**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**

1. **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**

**}**

1. **无处不在的安全和隐私需求；**
2. **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

p o cci

**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 )**

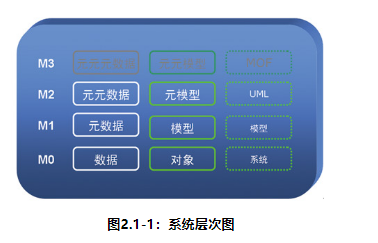
**参考资料**

* introduction.ppt （p34~p47）
* [*MDSF：模型驱动开发（MDD）介绍*](http://www.zhoujingen.cn/blog/2749.html)
* [*元模型*](http://blog.sina.com.cn/s/blog_8077b6d901012iyf.html)

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。



**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。

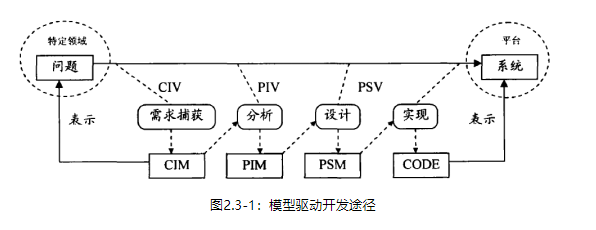
* 平台独立模型（PIM，Platform-Independent Model）

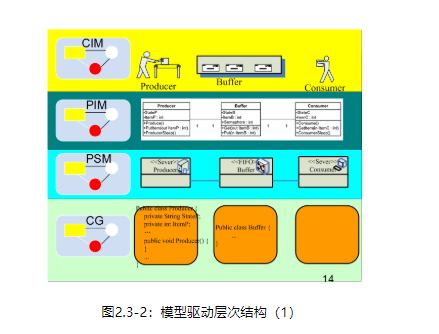
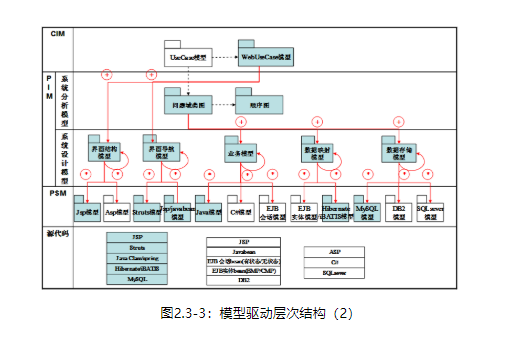
描述如何构造系统，而不涉及到用于实现模型的技术。此模型不描述用于为特定平台构建解决方案的机制。PIM 在由特定平台实现时可能是适当的，或者可能适合于多种平台上的实现。

* 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**





**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*](http://blog.csdn.net/thl789/article/details/617629)

**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：

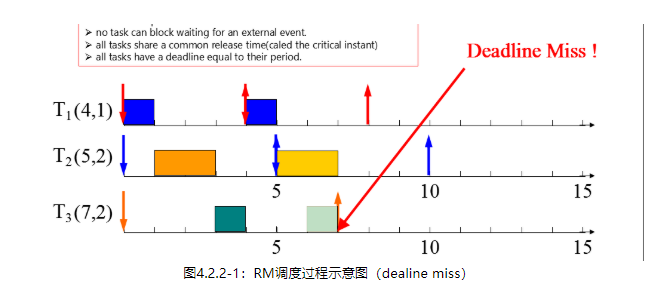
（1）RM的最优性可以描述为：如果一个任务集能够被静态调度, 那么RM调度算法就能够调度这个任务集。

