八道大题，两个简答
描述一下什么是模型驱动开发
CPS，实时系统，MDA(CIM_PIM_PSM 例子重点)
RM/EDF 调度机制，选其一
用例图，类图，时序图（画用例图，其中某个功能的时序图）
自动机的事件，状态迁移等内容理解，描述系统的执行过程
各次实验的理解（三选二），考察查询语言
UML 扩展语言 MARTE 描述，与 UML 的区别
模型驱动开发有哪些基本的东西

去年考试回忆：

1、CPS 系统概述：

Cyber（信息技术）：**计算，通信，控制（3C 技术,computation, communication, control ）**，并且离散，逻辑，交换的系统；

Physical（物理系统）： 一些受物理定律的支配，并可持续运行的自然或人工系统；

***Cyber-Physical Systems*（信息物理融合系统）：网络和物理系统紧密结合在一起的系统。是一个结合计算、网络和物理环境的多位复杂系统，通过 3C 技术的有机融合与深度协作，实现大型工程系统的实时感知，动态控制和信息服务。**

CPS 系统的特点：

1. **信息-物理的耦合由新的需求和应用驱动**
   {
   网络工作在多个和极端的规模下；
   每个物理组建中的网络性能；
   大规模的有线和无线网络
   }
   系统的系统
2. **新的时空约束（New patial-temporal constraints）**
   {
   多个时间和空间尺度上的复杂性；
   动态重组/重新匹配；
   非传统法的计算和物理基块

}

3. **无处不在的安全和隐私需求；**
4. **操作必须可靠**

2、名词解释：

及时性（timeless）：行为的及时性与时间约束有关，如截止时间。最后期限可能很硬或很软。时效性的重要建模问题是建模执行时间，截止时间，到达模式，同步模式和时间源。

并发性（concurrency）：多个操作顺序链的同时执行。围绕并发系统执行的问题与此有关：

a.并发线程的调度特性
b.传入事件的到达模式
c.线程必须同步时使用集合点模式
d.控制对共享资源访问的方法

可预测性（predictability）：实现系统的时候，能够预知未来执行到哪里。

正确性（correctness）：正确性表明一个系统总是运行正确。

鲁棒性（robustness）：鲁棒性表明系统即使在遇到新的情况（不在计划中）下也是可靠的。因此必须警惕死锁，竞争以及其他异常情况。

3、RM、EDF 调度算法：

RM：（周期越短优先级越高）

4.2.1 调度算法描述

- 最佳的静态优先级调度
- 根据周期分配优先权
- 周期较短的任务具有较高的优先级
- 以最短的时间执行一项工作

注：

（1）RM 的最优性可以描述为：如果一个任务集能够被静态调度，那么 RM 调度算法就能够调度这个任务集。

其最最优性可以证明，证明方法类似于动态规划中的“剪贴法”。

（2）RM 调度算法属于“可抢占”式调度一类（事实上，后面的 EDF 也是这一类）。

EDF：（截止时间越早，优先级越高）

4.3 EDF（最早截止时间优先，Earliest Deadline First）调度算法

4.3.1 调度算法描述

- 最佳的动态优先级调度
- 具有较早截止时间的任务具有较高的优先级
- 以最早的截止时间执行任务

6、UML Profiles 与 UML 区别：

UML Profiles 是用补充信息扩充 UML 模型的工具。此机制可通过以下两种方式之一使用：
一。它可以用来扩展 UML 语言。例如，UML 不提供明确的信号量概念，但是可以通过重载现有的 UML 概念（如类）来添加它。结果是一种特殊的类，除了它的标准类语义之外，还包含信号量语义。
二。它可用于将附加信息附加到辅助用途 (如模型分析或代码生成) 所需的模型。例如，可以使用这样的注释来指定类的某个操作的最坏执行时间，这可能是分析应用程序计时特性所必需的。

5.9 UML 预定义包（UML profile）

一个 UML 预定义包是一个扩充具有补充信息的 UML 模型的工具。 这种机制可以以两种方式使用：

- 用以扩展 UML 语言。

  例如，UML 没有提供明确的信号量概念，但可以通过重载现有的 UML 概念（如 Class）来添加它。

  结果是一种特殊的类，它除了标准的类语义之外，还包含信号量语义。
- 可用于将附加信息附加到辅助用途（如模型分析或代码生成）所需的模型。

  例如，可以使用这种注释来指定类的某些操作的最差情况执行时间，这可能需要用于分析应用程序的时序特性。

TimedObservation 是 TimedInstantObservation 和 TimedDurationObservation 的抽象超类。

TimedInstantObservation 表示与事件发生（eocc 属性）相关联并且在给定时钟上观察到的时刻。

UML 预定义包的优点：

- 语言基础设施的再利用（工具，规格）
- 需要较少的语言设计技能
- 允许扩展构造型采用新（图形）符号
- 配置文件可以定义模型视点

缺点：受 UML 元模型的约束

7、：

8、ATM 死锁，死锁的危害：

实验一 ATM 的死锁问题：要理解实验一的原模型是 Eric 每次取 10 块钱，在收到取钱请求后，Bank 会把 Eric 要取钱的数目与 Eric 账户的余额进行比较，如果超支，则会返回 not_Ok!信号，然后 Bank 回到起点状态。同时 ATM 会收到这个信号进行转换状态（退卡），然后 Atm 回到起点状态。但是 Eric 将会一直停留在等待 cash 信号的这个状态，无法回到起点状态，所以造成了死锁。

危害：

(1) 死锁会使进程得不到正确的结果。因为处于死锁状态的进程得不到所需的资源，不能向前推进，故得不到结果。

(2)死锁会使资源的利用率降低。因为处于死锁状态的进程不释放已占有的资源，以至于这些资源不能被其他进程利用，故系统资源利用率降低。

(3)死锁还会导致产生新的死锁。其它进程因请求不到死锁进程已占用的资源而无法向前推进，所以也会发生死锁。

w

模型驱动开发（MDD,Model-Driven Development）

参考资料

- introduction.ppt （p34~p47）
- *MDSF：模型驱动开发（MDD）介绍*
- *元模型*

模型驱动开发Model Driven Development （MDD）是一种以模型作为主要工件的高级别抽象的开发方法，模型在工具的支持下，被作为核心资产被转换成代码或者可运行配置。

2.1 模型

是指某个系统的简化/抽象表示，从给定的角度突出显示感兴趣的属性。这个观点定义了模型的关注和范围。

图2.1-1：系统层次图

2.2 MDA 定义的三种模型

- 计算独立模型（CIM，Computation-Independent Model）

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（PIM，Platform-Independent Model）

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 （PSM，Platform-Specific Model）

