第3章 软件建模

软件建模的三个层面:CIM、PIM和PSM

（1）计算无关模型（Computation Independent Model，CIM）。
（2）平台无关模型（Platform Independent Model，PIM）。
（3）平台相关模型（Platform Specific Model，PSM），又称为平台特定模型。

图3-1 基于模型的软件开发生命周期

软件建模方法

- **结构化方法 (Structured Method)**
- **面向对象方法（Object Oriented Method）**
- **基于构件的开发方法（Component Based Development）**
- **面向服务方法（Service Oriented Method）**
- 面向方面方法（Aspect Oriented Method）
- **模型驱动开发方法 （Model Driven Development）**
- **形式化方法 (Formal Method)**
- 产品线开发方法和领域工程

2.1 结构化方法 (详见第4章)

- 核心：自顶向下，逐步求精
- 手段：分解（模块化）、抽象
- 任务：结构化分析、结构化设计、结构化编程
- 常用建模工具:
  - 需求建模：
    - DFD(数据流图)
    - DD(数据字典)、ERD(实体关系图)
    - STD(状态图)
  - 设计建模：
    - 概要设计：结构图（SC）
    - 详细设计：程序流程图、 N-S图、PAD图、伪代码
  - 结构化编程：三种经典程序结构

2.2 面向对象方法(详见第5章)

- 九十年代以来的主流开发方法
  - 符合人们对客观世界的认识规律
    面向对象=对象+类+继承+消息通信
  - 开发的系统结构易于理解、易于维护
  - 继承机制有力支持软件复用
- 常见的面向对象方法
  - Booch method 1994
  - Coad and Yourdon method 1991
  - Rambaugh method -- OMT 1991
  - Jacobson method - OOSE 1992
  - Wirfs-Brock method 1990
  - 国际标准统一建模语言 UML 1997

2.3 基于构件的开发方法
（Component Based Software Development，CBSD）

- 跨时间、跨空间、跨用户以及跨用户的共享
- 异构协同工作、各层次集成、反复重用
- 支持分布式计算与构件化集成、不同标准构件拼装

CBSD具备：

（1）用预先编程功能明确的部件定制而成，可用不同版本部件实现应用扩展；

（2）系统分解为相互独立协同工作的部件，可反复重用；

（3）利用统一的接口和标准实现跨平台的互操作。

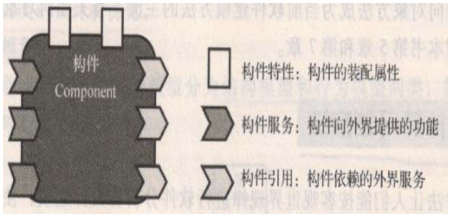
CBSD通过构造和组装可复用构件来开发一个新系统。

抽象程度：

- 面向对象方法，以类为封装单位，为类及重用；
- 构件化方法，对一组类进行组合封装，通过接口将底层多个逻辑组合成高层次的新构件，可在代码级、对象级、架构级、系统级重用，实现定制组装软件。

