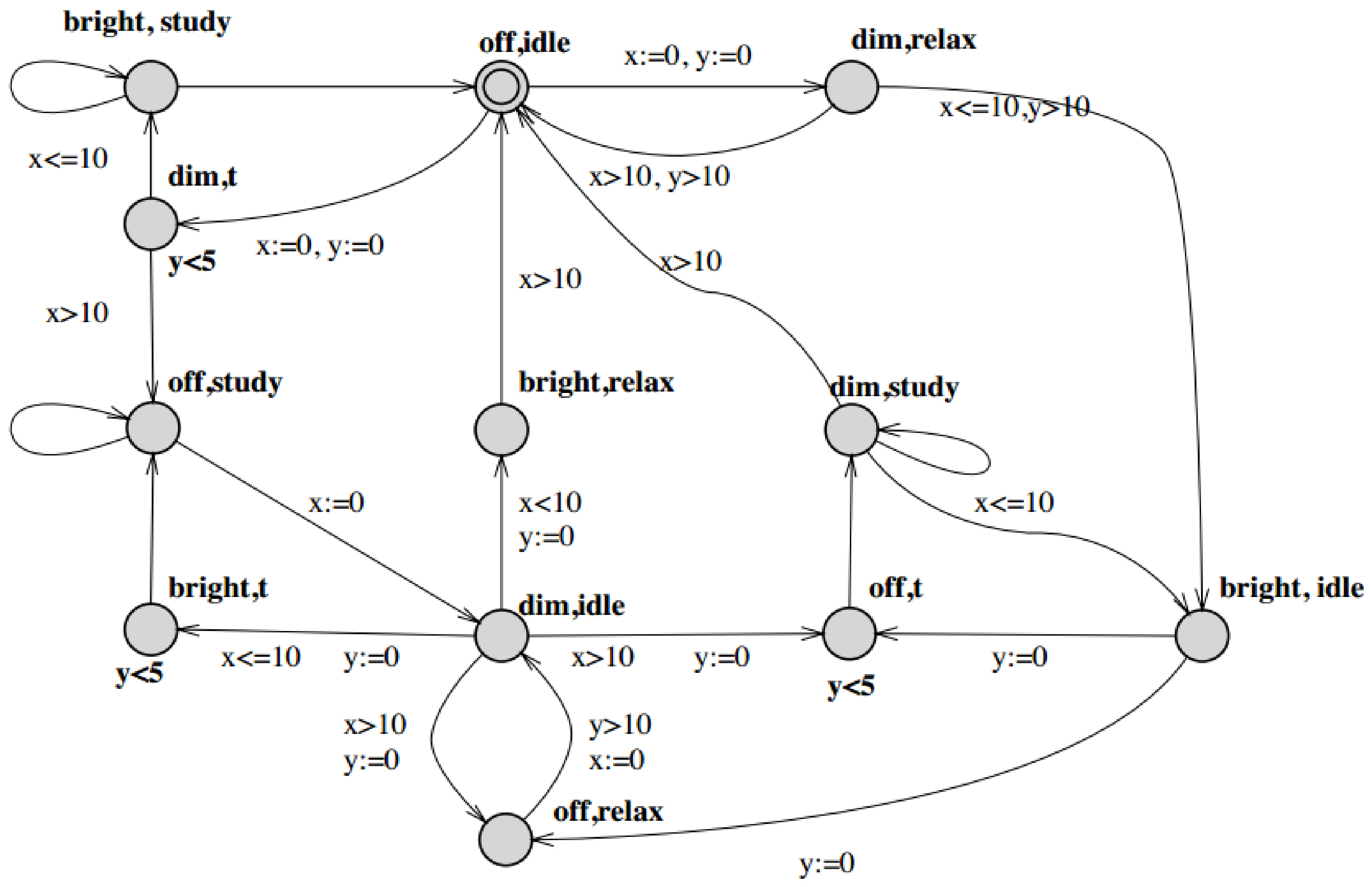