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。

2.3 模型驱动层次结构

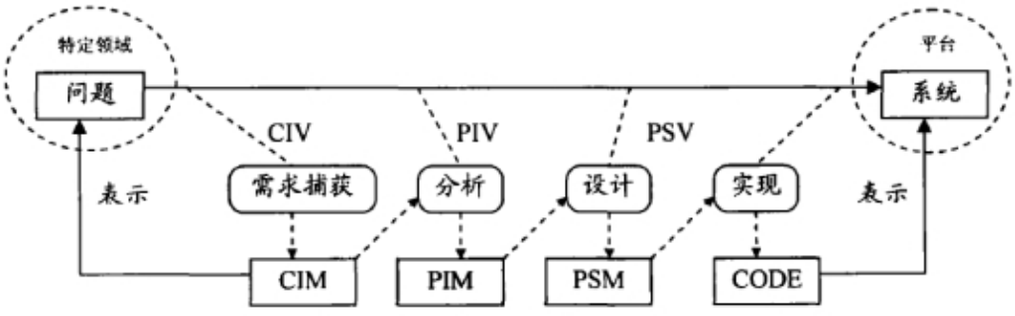


图2.3-1：模型驱动开发途径

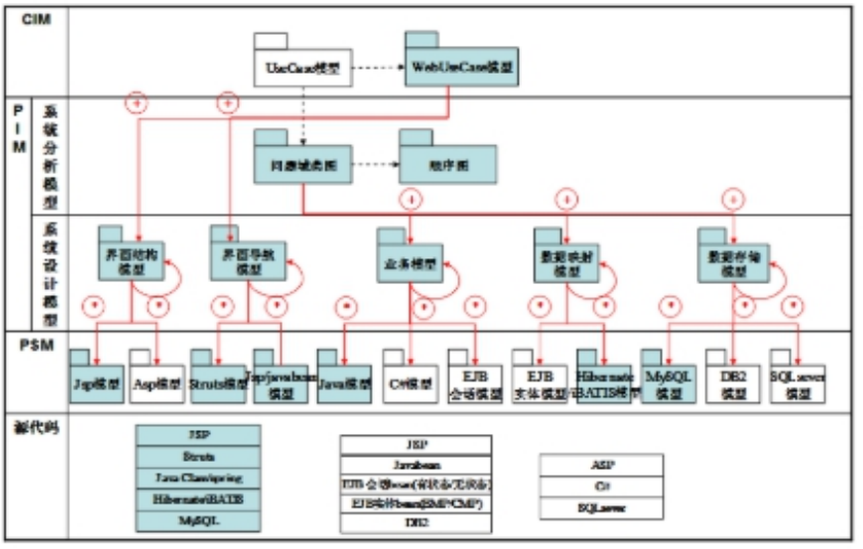


图2.3-3：模型驱动层次结构（2）

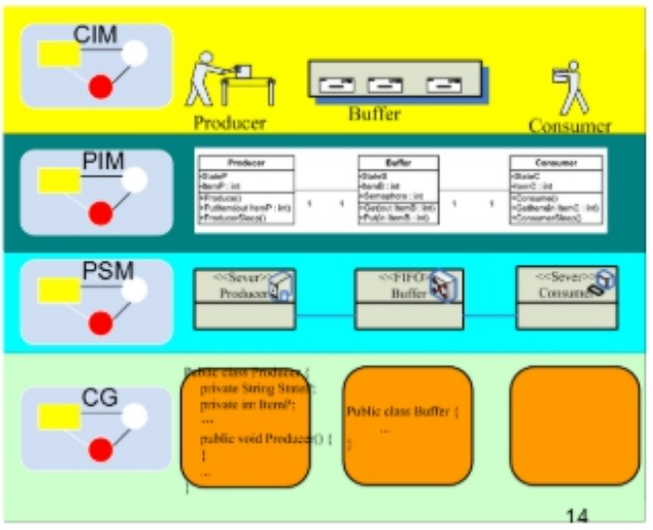


图2.3-2：模型驱动层次结构（1）

4．实时系统任务调度（Real-time workload)

本小结重点内容是实时调度算法，主要包括 RM 算法（静态优先级）和 EDF 算法（动态优先级）

参考资料：

- 2.safety-critical system.ppt （p47~p106）
- *优先级反转和优先级继承*

4.1 术语和概念

实时工作量

- 工作（Job，工作单位）：计算，文件读取，消息传输等
- 属性（Attributes）：推进需要的资源，时间参数

实时任务

任务：从读取输入数据和内部状态开始，结束生成结果并更新内部状态。

在调用点没有内部状态的任务称为无状态任务，否则称为有状态任务。

对于周期任务（p，e），任务周期性的重复。（注意任务的描述符号表示，调度算法部分会用到）

- 周期 p = inter - release time；p>0
- 执行时间 e = maximum execution time （0 < e < p)
- 利用率 U = e/p