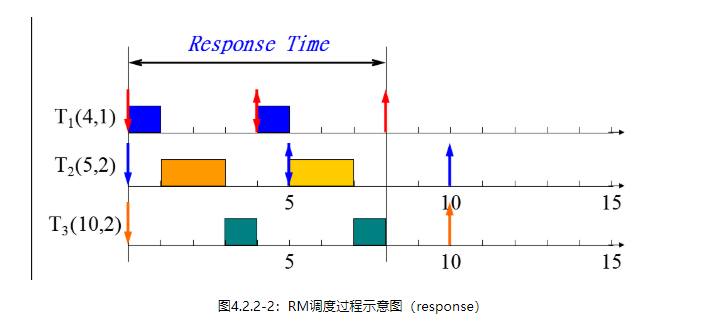
        其最最优性可以证明，证明方法类似于动态规划中的“剪贴法”。

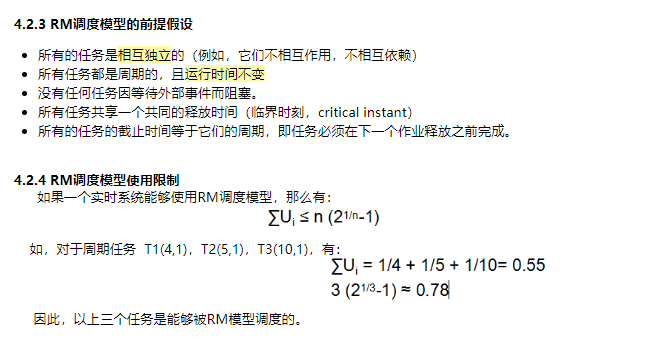
（2）RM调度算法属于“可抢占”式调度一类（事实上，后面的EDF也是这一类）。

**4.2.2 调度过程**

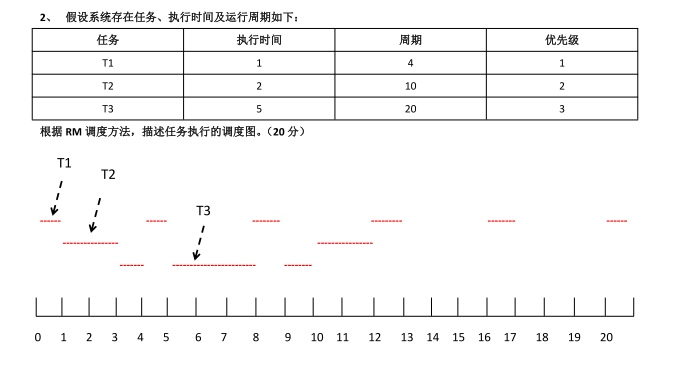
响应时间**：**任务从释放时间到结束时间







**例题：**

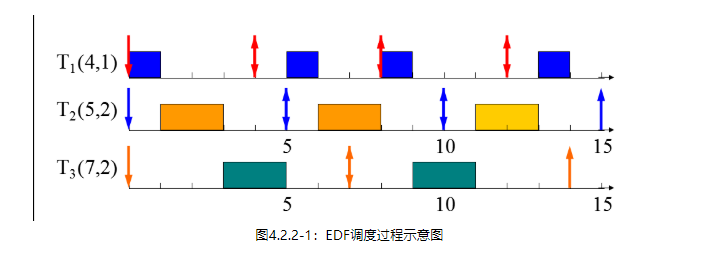


**4.3 EDF（最早截止时间优先，Earliest Deadline First）调度算法**

 **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

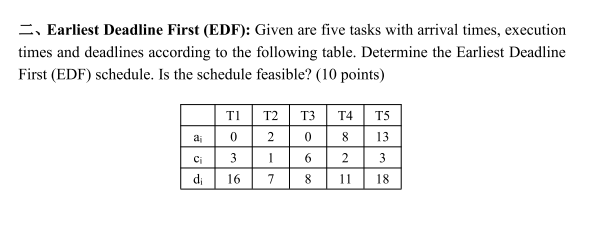
**4.3.2 调度过程**

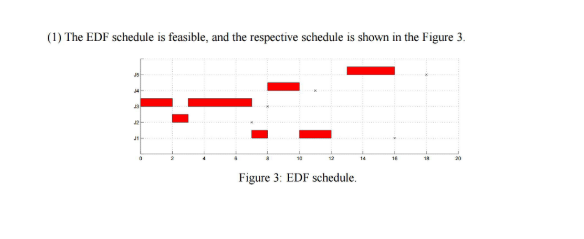


**4.3.3 EDF调度模型使用限制**

           如果一个实时系统可以使用EDF算法进行调度，则有： ∑Ui ≤ 1

**例题：**





**5 . UML建模（Modeling with UML）**

**参考资料：**

* [*UML基础: 统一建模语言简介*](https://www.ibm.com/developerworks/cn/rational/r-uml/)
* [*五分钟读懂UML类图*](https://www.cnblogs.com/shindo/p/5579191.html)
* [*UML实践详细经典教程*](http://www.uml.org.cn/oobject/201609092.asp)
* [*活动图实例：细谈UML建模语言中的活动图模型*](http://www.woshipm.com/pd/555549.html)
* [*UML建模之活动图介绍*](https://www.cnblogs.com/ywqu/archive/2009/12/14/1624082.html)
* [*UML学习（三）-----序列图*](https://www.cnblogs.com/silent2012/archive/2011/09/14/2172219.html)