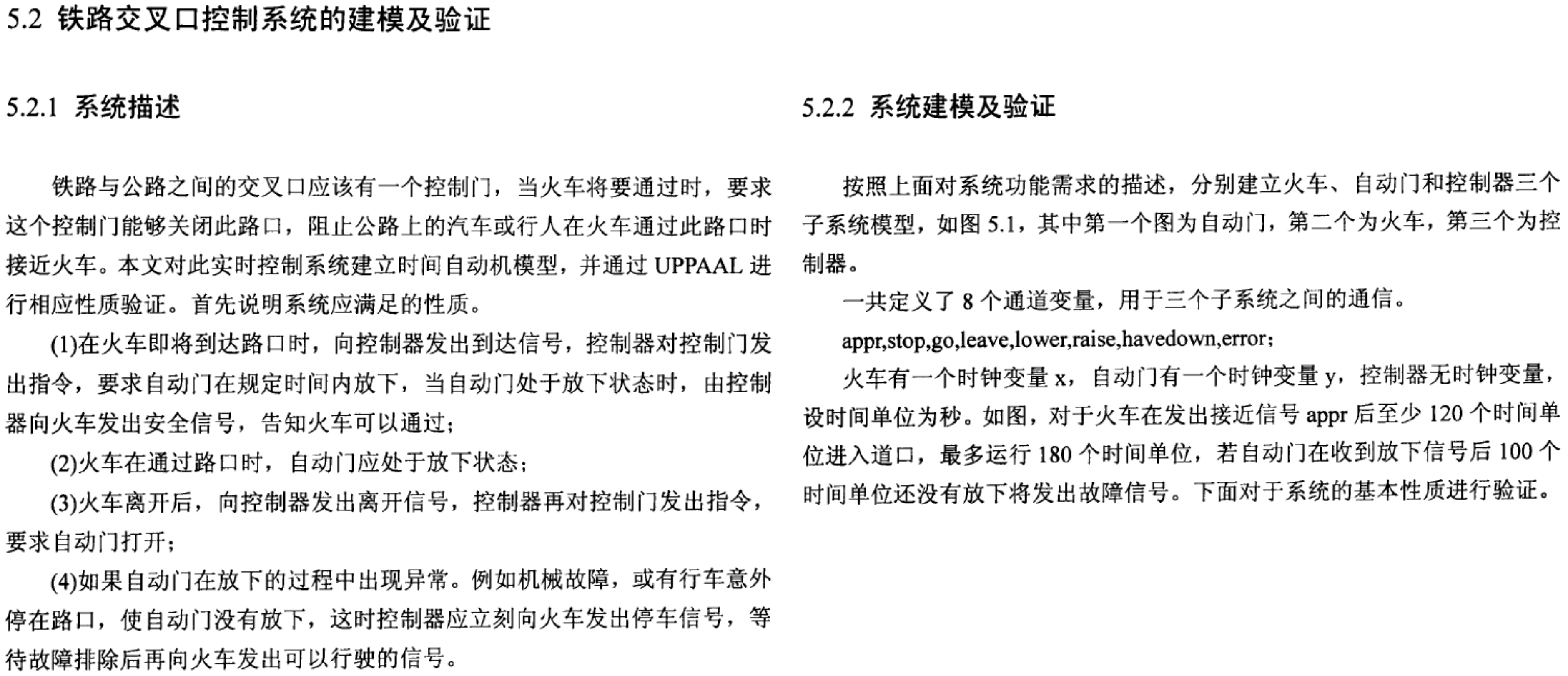