2.3.1 构件的概念

独立于特定平台和应用系统，具有标准接口，支持一定功能，可重用与自包含的软件构成部分。



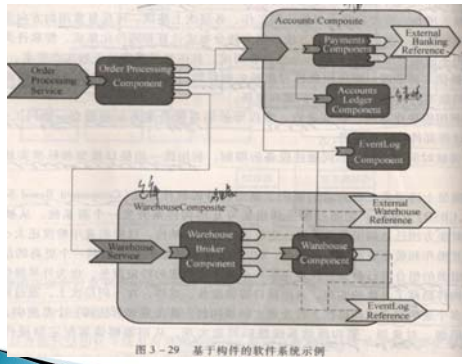


图3－29 基于构件的软件系统示例

2.3.2 构件的类型

（1）按重用的方式

白盒构件、灰盒构件、黑盒构件

（2）按使用的范围

通用构件、领域共性构件、应用专用构件

（3）按粒度的大小

小粒度，基本数据结构构件，如窗口、菜单等

中粒度，如文本录入、查询、删除功能构件等

大粒度，子系统构件，如文本编辑子系统、图形图像处理子系统等。

（4）按用途

系统构件，整个集成与系统环境中都使用的构件；

支撑构件，在集成环境和构件管理系统中使用的构件；

领域构件，专门领域开发的构件。

（5）按构件结构

原子构件，无需再分的最小基本单元；

组合构件，由多个构件聚集而成。

2.3.2 常用的构件标准

（1）CORBA

CORBA（Common Object Request Broker Architecture）,公共对象请求代理体系结构）是由OMG组织制订的一种标准的面向对象应用程序体系规范。CORBA体系结构是对象管理组织（OMG）为解决分布式处理环境(DCE)中，硬件和软件系统的互连而提出的一种解决方案。

CORBA的三个层次：

- 最底层：对象请求代理ORB，分布对象的定义（接口）和语言映射，实现对象间的通信和互操作，是分布式对象系统中的软总线；
- 中间层：公共对象服务，提供并发服务、名字服务、事务服务、安全服务等；
- 最上层：公共设施，定义了构件框架，提供直接由业务对象使用的服务，规定了协作规则。

优点：大而全、互操作性和开放性好；

缺点：慢

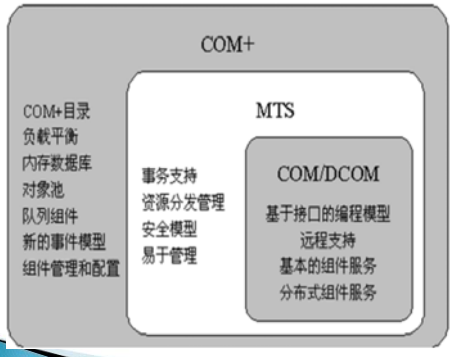
（2）COM/DCOM/COM+

COM(Component Object Model）提供了一个 Windows 平台上的对象通讯技术，并且逐渐成为应用程序之间彼此通讯及互动的技术主流。

DCOM(Distributed COM)，DCOM 解决了计算机的通信和互动技术，负责让 COM 组件可以在网络环境下持续提供服务。

COM+把COM组件提升到应用层，把底层细节留给操作系统，使COM+与操作系统的结合更加紧密。COM+的底层结构仍然以COM为基础，但在应用方式上则更多地继承了MTS（Microsoft Transaction Server）的处理机制，包括MTS的对象环境、安全模型、配置管理等。

COM+把COM、DCOM和MTS三者有机地统一起来，同时也新增了一些服务，如负载平衡、内存数据库、事件模型、队列服务等，形成一个概念新、功能强的组件体系结构，使得COM+形成真正适合于企业应用的组件技术。



（3）Java EE 和 EJB

Java EE（Java Platform，Enterprise Edition，J2EE）是sun公司推出的企业级应用程序版本，能够帮助我们开发和部署可移植、健壮、可伸缩且安全的服务器端 Java应用程序。它提供 Web 服务、组件模型、管理和通信 API，可以用来实现企业级的面向服务体系结构（service-oriented architecture，SOA）和 Web 2.0应用程序。

EJB是sun的JavaEE服务器端组件模型，设计目标与核心应用是部署分布式应用程序。EJB 定义了一个用于开发基于组件的企业多重应用程序的标准。其特点包括网络服务支持和核心开发工具(SDK)。 在J2EE里，Enterprise Java Beans(EJB)称为Java企业Bean，是Java的核心代码，分别是会话Bean（Session Bean），实体Bean（Entity Bean）和消息驱动Bean（Message Driven Bean）。