截止时间：硬截止时间 VS 软截止时间

硬截止时间：如果错过了最后期限，可能会造成灾难性的或非常严重的后果

因此验证是至关重要的：即使在最坏的情况下，系统运行是否能够满足所有的截止时间

确定性保证

软截止时间：理想情况下，截止时间应该达到最高性能。 在最后期限未到的情况下，性能会下降。

属于尽最大努力保证实时服务一类

可调度性：

表示实时系统（一组实时任务）是否能够按期完成的属性

4.2 RM（单调速率，Rate Monotonic）调度算法

4.2.1 调度算法描述

- 最佳的静态优先级调度
- 根据周期分配优先权
- 周期较短的任务具有较高的优先级
- 以最短的时间执行一项工作

注：

（1）RM 的最优性可以描述为：如果一个任务集能够被静态调度，那么 RM 调度算法就能够调度这个任务集。

其最最优性可以证明，证明方法类似于动态规划中的“剪贴法”。

（2）RM 调度算法属于“可抢占”式调度一类（事实上，后面的 EDF 也是这一类）。

4.2.2 调度过程

响应时间：任务从释放时间到结束时间

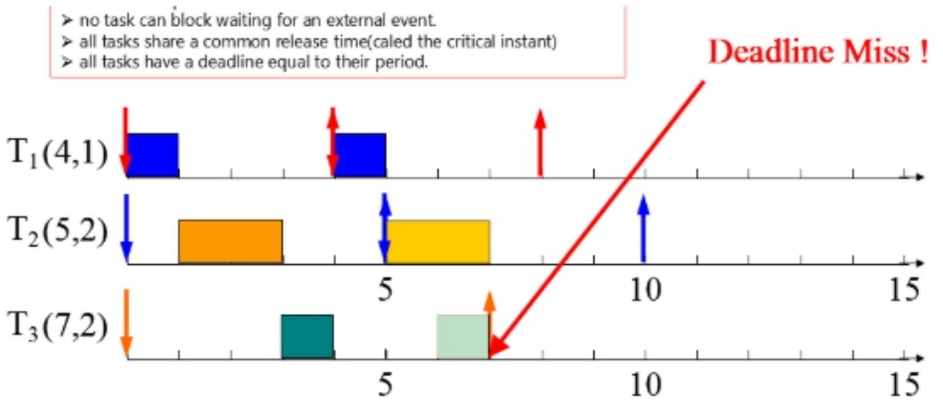


图4.2.2-1：RM调度过程示意图（dealine miss）

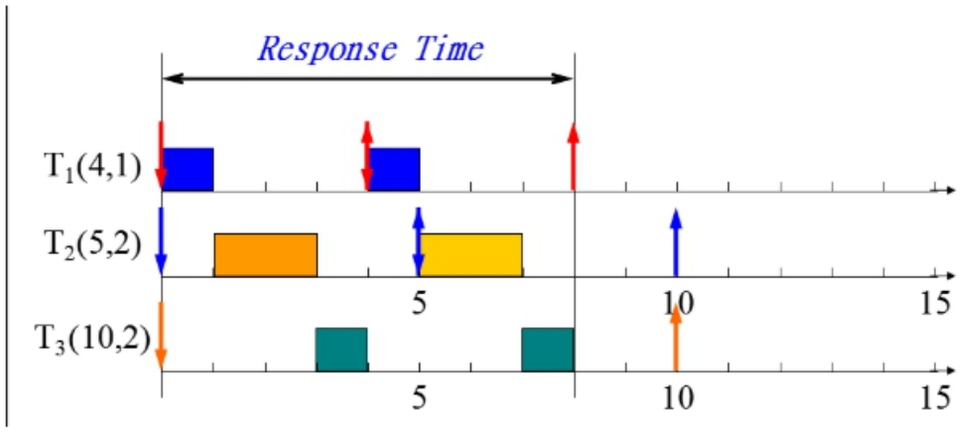


图4.2.2-2：RM调度过程示意图（response）

4.2.3 RM调度模型的前提假设

- 所有的任务是相互独立的（例如，它们不相互作用，不相互依赖）
- 所有任务都是周期的，且运行时间不变
- 没有任何任务因等待外部事件而阻塞。
- 所有任务共享一个共同的释放时间（临界时刻，critical instant）
- 所有的任务的截止时间等于它们的周期，即任务必须在下一个作业释放之前完成。

4.2.4 RM调度模型使用限制

如果一个实时系统能够使用RM调度模型，那么有：

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

如，对于周期任务 T1(4,1)，T2(5,1)，T3(10,1)，有：

$$\sum U_i = 1/4 + 1/5 + 1/10 = 0.55$$

$$3\,(2^{1/3}-1) \approx 0.78$$

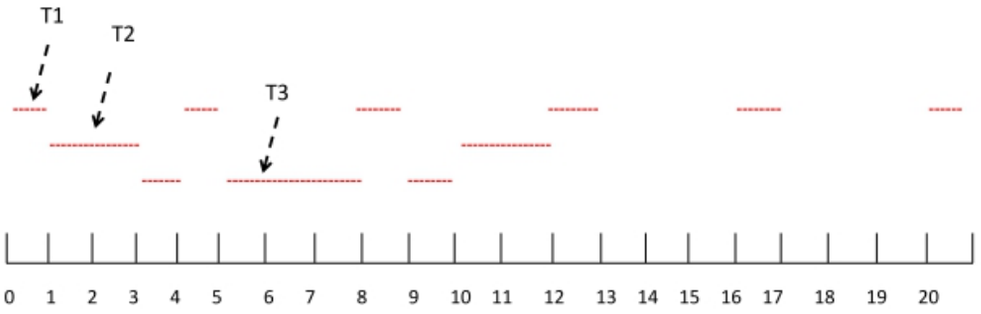
因此，以上三个任务是能够被RM模型调度的。

例题：

2、 假设系统存在任务、执行时间及运行周期如下：

任务	执行时间	周期	优先级
T1	1	4	1
T2	2	10	2
T3	5	20	3

根据 RM 调度方法，描述任务执行的调度图。（20 分）



4.3 EDF（最早截止时间优先，Earliest Deadline First）调度算法

4.3.1 调度算法描述

- 最佳的动态优先级调度
- 具有较早截止时间的任务具有较高的优先级
- 以最早的截止时间执行任务

4.3.2 调度过程

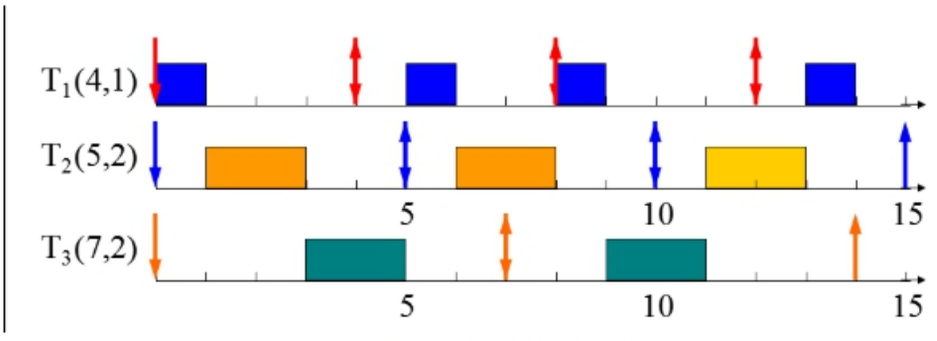


图4.2.2-1：EDF调度过程示意图

4.3.3 EDF 调度模型使用限制

如果一个实时系统可以使用 EDF 算法进行调度，则有： $\sum U_i \leq 1$

例题：

二、**Earliest Deadline First (EDF):** Given are five tasks with arrival times, execution times and deadlines according to the following table. Determine the Earliest Deadline First (EDF) schedule. Is the schedule feasible? (10 points)

	T1	T2	T3	T4	T5
a_i	0	2	0	8	13
c_i	3	1	6	2	3
d_i	16	7	8	11	18

(1) The EDF schedule is feasible, and the respective schedule is shown in the Figure 3.

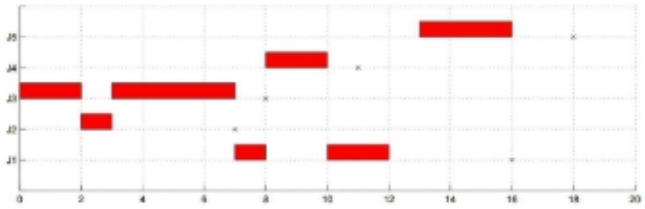


Figure 3: EDF schedule.

5 . UML 建模（Modeling with UML）

参考资料：

- *UML 基础: 统一建模语言简介*
- *五分钟读懂 UML 类图*
- *UML 实践详细经典教程*
- *活动图实例：细谈 UML 建模语言中的活动图模型*
- *UML 建模之活动图介绍*
- *UML 学习（三）-----序列图*
- *UML 2.0 时间图*
- 4.modeling with UML（requirement）.ppt

录音笔 UML 建模 ：

https://wenku.baidu.com/view/0e746a200722192e4536f66e.html

5.1 用例图（Use case diagrams），类图（Class diagrams），序列图（Sequence diagrams）和状态图（State diagrams）

哪些图可以描述用例：

用例图，顺序图，协作图，状态图，活动图 。

- 用例图：描述用户所看到的系统的功能行为，用例图从用户的角度来表示系统的功能（静态视图）

包括：参与者，用例

- 类图：描述系统的静态结构（对象，属性，联系），类图表示系统的结构

- 序列图：描述系统对象之间的动态行为，序列图将系统的行为表示为不同对象之间的消息交互
- 状态图：描述单个对象的动态行为，用有序的动态行为来表示单个对象的行为

状态图是描述某一对象的状态转化的，它主要是展示的是对象的状态。描述的是一个对象的事情。从状态图中我们可以看出，对象在接受了事件刺激后，会做出什么样的反应。

活动图是描述系统在执行某一用例时的具体步骤的，它主要表现的是系统的动作，描述的是整个系统的事情。

(1)、流程图着重描述处理过程，它的主要控制结构是顺序、分支和循环，各个处理过程之间有严格的顺序和时间关系。而 UML 活动图描述的是对象活动的顺序关系所遵循的规则，它着重表现的是系统的行为，而非系统的处理过程。

(2)、UML 活动图能够表示并发活动的情形，而流程图不行。

(3)、UML 活动图是面向对象的，而流程图是面向过程的。

在需求分析阶段使用：用例图和活动图。

在详细设计阶段使用：时序图。

活动图与状态图的异同：

1.活动图与状态图的相同点：

都是对系统的动态行为建模。

2.活动图与状态图的区别：

①描述对象不同

> **状态图：**描述对象状态及状态之间的转移；
> **活动图：**描述从活动到活动的控制流。

②使用场合不同

> **状态图：**描述对象在其生命期中的行为状态变化；
> **活动图：**描述过程的流程变化。

顺序图调用顺序：

Sequence Diagrams-Partial Ordering

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

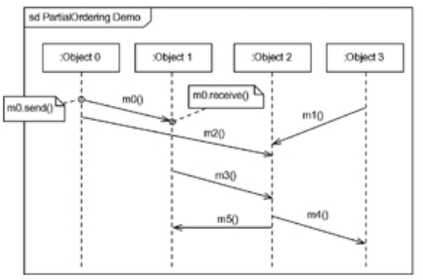
m1.send()->m1.receive()

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*.