Fig. 15. Product Automaton for the Network in Fig. 14

Timed automata-Example



5.2 铁路交叉口控制系统的建模及验证

5.2.1 系统描述

铁路与公路之间的交叉口应该有一个控制门，当火车将要通过时，要求这个控制门能够关闭此路口，阻止公路上的汽车或行人在火车通过此路口时接近火车。本文对此实时控制系统建立时间自动机模型，并通过 UPPAAL 进行相应性质验证。首先说明系统应满足的性质。

(1)在火车即将到达路口时，向控制器发出到达信号，控制器对控制门发出指令，要求自动门在规定时间内放下，当自动门处于放下状态时，由控制器向火车发出安全信号，告知火车可以通过；

(2)火车在通过路口时，自动门应处于放下状态；

(3)火车离开后，向控制器发出离开信号，控制器再对控制门发出指令，要求自动门打开；

(4)如果自动门在放下的过程中出现异常。例如机械故障，或有行车意外停在路口，使自动门没有放下，这时控制器应立刻向火车发出停车信号，等待故障排除后再向火车发出可以行驶的信号。

5.2.2 系统建模及验证

按照上面对系统功能需求的描述，分别建立火车、自动门和控制器三个子系统模型，如图 5.1，其中第一个图为自动门，第二个为火车，第三个为控制器。

一共定义了 8 个通道变量，用于三个子系统之间的通信。

appr,stop,go,leave,lower,raise,havedown,error；

火车有一个时钟变量 x，自动门有一个时钟变量 y，控制器无时钟变量，设时间单位为秒。如图，对于火车在发出接近信号 appr 后至少 120 个时间单位进入道口，最多运行 180 个时间单位，若自动门在收到放下信号后 100 个时间单位还没有放下将发出故障信号。下面对于系统的基本性质进行验证。

Timed automata-Example

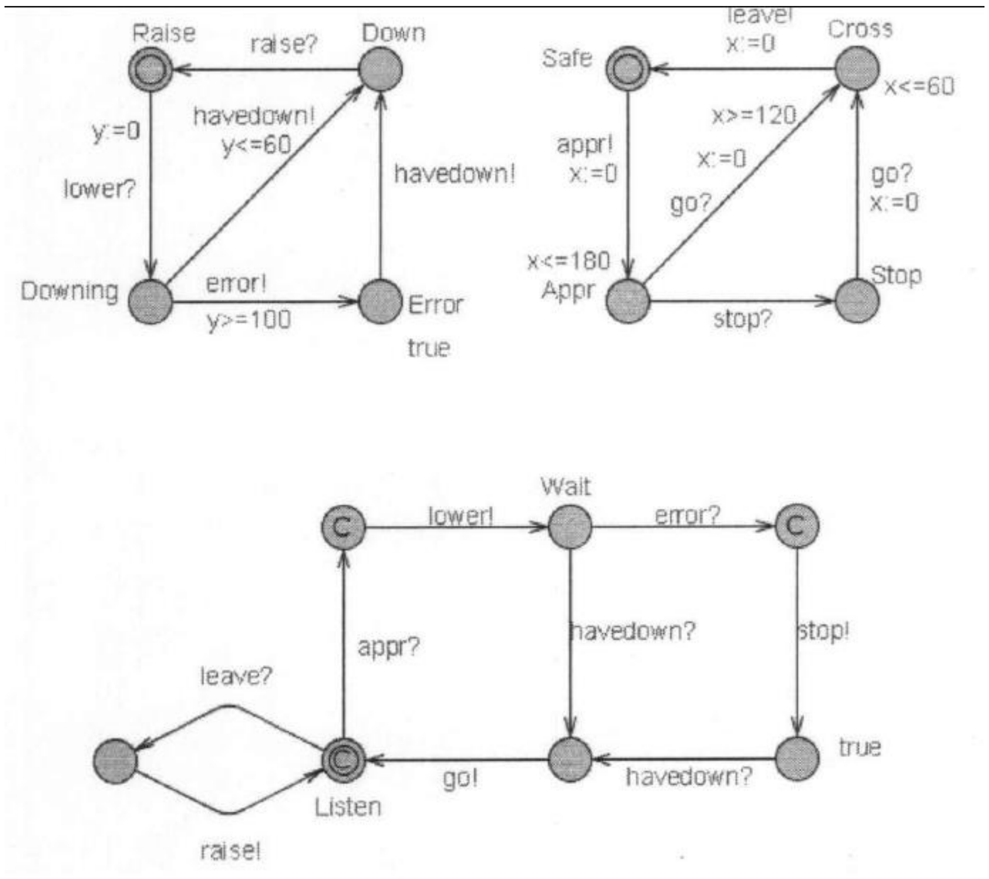
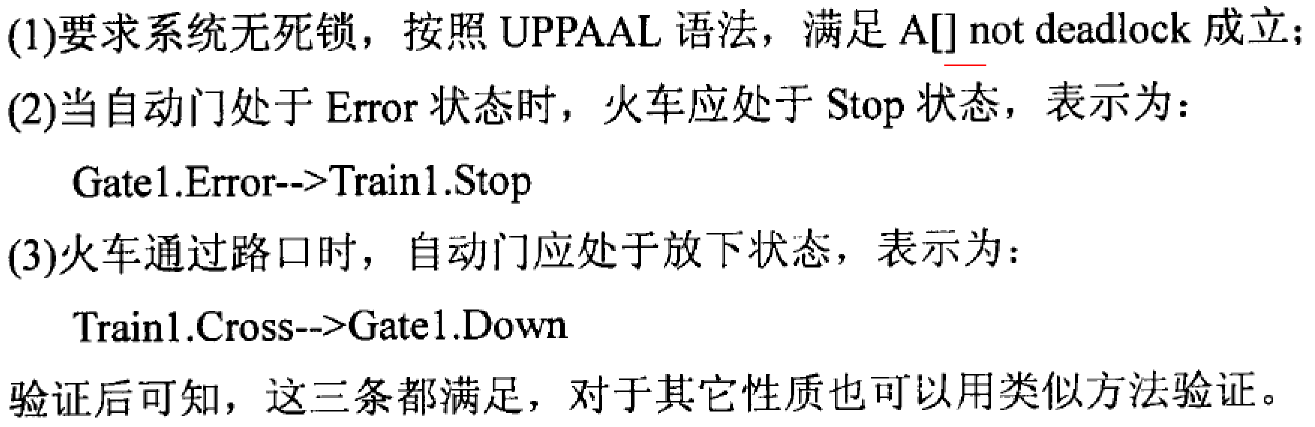