* [*UML 2.0 时间图*](http://www.sparxsystems.cn/resources/uml2_tutorial/uml2_timingdiagram.html)
* 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。

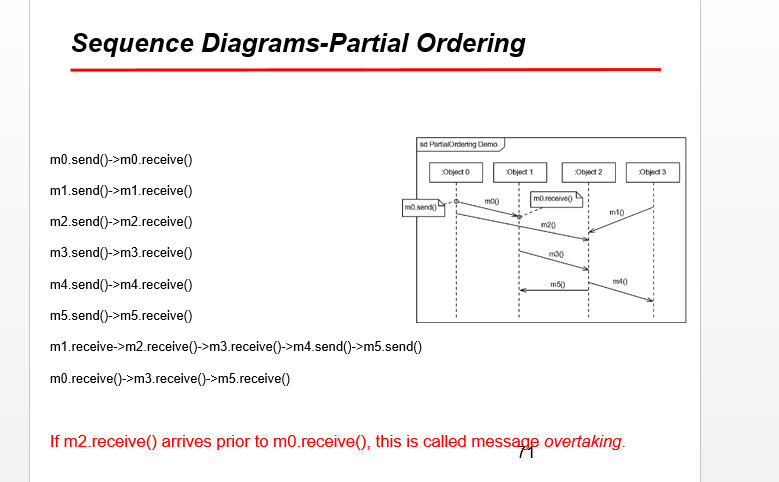
在需求分析阶段使用：用例图和活动图。

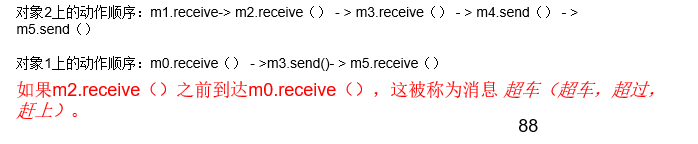
在详细设计阶段使用：时序图。

活动图与状态图的异同：



顺序图调用顺序：





**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*](https://www.cnblogs.com/silent2012/archive/2011/09/14/2172219.html)

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*](http://www.sparxsystems.cn/resources/uml2_tutorial/uml2_timingdiagram.html)

值生命线（Value lifeline）和状态生命线( State lifeline)

**5.8 时间自动机（Timed automata）**

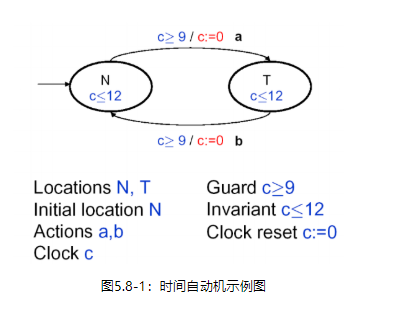
        时间自动机是实时系统建模与验证的理论。 具有相同目的的其他形式的例子还有时间Petri网（timed Petri Nets），

        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。



   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：



 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：**

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；**

**打给莫的有线和无线网络**

**}**

1. **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**

**}**

1. **无处不在的安全和隐私需求；**
2. **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）：**