71

对象2上的动作顺序：m1.receive-> m2.receive（） - > m3.receive（） - > m4.send（） - > m5.send（）

对象1上的动作顺序：m0.receive（） - >m3.send()- > m5.receive（）

如果m2.receive（）之前到达m0.receive（），这被称为消息 *超车（超车，超过，赶上）*。

88

5.2 行为建模

从逻辑上讲，可以将单个元素或元素组的行为建模为三种不同类型：（详见 4.modeling with UML.ppt p72~p82）

- 简单行为

- 状态行为
- 持续行为

其他非功能性限制也经常适用。这种约束被称为 QOS 约束，完成这一行为的质量。

例如：一个动作需要多长时间？两位数的准确度足够了吗？QOS 约束通常使用约束语言进行建模（OCL）。

UML 的主要特征是支持有限状态机。状态图的两个基本概念是状态和转换。

5.3 状态图（state chart）

状态的特点：

可以在状态图上捕获行为的对象被认为是被动的。

这样一个对象的行为空间被分解为存在的不相交和不相交的条件，称为状态。

转换是对导致状态改变的事件的响应。该对象可以在接收到事件（尽管不进行转换）或者进行转换时执行动作，进入或退出状态。

UML 中定义了四种事件：

- SignalEvent：与信号相关的事件。

  Signal 是一个异步通信的规范，所以 SignalEvent 是一个与异步接收信号相关的事件。
- CallEvent：与调用相关联的事件。

  Call 是一个同步通信的规范，所以 CallEvent 允许一个对象直接调用其中一个方法来影响另一个对象的状态。
- TimeEvent：与时间流逝相关的事件，通常用 tm（<duration>）或 after（<duration>）表示。

几乎所有的 TimeEvents 都是相对的时间。也就是说，它们指定事件将在对象处于指定状态至少<duration>时间单位后发生。

如果对象在超时之前离开该状态，则与该持续时间关联的逻辑计时器将消失，而不会创建超时事件

- ChangeEvent：与属性的值更改关联的事件。

它很少用在软件应用程序中。然而，当一个状态属性被内存映射到一个硬件上时，它可以用来指示内存地址改变的值

转换（transition）

转换是从起始状态开始并在目标状态结束的弧。

转换通常具有命名的事件触发器，可选地随后是执行转换时执行的动作（即可执行语句或操作）。

过渡事件签名的格式为：

event-name '(' parameter-list ')' '[' guard-expression ']' '/' action-list

注：事件可以指定形式参数列表，这意味着事件可以携带实际的参数。

digit(key: keyType)/ show(key)

约束（Guard）和行为执行顺序（Execution Order）：
guard 表达式是一个布尔表达式，包含在方括号中，必须计算为 true 或 false。
行动的执行顺序很重要，基本规则是退出->转换->进入（exit-transition-entry）。
也就是说，先行状态的退出动作首先执行，然后是转换动作，随后是后续状态的进入动作。

注：Guard 的原意为“保卫，守卫”，根据实际意义暂译为“约束”，不一定准确。

5.4 状态机（state chart）

伪状态（Pseudostates）符号表示，历史伪状态（History）：详见 4.modeling with UML.ppt p91~p98

5.5 活动图（Activity Diagrams）

UML 1.x 中的活动图在语义上等同于状态图，并共享一个通用的元模型。
在 UML 2.0 中，活动图给出了独立于状态图的语义基础。 在 UML 2.0 中，有多个级别的活动图：基本的，中间的，完整的等等。
详见 4.modeling with UML.ppt p99~p104

活动图元素：
1、活动状态图（Activity）
2、动作状态（Actions）
3、动作状态约束（Action Constraints）
4、动作流（Control Flow）
5、开始节点（Initial Node）
6、终止节点（Final Node）
7、对象（Objects）
8、数据存储对象（DataStore）
9、对象流（Object Flows）
10、分支与合并（Decision and Merge Nodes）
11、分叉与汇合（Fork and Join Nodes）
12、异常处理（Exception Handler）
13、活动中断区域（Interruptible Activity Region）
14、泳道（Partition）

- UML 中有三种主要的图形式来描述交互场景 ： 交互图（communication diagrams）
- 序列图（sequence diagrams）
- 时序图（timing diagrams）

5.6 序列图（Sequence Diagrams）

序列图：详见 参考资料 *UML 学习（三）-----序列图*
序列图局部顺序的确定，如作业 exercise 中的第二题。
序列图中的循环

5.7 时间图（Timing Diagrams）

时间图： 详见参考资料 *UML 2.0 时间图*
值生命线（Value lifeline）和状态生命线(State lifeline)

5.8 时间自动机（Timed automata）

时间自动机是实时系统建模与验证的理论。 具有相同目的的其他形式的例子还有时间 Petri 网（timed Petri Nets），

时间过程代数（timed process algebras）和实时逻辑（real time logics）

公平地说，以时间自动机作为输入语言的核心开发的几种模型检查器是理论应用和发展的动力。

时间自动机本质上是一个用实值变量（时间系统）进行扩展的有限自动机（包含一组有限节点和一组有限带有标记的边的图）

- 变量（variables）对系统中的逻辑时钟进行建模，在系统启动时用零初始化，然后以相同的速率同步增加
- 时钟约束（clock constraints），即边上用于限制自动机行为的约束（Guard）
- 当时钟值满足在边上标记约束（Guard）时，边表示的转换（transition）就可以发生。 当进行转换时，时钟可能被重置为零。

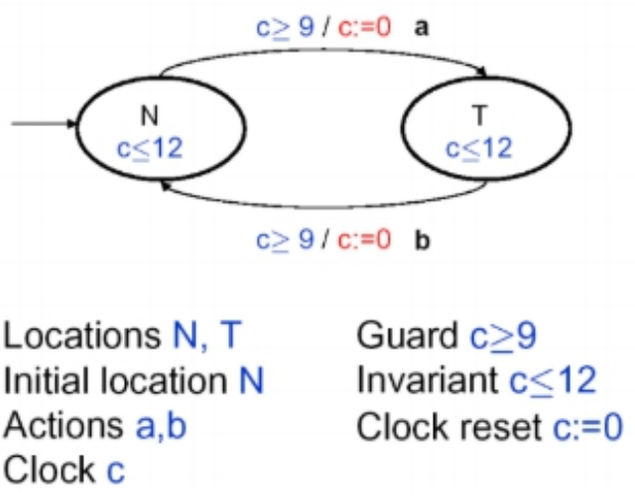


图5.8-1：时间自动机示例图

时间自动机语法：Review.ppt p96

时间自动机工具 UPPAAL

- 同步通信（synchronous）通过使用输入输出行为的握手同步（hand-shake synchronization）来实现
- 异步通信（asynchronous）使用共享变量（shared variables）来实现