图 5.1 火车、自动门和控制器的时间自动机模型

Timed automata-Example


(1)要求系统无死锁，按照 UPPAAL 语法，满足 A[] not deadlock 成立；

(2)当自动门处于 Error 状态时，火车应处于 Stop 状态，表示为：

Gate1.Error-->Train1.Stop

(3)火车通过路口时，自动门应处于放下状态，表示为：

Train1.Cross-->Gate1.Down

验证后可知，这三条都满足，对于其它性质也可以用类似方法验证。

UML Profile

- A UML profile is a facility for augmenting UML models with supplementary information. This mechanism can be used in either of two ways:
- 1. *It can be used to extend the UML language*. For example, UML does not provide an explicit semaphore concept, but it is possible to add it by overloading an existing UML concept, such as Class. The result is a special kind of Class that, in addition to its standard Class semantics, incorporates semaphore semantics.
- 2. *It can be used to attach additional information to models needed for ancillary purposes such as model analyses or code generation.* Such an annotation can be used, for instance, to specify the worst-case execution time of some operation of a class, which might be needed for analyzing the timing characteristics of an application.

UML Profile

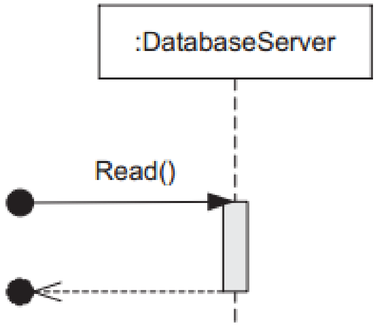


FIGURE 2.1

An unadorned UML sequence diagram fragment showing a database read request.

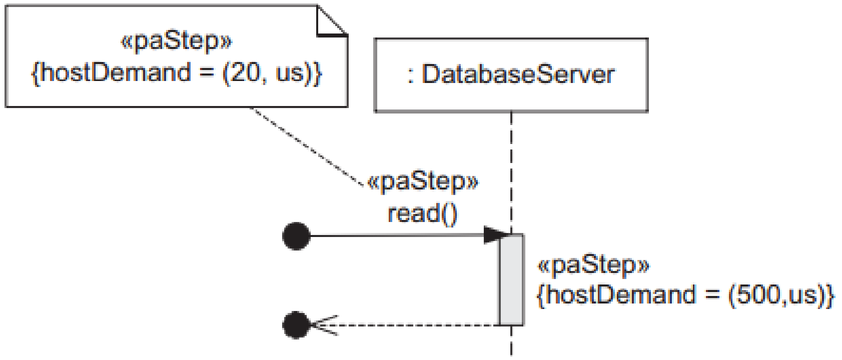


FIGURE 2.2

The annotated version of the model in Figure 2.1 with performance data included.

In this particular example, we can conclude based on the annotations that the CPU time required to execute this behavior fragment is equal to (20 microseconds + 500 microseconds) = 520 microseconds.