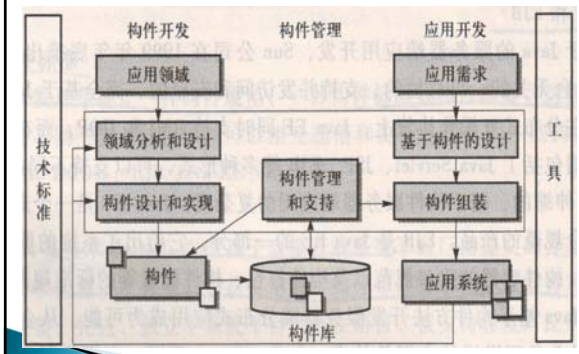
（4）SCA

服务组件架构（Service Component Architecture，简称SCA）

以上三种构件互操作机制具有紧耦合的特征，难适应Internet 开放、动态和多变的特征。

面向服务架构(Service-Oriented Architecture，SOA），SOA是一种粗粒度、松耦合服务架构，服务之间通过简单、精确定义接口进行通讯，不涉及底层编程接口和通讯模。它将帮助企业系统架构者以更迅速、更可靠、更具重用性架构整个业务系统。

SOA 和构件技术相结合，产生了服务组件架构（SCA), 目前业界主要的软件厂商大力推行的一种与实现语言无关的构件模型。组成系统的单元是粗粒度的服务构件，实现一个完整业务功能服务的单位。在系统建模阶段，系统模型中的SCA 构件实例与具体实现技术无关，是抽象概念的构件。



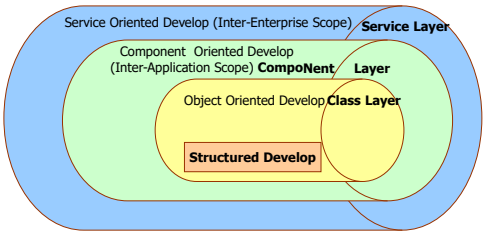
基于构件的软件开发方法

（1）构件开发
- 领域分析与设计
- 构件设计与实现
- 构件入库

（2）构件管理
- 组织、描述、分类、检索构件
- 管理与维护构件库
- 评估构件资产，不断改进

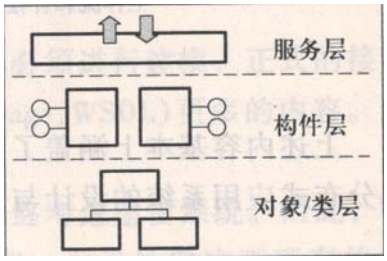
（3）应用开发与组装
- 基于构件的设计（基于构件的应用系统体系结构）
- 构件组装
- 系统测试与发布

2.4 面向服务的方法（Service Oriented Method）



SOD<-COD<-OOD<-SD

服务、构件和类之间的关系



构件是业务实体的映射，服务是业务规划的映射。

服务的特征

- **服务的抽象性（基于接口的编程）**
  - 服务是实际程序、数据库、业务过程等软件实体的抽象了的逻辑视图。
  - 实现平台透明性
- **服务的自治性（实现分布式应用）**
  - You don't "new" a service – it's just there.
- **服务间的松耦合式绑定，基于标准化消息进行通信**
- **服务的自描述性（支持动态发现与延迟绑定）**
  - 服务具有可发布、可发现、机器可处理的接口契约
- **服务的粗粒度（支持基于业务逻辑的积木式装配）**

SOA 模型

SOA架构基本的要求：
（1）相对较粗的粒度上对应用服务或业务模块进行封装与重用；
（2）服务间保持松散耦合，基于开放的标准， 服务的接口描述与具体实现无关；
（3）灵活的架构 -服务的实现细节，服务的位置乃至服务请求的底层协议都应该透明；