为了对握手同步进行建模，字母 ∑ 定义为包括：输入动作 a？，输出动作 a！，内部行为 ‘T’

时间自动机示例：铁路交叉口控制系统的建模与验证：Review.ppt p101~p103

5.9 UML 预定义包（UML profile）

UML 是一种通用的建模语言,但是其缺少一种严格的形式化语义基础,从而无法对其所建模型进行形式化的分析以发现设计阶段的问题.本文结合了最新的 MARTE 规范中引入的时间相关模型元素,研究了 UML 状态机图的语义基础,提出一种基于扩展层次式时间自动机的形式操作语义方法.该方法通过一个汽车工业领域用例加以说明.基于该语义基础,提出一种算法,通过模型转换,自动地将 UML/MARTE 模型转换为模型检测工具 UPPAAL 的建模语言,验证了所提方法的正确性.

一个 UML 预定义包是一个扩充具有补充信息的 UML 模型的工具。 这种机制可以以两种方式使用：

- 用以扩展 UML 语言。

  例如，UML 没有提供明确的信号量概念，但可以通过重载现有的 UML 概念（如 Class）来添加它。

  结果是一种特殊的类，它除了标准的类语义之外，还包含信号量语义。
- 可用于将附加信息附加到辅助用途（如模型分析或代码生成）所需的模型。

  例如，可以使用这种注释来指定类的某些操作的最差情况执行时间，这可能需要用于分析应用程序的时序特性。

TimedObservation 是 TimedInstantObservation 和 TimedDurationObservation 的抽象超类。

TimedInstantObservation 表示与事件发生（eocc 属性）相关联并且在给定时钟上观察到的时刻。

UML 预定义包的优点：

- 语言基础设施的再利用（工具，规格）
- 需要较少的语言设计技能
- 允许扩展构造型采用新（图形）符号
- 配置文件可以定义模型视点

缺点：受 UML 元模型的约束

5．设计阶段（Design Phase）

参考资料：4.modeling with UML（design）.ppt

- 架构设计（Architectural design）详细介绍了最大的软件结构，如子系统，软件包和任务
- 机械设计（Mechanistic design）包括共同努力实现共同目标的类
- 详细设计（Detailed design）指定了各个类内部的原始数据结构和算法

表格中详细描述了三个类别：

Design Phase	Scope	What Is Specified
Architectural	System-wide Processor-wide	• Subsystems and components • Concurrency and resource management • Distribution across multiple address spaces • Safety and reliability management • Deployment of software onto hardware elements
Mechanistic	Collaboration-wide	• Instances of design patterns of multiple objects collaborating together • Containers and design-level classes and objects • Medium-level error handling policies
Detailed	Intra-object	• Algorithmic detail within an object • Details of data members (types, ranges, structure) • Details of function members (arguments, internal structure)

图5-1：三个层次的设计类别

Notice:

- 对于简单的系统，大部分的设计工作可能会花在机械和细节层面。
- 对于包括航空电子设备和其他分布式实时系统在内的大型系统而言，架构级别设计对项目成功至关重要。
- 4.modeling with UML（design）.ppt 中主要讲的是架构层次的设计

CPS 系统概述：

Cyber（信息技术）：**计算，通信，控制（*3C* 技术，*computation, communication, control* ）**，并且离散，逻辑，交换的系统；

Physical（物理系统）： 自然的和人造的系统受物理定律的支配，并持续运行；

***Cyber-Physical Systems*（信息物理融合系统）：网络和物理系统紧密结合在一起的系统。是一个结合计算、网络和物理环境的多位复杂系统，通过 *3C* 技术的有机融合与深度协作，实现大型工程系统的实时感知，动态控制和信息服务。**

CPS 系统的特点：

5. **信息-物理的耦合由新的需求和应用驱动**
   {
   网络工作在多个和极端的规模下；
   每个物理组建中的网络性能；
   打给莫的有线和无线网络
   }
6. **系统的系统**
7. **新的时空约束（New patial-temporal constraints）**
   {
   多个时间和空间尺度上的复杂性；
   动态重组/重新匹配；
   非传统法的计算和物理基块
   }

8. **无处不在的安全和隐私需求；**
9. **操作必须可靠**

硬实时（*Hard real time*）：有一个刚性的、不可改变的时间限制，它不允许任何超出时限的错误。超时错误会带来损害甚至导致系统失败、或者导致系统不能实现它的预期目标。不仅要求任务相应要求实时，而且必须在规定的时间内完成，满足最后期限的限制，否则会给系统带来灾难性的后果。因此验证是至关重要的：即使在最坏的情况下，系统也能满足所有的截止时间，确定性保证。

实时系统的描述（*Description of the real-time system*）：

及时性（timeless）：行为的及时性与时间约束有关，如截止时间。最后期限可能很硬或很软。时效性的重要建模问题是建模执行时间，截止时间，到达模式，同步模式和时间源。

并发性（concurrency）：多个操作顺序链的同时执行。围绕并发系统执行的问题与此有关：

a.并发线程的调度特性
b.即将到来的事件的到来模式
c.线程必须同步时使用的集合点模式
d.控制对共享资源访问的方法

可预测性（predictability）：实现系统的时候，能够预知未来执行到哪里。

正确性（correctness）：正确性表明一个系统总是运行正确。

鲁棒性（correctness and robustness）：鲁棒性表明系统即使在遇到新的情况（不在计划中）下也是可靠的。因此必须警惕死锁，竞争以及其他异常情况。

时间自动机（*Time automaton*）：

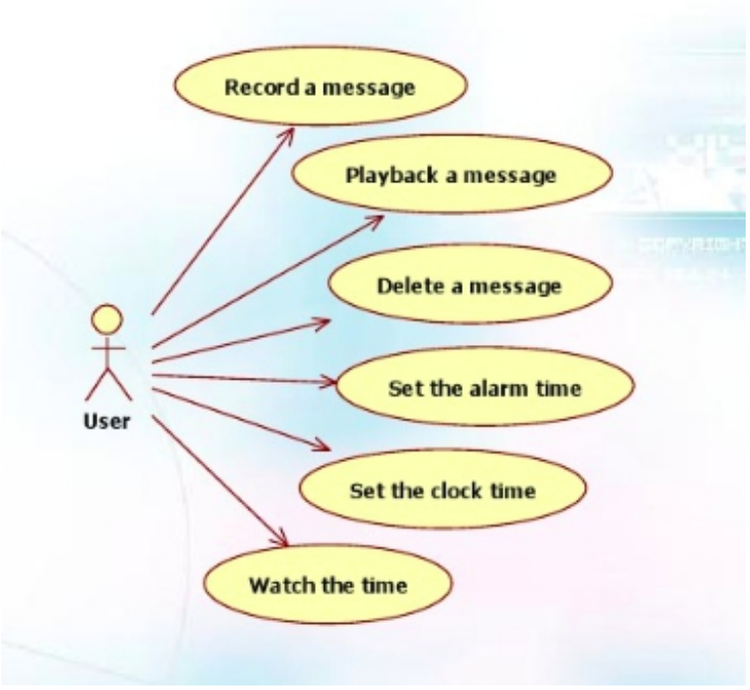
8

变量（*variables*）：对系统中的逻辑时钟进行建模，在系统启动时用零初始化，然后以相同的速率同步增加

时钟约束（*clock constraints*），即边上用于限制自动机行为的约束（*Guard*）当时钟值满足在边上标记约束（*Guard*）时，边表示的转换（*transition*）就可以发生。 当进行转换时，时钟可能被重置为零。