UML MARTE

www.omgmarte.org

MARTE Tutorial

An OMG standard:
UML profile to develop Real-Time systems

How to read this tutorial

- **Within slides, we may shown models at different levels of abstraction. We will clarify each level through following pictograms**
  - For Domain View level — Domain
  - For UML Profile View Level — UML
  - For User Model View Level — User

www.omgmarte.org

Agend

- **Part 1**
  - **Introduction to MDD for RT/E systems & MARTE in a nutshell**
- **Part 2**
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- **Part 3**
  - The timing model
- **Part 4**
  - A component model for RT/E
- **Part 5**
  - Platform modeling
- **Part 6**
  - Repetitive structure modeling
- **Part 7**
  - Model-based analysis for RT/E
- **Part 8**
  - MARTE and AADL
- **Part 9**
  - Conclusions

www.omgmarte.org

A Bit of Modern Software…

```
SC_MODULE(producer) {
  sc_outmaster<int> out1;
  sc_in<bool> start; // kick-start
  void generate_data () {
    for(int i =0; i <10; i++) {
      out1 =i ; //to invoke slave;
    }
  }
  SC_CTOR(producer) {
    SC_METHOD(generate_data);
    sensitive << start;
  }
};
SC_MODULE(top) { // container
  producer *A1;
  consumer *B1;
  sc_link_mp<int> link1;
  SC_CTOR(top) {
    A1 = new producer("A1");
    A1.out1(link1);
    B1 = new consumer("B1");
    B1.in1(link1);
}
;
```

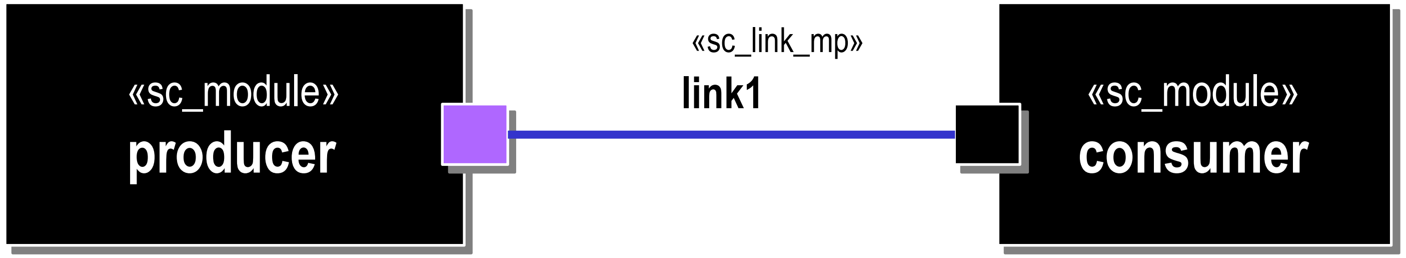
```
SC_MODULE(consumer) {
  sc_inslave<int> in1;
  int sum; // state variable
  void accumulate (){
    sum += in1;
    cout << "Sum = " << sum << endl;
  }
  SC_CTOR(consumer) {
    SC_SLAVE(accumulate, in1);
    sum = 0; // initialize
  }
};
```

Can you spot the architecture?