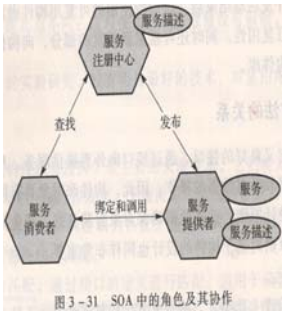


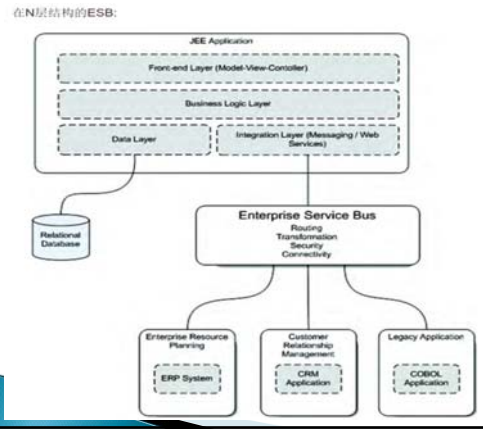
图 3-31 SOA 中的角色及其协作

企业服务总线（Enterprise Service Bus，ESB）

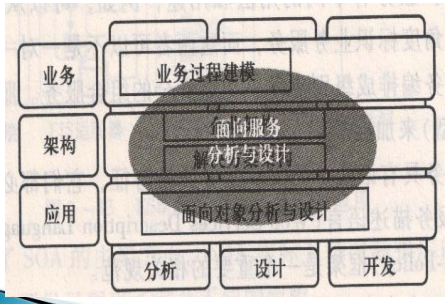
在SOA中还需要一个中间层，能够帮助实现在SOA架构中不同服务之间的智能化管理。这就是企业服务总线.

ESB的主要功能是:

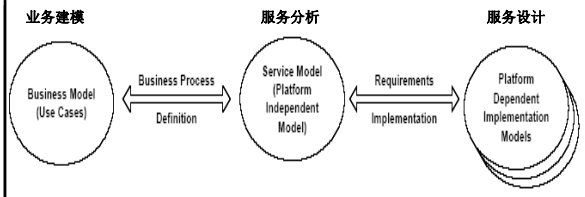
1. **对各个服务之间消息监控与路由**
2. **解决各个服务组件之间通信**
3. **控制服务版本与部署**
4. **满足服务像事件处理,数据转换与映射,消息与事件查询与排序,安全或异常处理,协议转换,保证服务通讯的质量.**



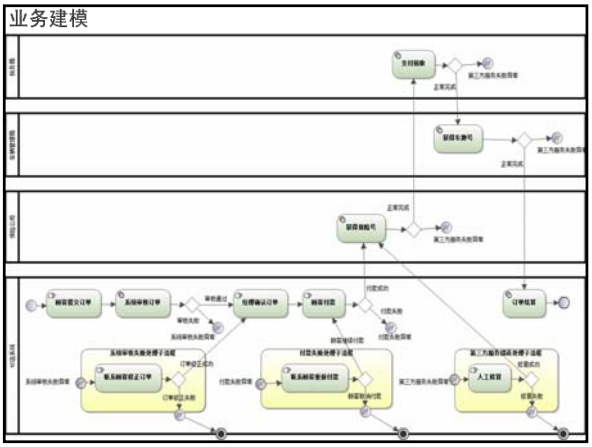
SOAD与现有建模方法之间的关系



面向服务软件开发SOAD



业务建模



服务分析－step1

- 服务识别
  - **自顶向下分解**，由业务需求驱动，设计服务。如果没有已有的软件服务相对应，就需要设计一个全新的服务。
    - 优点：业务流程中的业务服务和软件服务之间的交互是完全匹配的；
    - 缺点：可能对现有软件服务的利用率不高，因为识别出来的业务服务有可能与现有软件服务之间不能完全匹配。
  - **自底向上抽象**，以现有服务驱动进行匹配
    - 优点：通过不断地将业务服务与已有的软件服务进行匹配，使得对已有软件服务能够进行最大限度的复用；
    - 缺点：可能会发现很多业务服务无法找到合适的现有服务进行匹配，导致需要新开发的服务数量变多。