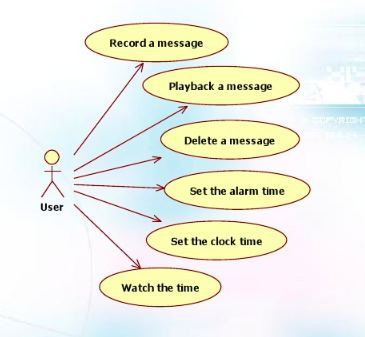
8

**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

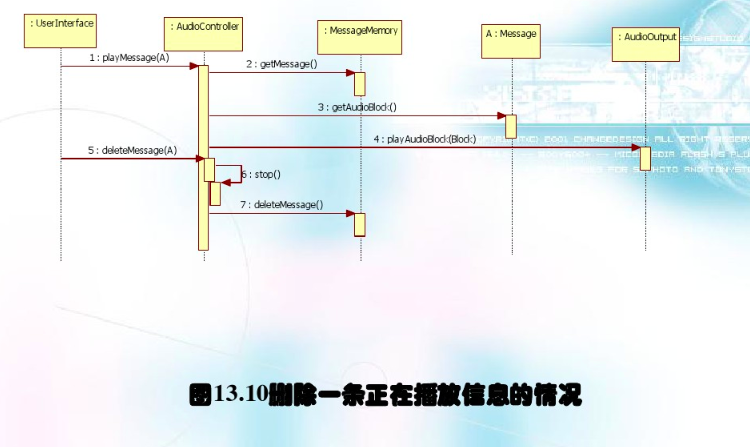
**时钟约束（clock constraints）**，即边上用于限制自动机行为的约束（Guard）当时钟值满足在边上标记约束（Guard）时，边表示的转换（transition）就可以发生。 当进行转换时，时钟可能被重置为零。