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

www.omgmarte.org

…and its UML Model

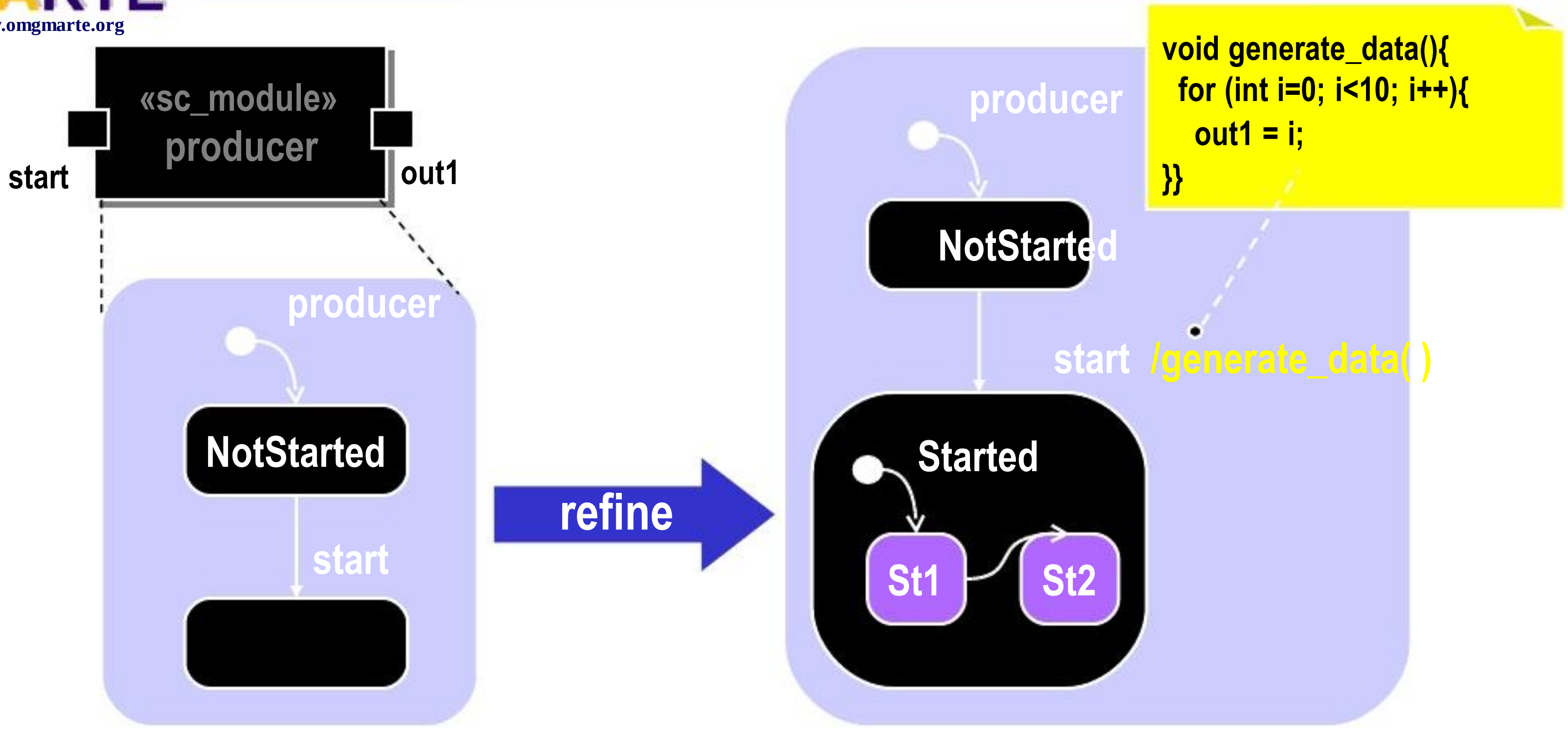


Can you spot the architecture?