中间汇合，两头并行设计

服务分析－step2

- 服务粒度的确定
  - 以对业务的精确掌握和理解为基础，在性能、可维护性和随需应变之间进行平衡，选择适合的粒度进行服务的识别和划分。
    - 服务粒度太细会使得服务流程过于复杂，而且服务流程执行的效率也会降低很多；
    - 服务流程太粗会使得服务流程难以改动，从而使得随需应变无法实现。
  - 需要我们能够预判业务流程在将来有可能出现的变化，然后根据这种预判来设计服务的粒度。

服务分析－step3

- 服务的组织
  - 编制
    - 在流程中有一个作为流程中心的服务，它负责控制流程中其他参与交互的服务，协调服务间的消息传递。
    - 除流程中心服务之外的其他参与交互的服务。
  - 编排
    - 参与业务过程的Web服务之间是协作关系，为了实现一个业务流程，每一个参与其中的服务都需要明确地知道自己应该在什么情况下与哪些其他服务进行交互。

面向服务建模所涉及的三个层面：

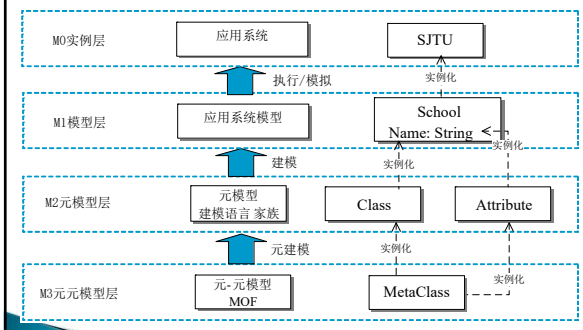
（1）业务层面
　　功能域：业务逻辑描述
　　业务流程：业务逻辑编排
　　业务服务：识别服务，产生业务服务集
（2）服务层面
　　根据业务流程和业务服务集设计软件服务集。
（3）构件层面
　　服务最终由构件实现的，在特定的开发语言与框架中进行设计，可将多个服务中的共用部分抽取出来构成通用的工具构件。
　　通过面向服务的架构将多个相关但独立开发的应用系统集成。
　　从底层构件层面向上，并从顶层的业务层面向下，最终在软件服务和业务服务的中间点会合，得到完整的设计方案。

2.5 模型驱动开发

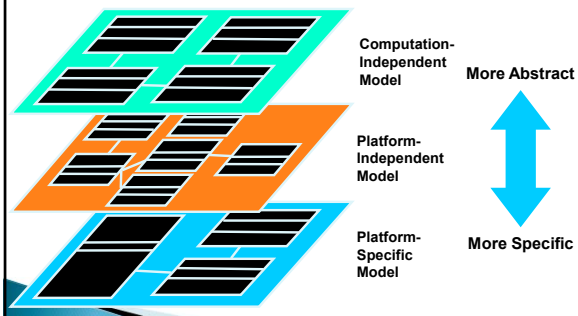
软件生命周期以模型为载体并由模型转换来驱动的过程

- 将软件的开发集中于模型而不是程序(代码)的一种开发方法
  - 从模型自动产生程序(代码)
  - 主要用来提高开发生产率和代码的可靠性
- OMG 定义模型驱动的体系结构(Model-Driven Architecture，简称MDA)的初衷
  - 定义一组标准来支持模型驱动的开发
  - UML 是 MDA 的基础之一

模型和元模型

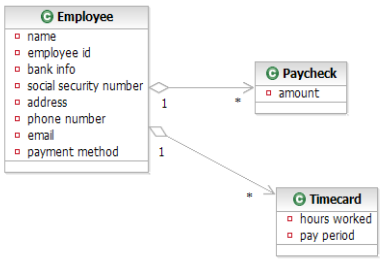


MDA中的三种模型



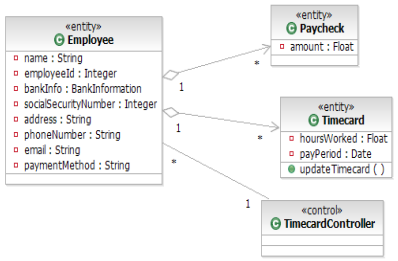
计算无关模型 (CIM) 举例

- Uses the vocabulary of the domain.
- No information in the model indicates that a computer-based solution will be used.