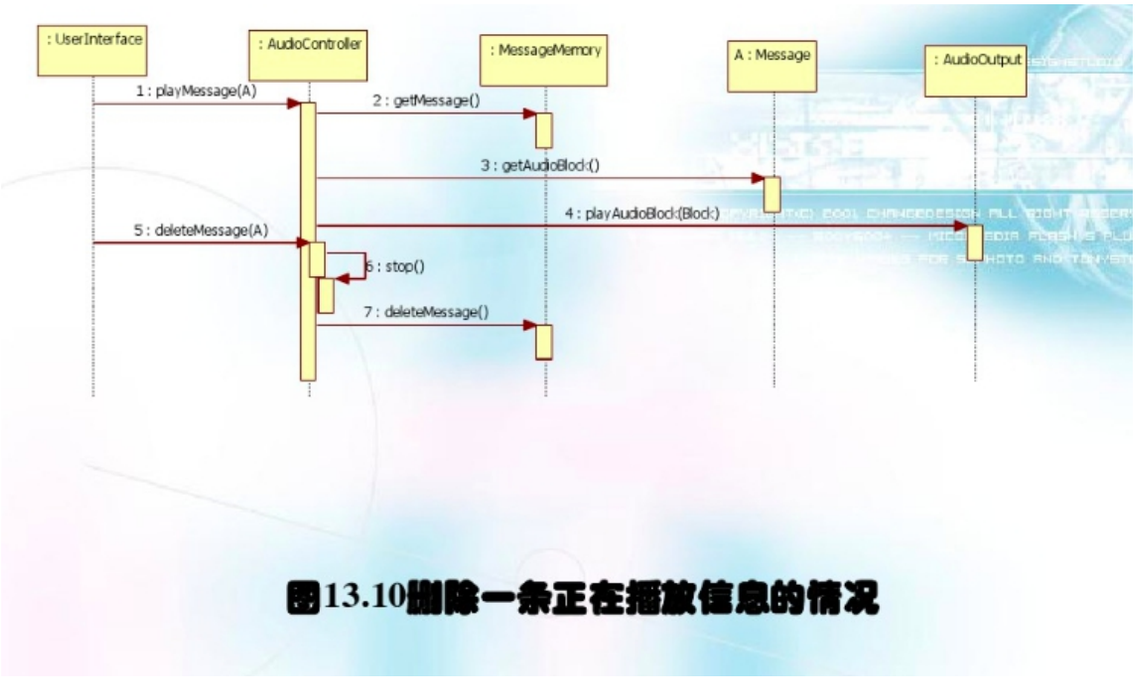
录音机

用例图



顺序图

2



EDF：

EDF 调度模型使用限制

如果一个实时系统可以使用 EDF 算法进行调度，则有： $\Sigma Ui \leq 1$

小火车：

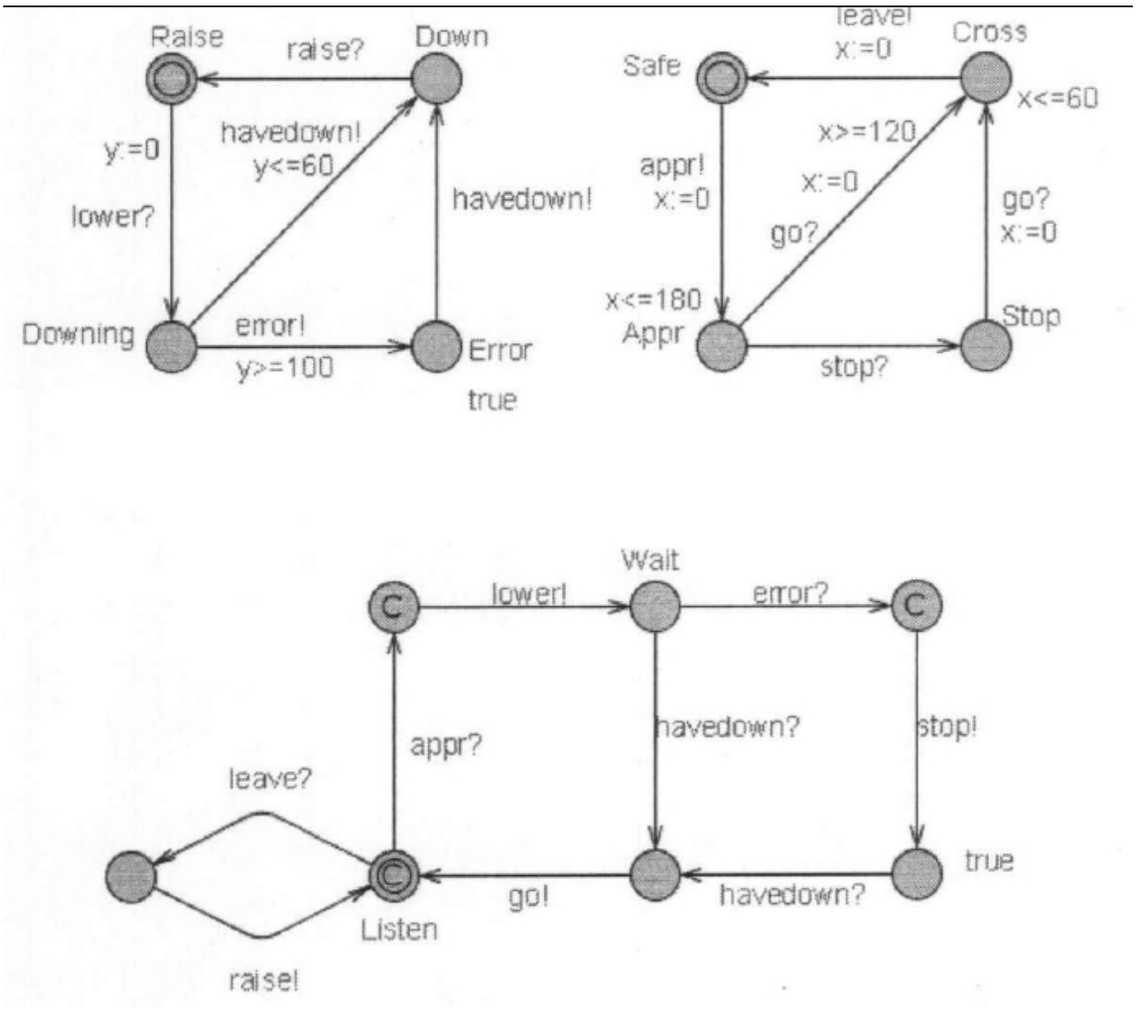


图 5.1 火车、自动门和控制器的时间自动机模型

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。

火车处于 **safe** 状态，并在靠近路口时发出 **appr** 同步事件；

控制器处于 **listen** 状态，在接收到同步事件 **appr** 后，迁移至委托状态（**C**）并发出 **lower** 同步事件，并迁移至 **wait** 状态；