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

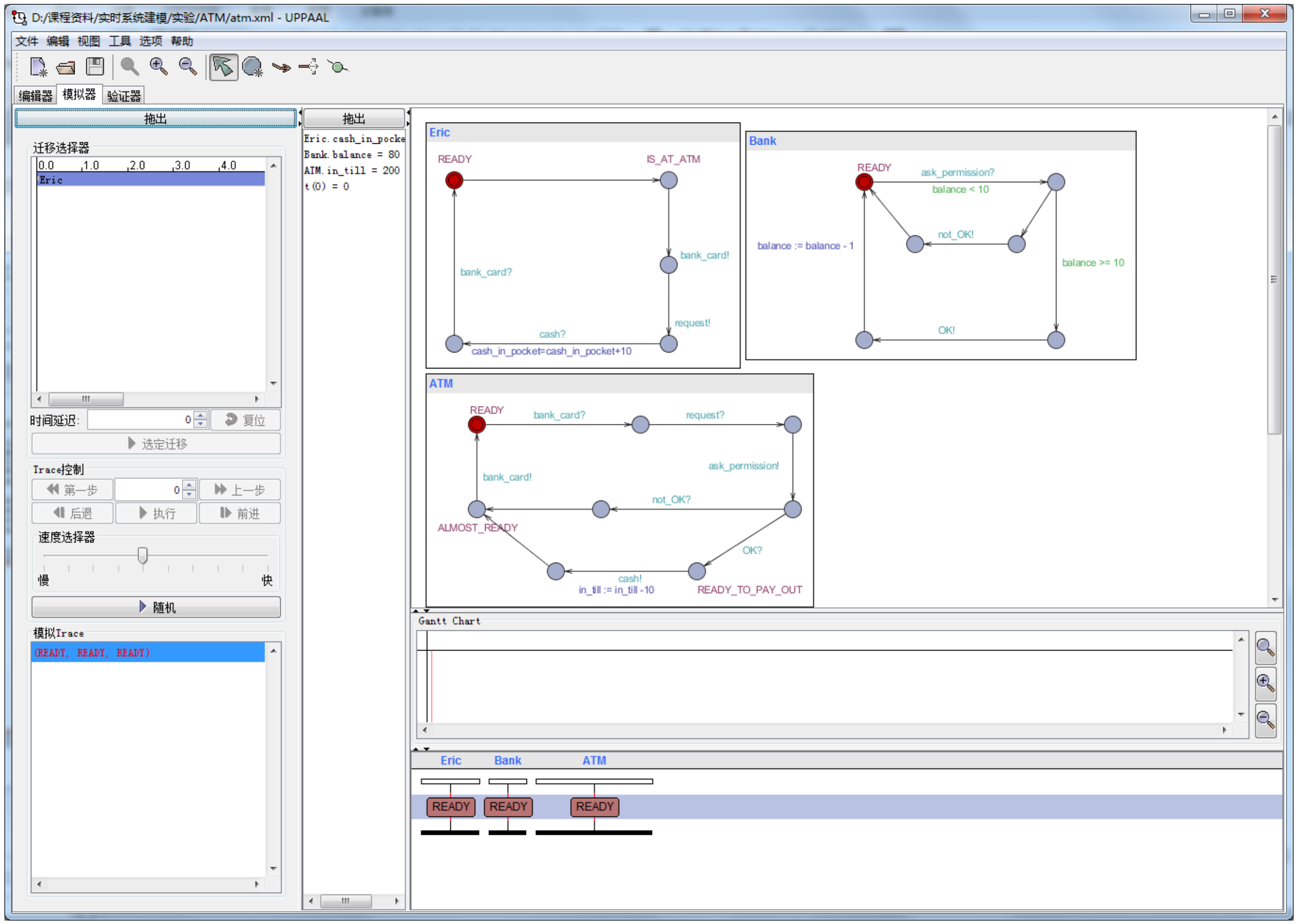
www.omgmarte.org

Model Evolution: Refinement



```
void generate_data(){
  for (int i=0; i<10; i++){
    out1 = i;
}}
```