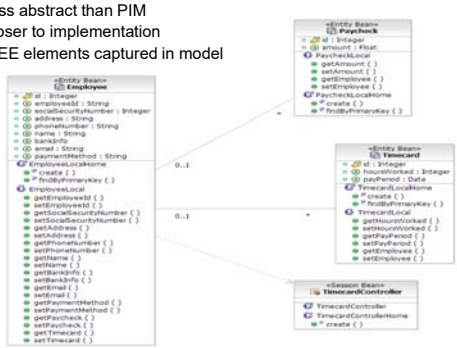
平台无关模型 (PIM) 举例

- Less abstract than CIM
- Closer to implementation but not tied to a platform.

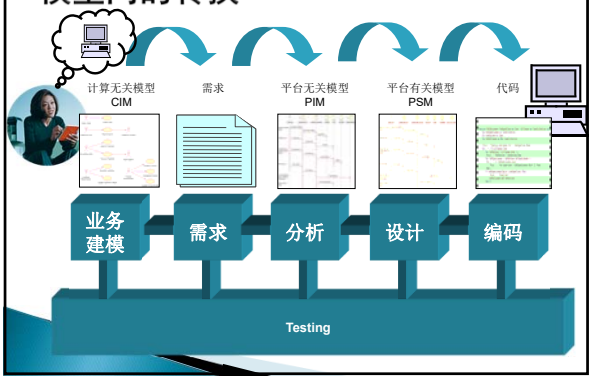


平台有关模型 (PSM) 举例

- Less abstract than PIM
- Closer to implementation
- J2EE elements captured in model

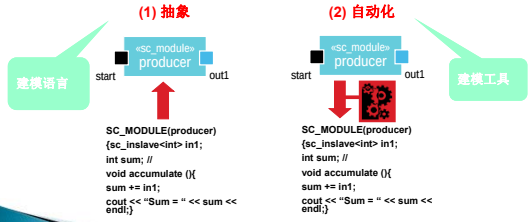


模型间的转换



模型驱动的软件开发 (MDD)

- 以模型为中心（相对于以代码为中心）的软件开发方法
- 基于以下两种久经考验的技术



MDD主要方法

1. 可执行的UML（xUML）
   - xUML =UML－语义较弱的元素+精确定义的动作语义
2. 从PIM到PSM或代码的（半）自动生成
   - 通过标记、profile、OCL或者其他途径进行平台无关模型（PIM）的刻画，然后定义相关的转换规则来完成从平台无关模型到平台相关模型或代码的转换。
3. 基于DSL的开发方法
   - 领域描述语言（Domain Specific Language，DSL）通过适当的表示方法和抽象机制来提供对特定问题领域的表述能力，DSL 常用来生成特定应用领域中的一个产品 / 应用家族的应用。

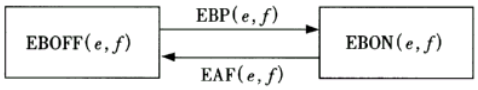
OCL（Object Constraint Language），主要用于UML模型上施加的约束。

2.6 形式化方法

- 形式化方法是基于数学的技术开发软件，如集合论、模糊逻辑、函数、有限状态机、Petri-net等。
- 形式化方法的好处：
  - 无二义性
  - 一致性
  - 正确性
  - 完整性

举例：电梯软件 ——有限状态机

1）电梯按钮的状态及其转换规则



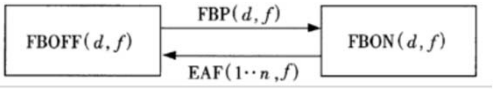
电梯按钮的状态转换图

- 令EB(e,f)表示按下电梯e内的按钮并请求到f层去。EB(e,f)有两个状态，分别是按钮发光(打开)和不发光(关闭)。更精确地说，状态是：
  - EBON(e,f)：电梯按钮(e，f)打开
  - EBOFF(e,f)：电梯按钮(e，f)关闭