控制门处于 **Raise** 状态，当接收到 **lower** 事件后，迁移至 **Downing** 状态，并将时钟 **y** 设置为 **0**；

控制门在成功下降后，发出 **havedown** 事件；

控制器处于 **wait** 状态并接收到 **havedown** 同步事件，并作出状态的迁移；并发出 **go** 的同步事件；

火车接收到 go 的信号，开始通过路口，并发出 leave 的信号，并将时钟 x 设置为 0；

控制器处于 Listen 状态，当收到 leave 的信号时，发出 raise 信号，回到 Listen 状态；

控制门处于 Down 状态，当收到 raise 的信号时，迁移至 Raise 状态。

控制门处于 Downing 状态，当时钟 y>=100 时，发出 error 的同步事件，并迁移至 error 状态；

控制器处于 wait 状态，接收到 error 同步时间后，迁移至（c）状态，并发出过 stop 同步事件，

迁移至 true 状态；
火车处于 appr 状态时，当时钟 x<=180 时，接收到 stop 同步事件，则迁移至 stop 状态；
控制门处于 error 状态，发出 havedown 同步事件，迁移至 Down 状态；
控制器处于 true 状态，接收到 havedown 事件，并发出 go 同步事件，迁移至 Listen 状态；
火车处于 Stop 状态，收到 go 事件的同时，将时钟 x 置为 0，并迁移至 Cross 状态，时钟 x<=60 时，发出 leave 同步事件并将时钟 x 再次置为 0，回到 Safe 状态；
控制器处于 Listen 状态，收到 leave 事件，发出 raise 同步事件，并回到 Listen 状态；
控制门处于 Down 状态，当收到 raise 事件时，回到 Raise 状态。

模型验证性质：

(1) A[] Queue.list[N-1]==0,表示队列中不可能有 N 个元素，即数组永远不会溢出。实际上，模型定义了 N 作为火车的数量+1 来验证这个属性。可以使用与列车数量匹配的队列长度来检查此属性。

(2) A[] Train1.Cross+ Train2.Cross+ Train3.Cross+ Train4.Cross<=1:表示在任何一个时刻多只有一辆火车通过桥，该表达式使用了 Train1.Cross 来表示结果的真或假，即 1 或 0，易知，模型应该满足这个性质。

(3) E<> Train1.Cross and Train2.Stop and Train3. Stop and Train4. Stop:表示火车 1 正在过桥，火车 2,3,4 正在等待过桥，其他火车具有相似的性质。

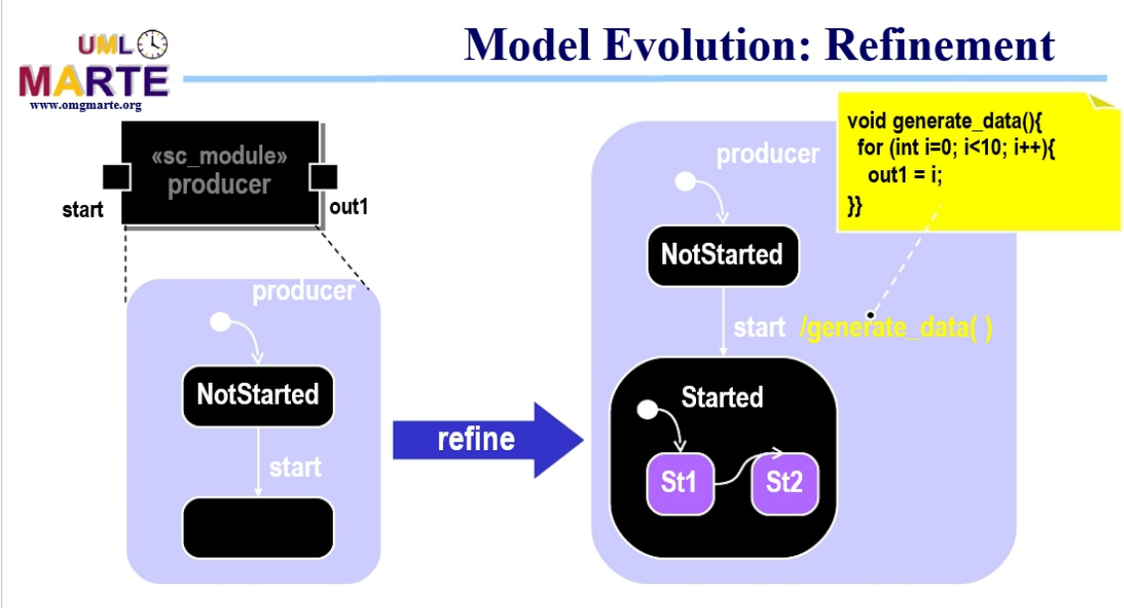
(4) 性质 E<>Train1.Cross and Train2.Stop 表示列车 1 可以过桥，而火车 2 正在等待过桥，其他火车具有相似的性质。

(5) 性质 E<>Train1.Cross 表示火车 1 可以过桥，可以用同样的方法检测其他火车。

UML：

spot the architecture（指出体系结构）

一个 producer 的模型和一个 consumer 的模型，他们之间存在着某种关系 link1



以 producer 模型为例，从一个模块进行扩展，producer 开始，在没有开始之前做的准

备工作，再设计开始之后的模块之后输出 out1；再进行细化，对 Started 进行细化开始 St1 到 St2 建立详细的模型到生成代码。

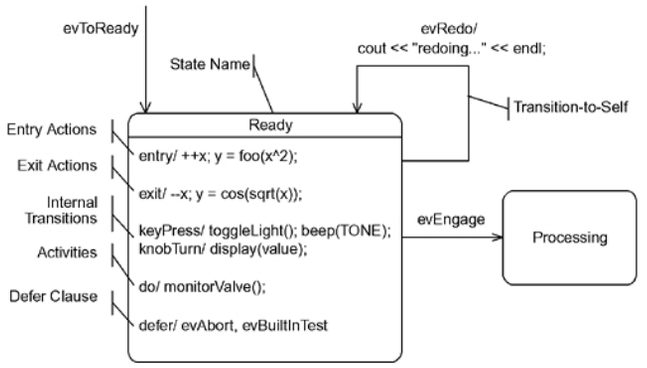
模型精化是工程中基于模型驱动开发的关键问题，被广泛应用于基本模型的驱动开发方法中。若在初始模型中引入过多细节会使得软件开发和测试不易管理，因此对于那些大型复杂系统的建模很难能够做到一步到位。在实际的软件开发建模过程中往往采用模型精化的技术，模型精化，即在原模型的基础上添加更多的细节，逐步细化，模型从刚开始的比较抽象变得逐渐具体化。

而模型精化过程中模型间的一致性是正确建模的必要前提，为了保证精化后的模型和精化前的模型是一致的，需要对精化前后的模型进行一致检验。一致性检验的重要意义就在于可在软件设计的早期阶段发现不一致问题，减少生成代码后出错所产生的代价。在软件开发过程中，模型与模型间存在不一致问题将导致用户需求和系统功能不符合，反应的不是同一个系统需求，使得软件的开发和维护代价过大，造成人力和财力资源的浪费，使得软件质量下降。

考点：执行 *evRedo* 之后的行动执行顺序，定义，实现过程

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.