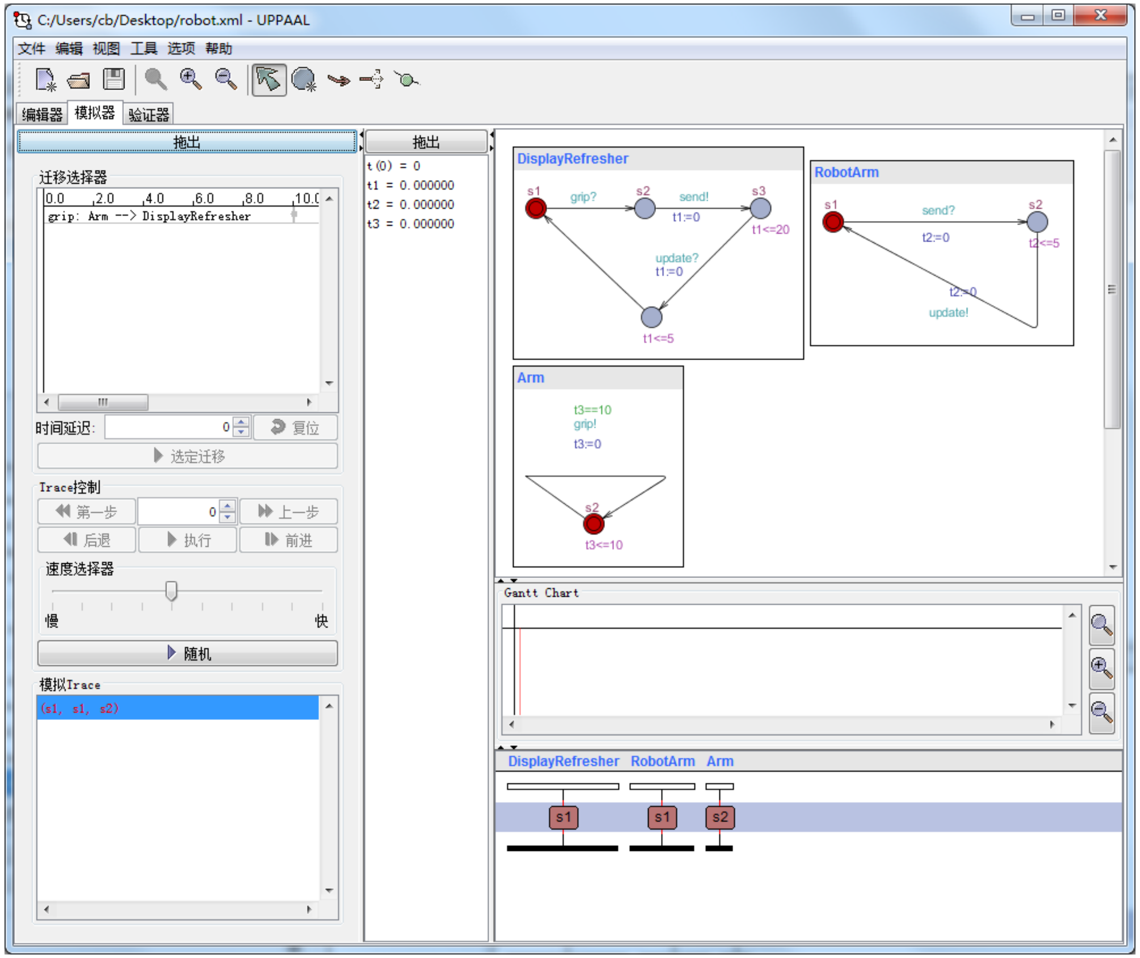
exp-1



```
Eric.cash_in_pocke
Bank.balance = 80
ATM.in_till = 200
t(0) = 0
```

```
(READY, READY, READY)
```

exp-2

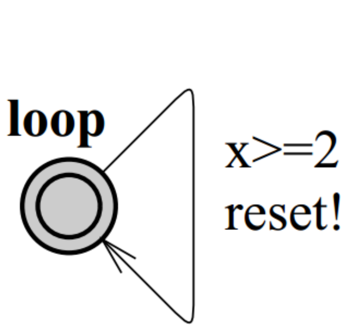

C:/Users/cb/Desktop/robot.xml - UPPAAL
文件 编辑 视图 工具 选项 帮助
编辑器 模拟器 验证器
拖出
迁移选择器
0.0 2.0 4.0 6.0 8.0 10.0
时间延迟: 0
复位
选定迁移
Trace控制
第一步 上一步 后退 执行 前进
速度选择器
慢 快
随机
模拟Trace
DisplayRefresher
s1 grip? s2 send! t1:=0 s3 t1<=20 update? t1:=0 t1<=5
RobotArm
s1 send? s2 t2:=0 t2<=5 t2>0 update!
Arm
t3==10 grip! t3:=0 s2 t3<=10
Gantt Chart
DisplayRefresher RobotArm Arm
s1 s1 s2

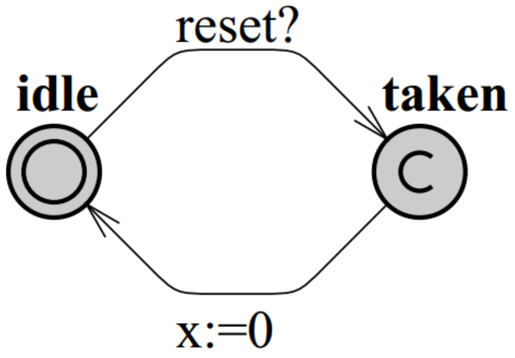
```
grip: Arm --> DisplayRefresher
```

```
t(0) = 0
t1 = 0.000000
t2 = 0.000000
t3 = 0.000000
```

```
(s1, s1, s2)
```

exp-3





(a) Test. || (b) Observer.