**录音机**

**用例图**

****

**顺序图**

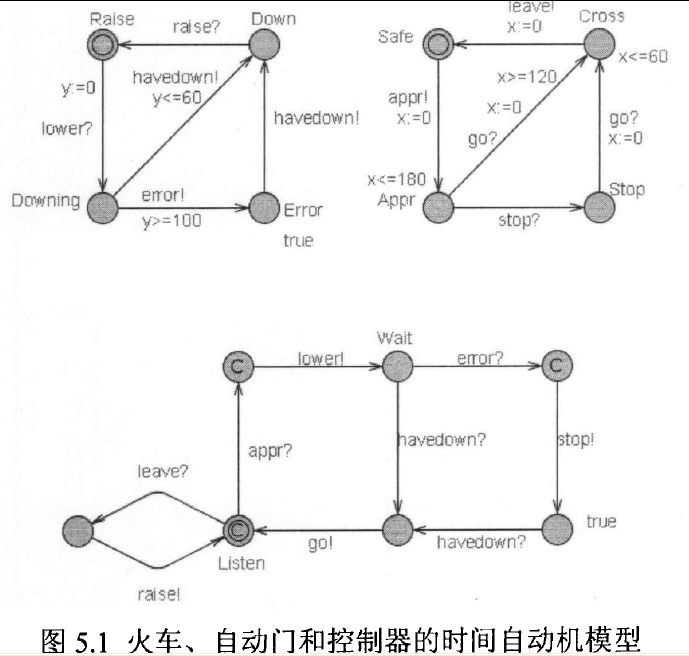
**2**

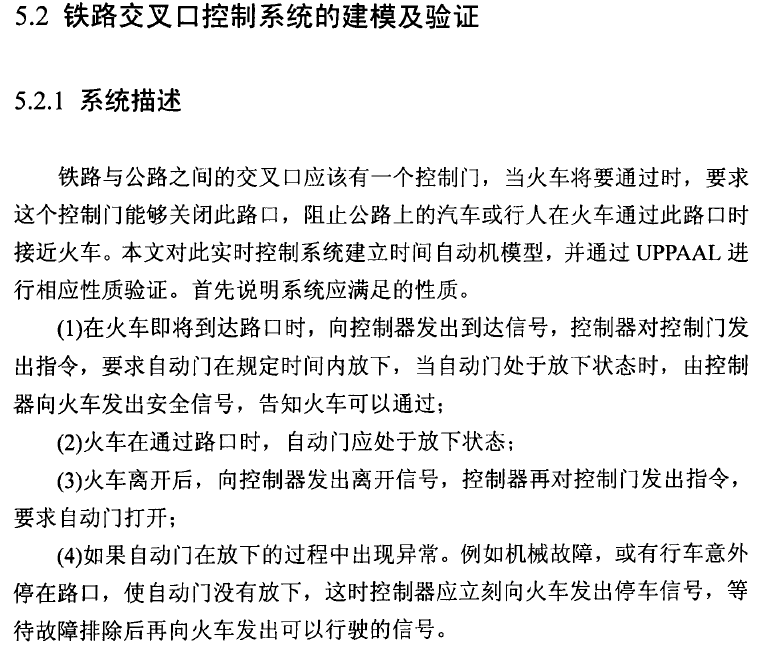
**EDF：**

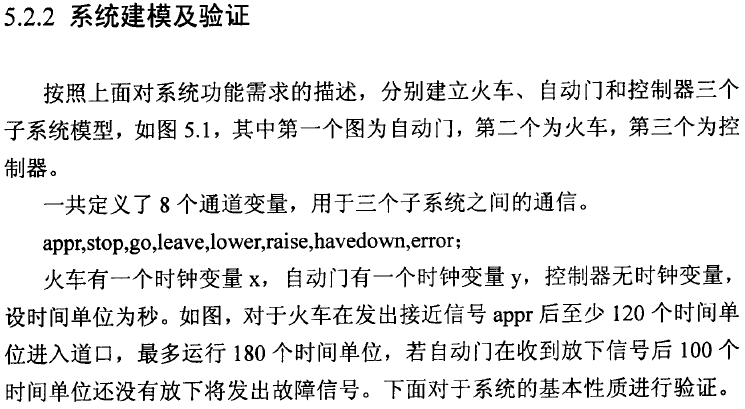
EDF调度模型使用限制

如果一个实时系统可以使用EDF算法进行调度，则有： ΣUi ≤ 1

**小火车：**







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。

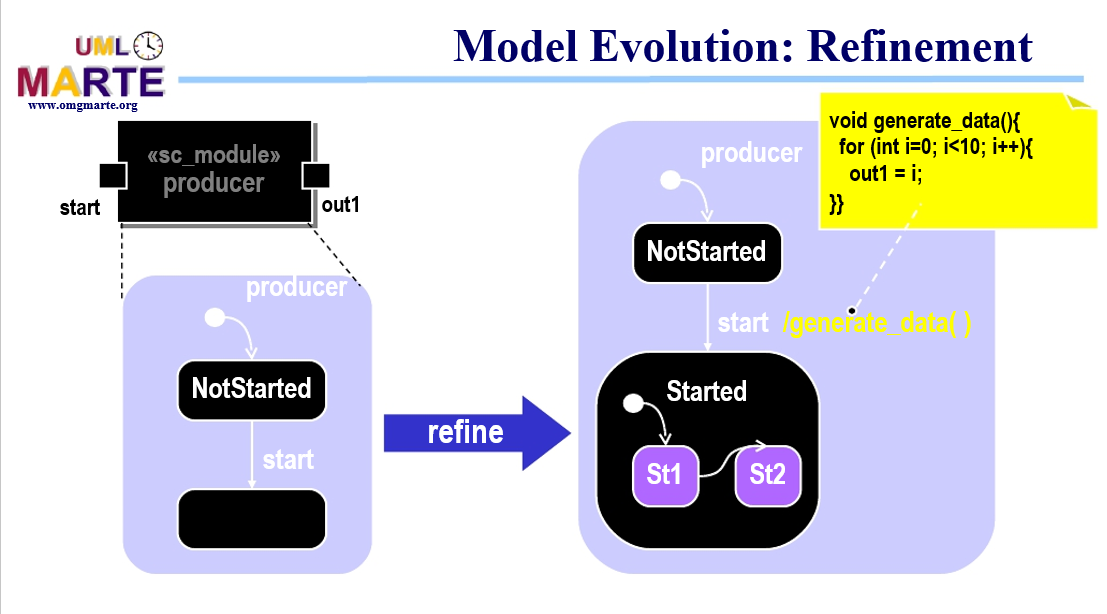
模型验证性质：

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。

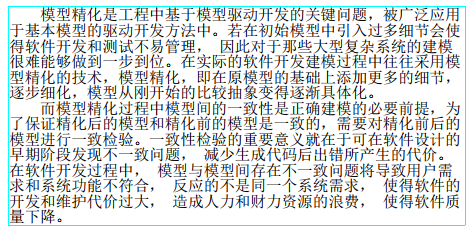
**UML：**

**spot the architecture（指出体系结构）**

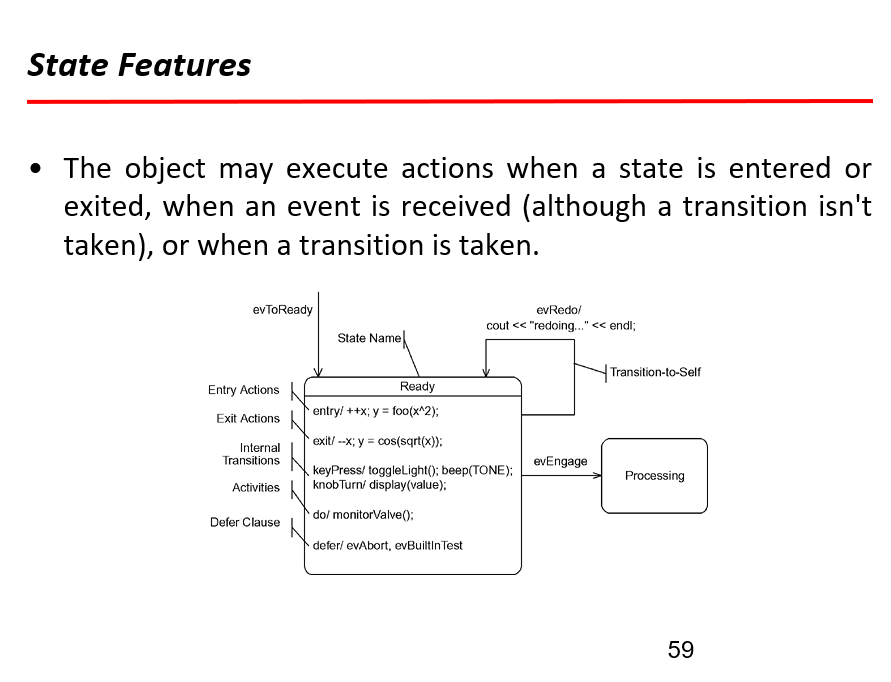
**一个producer的模型和一个consumer 的模型，他们之间存在着某种关系link1**



**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。