59

实验：(查询语言)

实验一：

实验一 ATM 的死锁问题：要理解实验一的原模型是 Eric 每次取 10 块钱，在收到取钱请求后，Bank 会把 Eric 要取钱的数目与 Eric 账户的余额进行比较，如果超支，则会返回

not_Ok!信号，然后 Bank 回到起点状态。同时 ATM 会收到这个信号进行转换状态（退卡），然后 Atm 回到起点状态。但是 Eric 将会一直停留在等待 cash 信号的这个状态，无法回到起点状态，所以造成了死锁。

实验二：

实验三验证条件：
首先要明白，系统每次运行时都有可能走不同的路径，并且这些路径运行的次数不止一次，会反复运行，反复循环

A[] Observer.idle imply x<=3
可以理解为：对于 Observer 系统下的所有路径，当该路径到达 idle 状态的时候，时钟 x 的值如果都是小于或者等于 3 的话，那么该式子为真

A<>Observer.idle and x>2
可以理解为：对于 Observer 系统下的所有路径，当所有的这些路径到达 idle 状态的时候，对于每个路径时钟 x 的值至少有一次大于 2 的话，那么该式子为真

E[] Observer.idle and x>2
可以理解为：对于 Observer 系统下的某一条路径，对于该条特定路径而言，在每次该路径到达 idle 状态的时候，时钟 x 的值一定是大于 2 的，那么该式子为真

E<>Obs.idle and x>3
可以理解为：对于 Obs 系统下的某一条路径，当该路径到达 idle 状态的时候，时钟 x 的值至少有一次大于 3 的话，那么该式子为真

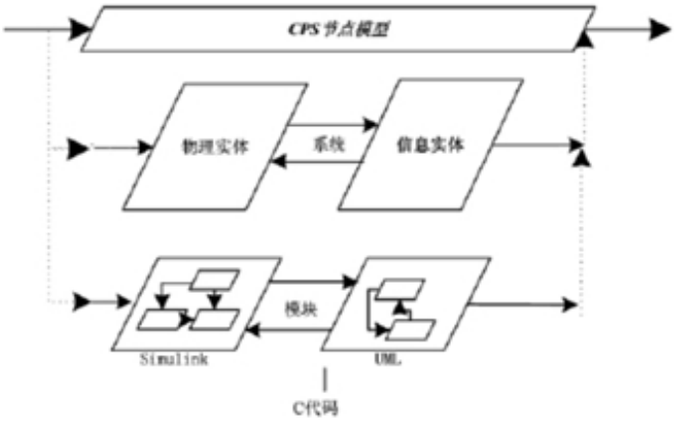
ppt 最后一页内容：

针对物理实体和计算实体的特征和建模方法，分别采用 Simulink / RTW 和 UML / Rhapsody 对物理实体进行建模和对计算实体进行建模，然后按照 CPS 系统异质模型融合思想将两个不同工具的模型进行融合。

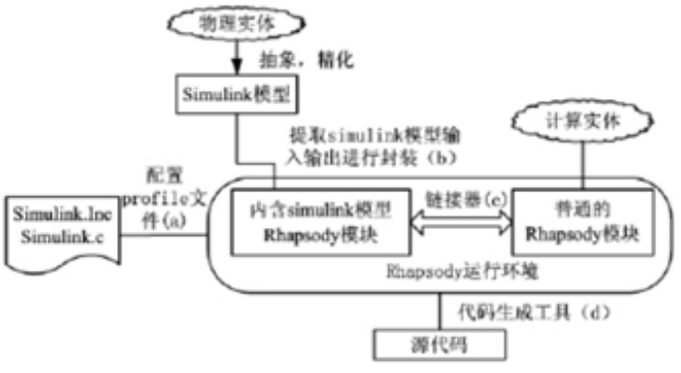
物理实体建模方法涉及到大量的微积分方程计算和时间连续行为，故采用 Simulink/RTW 来进行建模和代码生成。Simulink 被广泛应用于控制领域建模仿真，是一种高度图形化建模仿真工具，用来对动态系统进行建模、仿真和分析，是一种基于时间连续系统建模的可视化建模工具。Simulink 具有强大模块库，能够完成各种建模需求，并且应用 S 函数机制，用户可以方便地编写各种功能的 Simulink 模块[9]。而且 Simulink 和 Matlab 的无缝连接，使 Simulink 能够很容易调用 Matlab 的数学计算工具，具备强大的计算能力。RTW(real-time workspace)是 Simulink 的代码生成工具，能够实现从平台无关的 Simulink 模型生成各种语言的平台相关可执行代码，比如 C，C++,ADA（一种面向嵌入式系统和实时系统的编程语言）等。

统一建模语言 (UML) 是模型驱动的软件开发方法中广泛使用的建模和规约语言。最常用的 UML 图包括 用 例 图、类图、序列图、状态图、活动图、组件图和部署图。UML 建模语言以丰富的图形种类从不同角度支持系统设计与开发的各个阶段，是面向对象设计的标准语言，支持模块复用，在系统设计方面具有很大的优势，支持对模型设计的系统仿真，验证设计的各个 模 块 之 间 的 交 互、通 信 和 依 赖、继 承 等 关 系 [1 0]。Rhapsody 是 I B M推出的 UML 统一建模工具，遵循 UML 2． 1 协议，通过增加和定制特征文件 (Profile) 来支持面向特定领域的仿真建模，支持模型代码的自动生成，适用于事件驱动的计算实体建模仿真。

UML / Simulink 是一个功能强大的组合。在设计阶段，用户可以为每一个不同的活动选择最合适的建模工具。使用 UML，用户可以按需求捕获和分析函数，定义系统和软件架构，组织算法逻辑，模拟测试每一个模块的正确性，构建计算实体模型。使用 Simulink，用户可以创建设备模型定义的物理元素。它与系统交互并生成逻辑算法，控制这些设备模型的动态行为。代码可以从原型中使用的控制算法里产生，同时可以从硬件回路测试的设备模型中生成，模拟物理实体的动态特性



上图所示是 UML 模型和 Simulink 模型的融合结构。从外部看，这是一个 CPS 节点模型，但是在系统级上 划 分 为 物理实体模 型 和 信 息 实 体 模 型。底 层 的 模 块 则 采 用 Simulink 和 UML 模块来实现。最底层的 C 代码是将两种工具生成的代码手工 融 合 或 是 通 过 相 关 工 具 糅 合 起 来。UML 模 型 和 Simulink 模型的融合在 Rhapsody 中可以通过下述方式来实现。下图所示是模型融合流程图。



首先在分析 CPS 系统的特 征 和 属 性 基 础上将 CPS 系统划分为物理实体、计算实体和交互 实 体，再 通过精化、提取将 CPS 模型简化，将系统划分为物理 实 体 和

计算实体。采用不同实体不同建模方式的思想，分别提出了针对物理实体的基于连续时间的运动方程行为模型和面向计算实体的基于事件驱动的有限状态机行为模型。然后通过扩展物理实体行为模型和计算实体行为模型，实现了物理模型信息化和计算模型物理化，使两种模型融合在一起协同工作，刻画 CPS 系统。最后对两种模型所能采用的建模工具进行了可行分析，提出了物理行为模型采用 Simulink / RTW 来构建和生成代码，计算实体模型采用统一建模语言（UML）进行建模，并在此基础上提出了将 Simulink 模型转化为 UML 模型方法，完成了两种异质模型的协同建模仿真。