- 如果电梯按钮(e,f)发光且电梯到达f层，该按钮将熄灭。相反如果按钮熄灭，则按下它时，按钮将发光。
  - EBP(e,f)：电梯按钮(e,f)被按下
  - EAF(e,f)：电梯e到达f层
- 为了定义与这些事件和状态相联系的状态转换规则，需要一个谓词V，(e,f)，它的含义如下：

  V(e,f)：电梯e停在f层
- 如果电梯按钮(e,f)处于关闭状态，而且电梯按钮(e,f)被按下〔事件〕，而且电梯e不在f层〔谓词〕，则该电梯按钮打开发光〔下个状态〕。状态转换规则的形式化描述如下：
  - EBOFF(e,f)+EBP(e,f)+not V(e,f) [JX*9]EBON(e,f)
- 反之，如果电梯到达f层，而且电梯按钮是打开的，于是它就会熄灭。这条转换规则可以形式化地表示为：
  - EBON(e,f)+EAF(e,f) [JX*9]EBOFF(e,f)

2）楼层按钮的状态及其转换规则

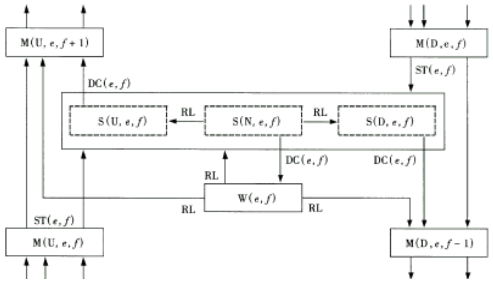


楼层按钮的状态转换图

- FB(d, f)表示f层的按钮请求电梯向d方向运动
- 楼层按钮的状态如下：
  - FBON(d, f)：楼层按钮(d, f)打开；
  - FBOFF(d, f)：楼层按钮(d, f)关闭。

- 如果楼层按钮已经打开，而且一部电梯到达f层，则按钮关闭。反之，如果楼层按钮原来是关闭的，被按下后该按钮将打开。
  - FBP(d, f)：楼层按钮(d, f)被按下
  - EAF(1‥n, f)：电梯1或…或n到达f层
- 谓词S(d, e, f)：电梯e停在f层并且移动方向由d确定为向上(d=U)或向下(d=D)或待定(d=N)。
- 形式化转换规则为：
  - FBOFF(d, f)+FBP(d, f)+not S(d, 1‥n, f) [JX*9]FBON(d, f)
  - FBON(d, f)+EAF(1‥n, f)+S(d, 1‥n, f) [JX*9]FBOFF(d, f)
  - 其中，d=UorD。

3）电梯的状态及其转换规则



- M(d,e,f)：电梯e正沿d方向移动，即将到达的是第f层；
- S(d,e,f)：电梯e停在f层，将朝d方向移动(尚未关门)；
- W(e,f)：电梯e在f层等待(已关门)。

- 电梯关门的状态转换规则
  - S(U，e,f)+DC(e,f) [JX*9]M(U，e,f+1)
    - 如果电梯e停在f层准备向上移动，且门已经关闭，则电梯将向上一楼层移动。
  - S(D，e,f)+DC(e,f) [JX*9]M(D,e,f-1)
    - 电梯即将下降时
  - S(N,e,f)+DC(e,f) [JX*9]W(e,f)
    - 电梯没有待处理的请求时

形式化方法的不足

- 形式化规约主要关注于功能和数据，而问题的时序、控制和行为等方面却更难于表示。此外，有些问题元素(如，人机界面)最好用图形技术来刻划。
- 使用形式化方法来建立规约比其他方法更难于学习，并且对某些软件实践者来说它代表了一种重要的“文化冲击”。
- 难以支持大的复杂系统。

尚未成为主流的开发方法，实践和应用较少