**



**考点：执行evRedo之后的行动执行顺序，定义，实现过程**



**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

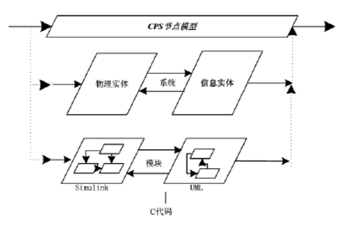
**ppt最后一页内容：**

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。

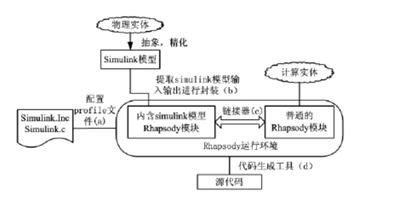
物理实体建模方法涉及到大量的微积分方程计算和时间连续行为，故采用Simulink/RTW来 进 行 建 模 和 代 码 生 成。Simulink被广泛应用于控制领域建模仿真，是一种高度图形化建模仿真工具，用来对动态系统进行建模、仿真和分析，是一种基于时间连续系统建模的可视化建模工具。Simulink具有强大模块库，能 够 完 成 各 种 建 模 需 求，并 且 应 用 Ｓ 函 数机制，用户可以方便地编写各种功能的 Simulink模块［９］。而且Simulink和Matlab的 无 缝 连 接，使 Simulink能 够 很 容 易调用Matlab的数学 计 算 工 具，具 备 强 大 的 计 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［１０］。Rhapsody是ＩＢＭ推出的UML统一建模工具，遵循 UML２．１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。