第13章 架构设计

•**为何需要架构?**

•架构并非可运行的软件。
相反，它是一种能让
软件工程师做到以下几点的表示法：

•(1) 分析设计在满足
其规定的要求

•(2) 在进行设计变更仍相对容易的阶段考虑架构替代方案
并且

•(3) 降低与软件构建相关的风险。

•为什么架构很重要?

•软件架构的表示是实现以下目标的推动因素
所有相关方（利益相关者）之间的沟通
涉及基于计算机的系统的开发。

•该架构凸显了早期的设计决策，这些决策将
对后续的所有软件工程工作产生深远影响，同样重要的是，对系统作为一个可运行实体的最终成功也会产生深远影响。

•架构“构成了一种相对较小、易于从智力上理解的模式，
展示了系统的结构以及其组件如何协同工作” [BAS03]。

•架构“构成了一种相对较小、易于从智力上理解的模式，展示了系统的结构以及其组件如何协同工作” [BAS03]。

•**架构描述**

•电气与电子工程师协会（IEEE）计算机协会提出了IEEE - Std - 1471 - 2000《软件密集型系统架构描述推荐实践》[IEE00]
•以建立一个在软件架构设计过程中使用的概念框架和词汇表

•为软件架构设计期间的使用建立概念框架和词汇表
•在软件架构设计方面

•为表示架构提供详细指南
描述，以及

•鼓励合理的架构设计实践。

•IEEE标准将架构描述（AD）定义为“记录架构的一组产品”。 - 描述本身使用多个视图来表示，其中每个视图是“从一组相关的[利益相关者]关注点的角度对整个系统的表示”。

•架构类型

•类型意味着整个软件领域内的特定类别。

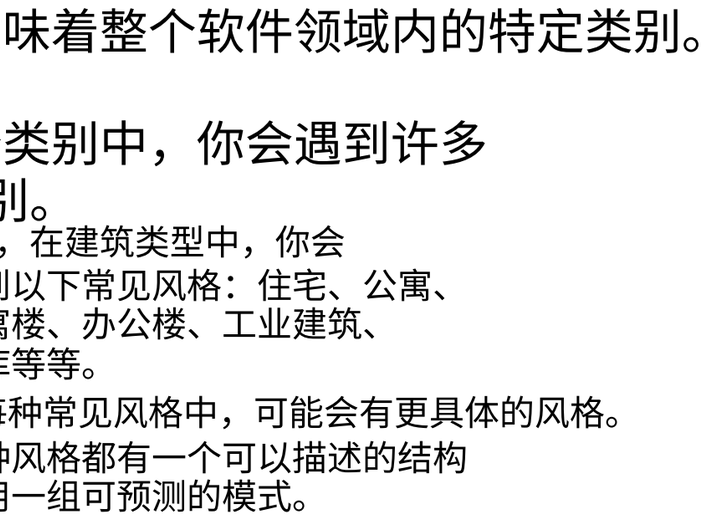
•在每个类别中，你会遇到许多
子类别。
•例如，在建筑类型中，你会
遇到以下常见风格：住宅、公寓、
公寓楼、办公楼、工业建筑、
仓库等等。
– 在每种常见风格中，可能会有更具体的风格。
每种风格都有一个可以描述的结构
使用一组可预测的模式。

• **架构风格**

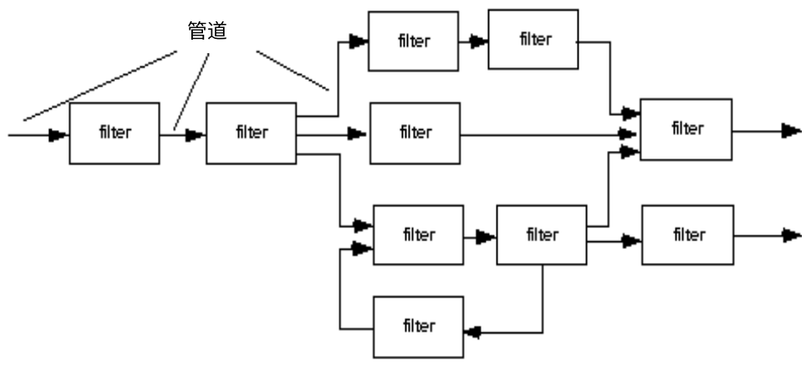
• 每种风格描述了一个系统类别，该类别包括：（1）一组执行系统所需功能的组件（例如，数据库、计算模块）；（2）一组使组件之间能够“通信、协调与合作”的连接器；（3）定义如何

将组件集成以形成系统的约束条件；以及（4）
使设计师能够理解的语义模型
通过分析已知属性来确定系统的整体属性
其组成部分的属性。
•以数据为中心的架构
•数据流架构 - 调用和返回架构
- 面向对象架构 - 分层架构

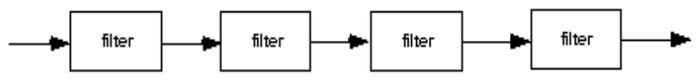
•以数据为中心的架构



•数据流架构

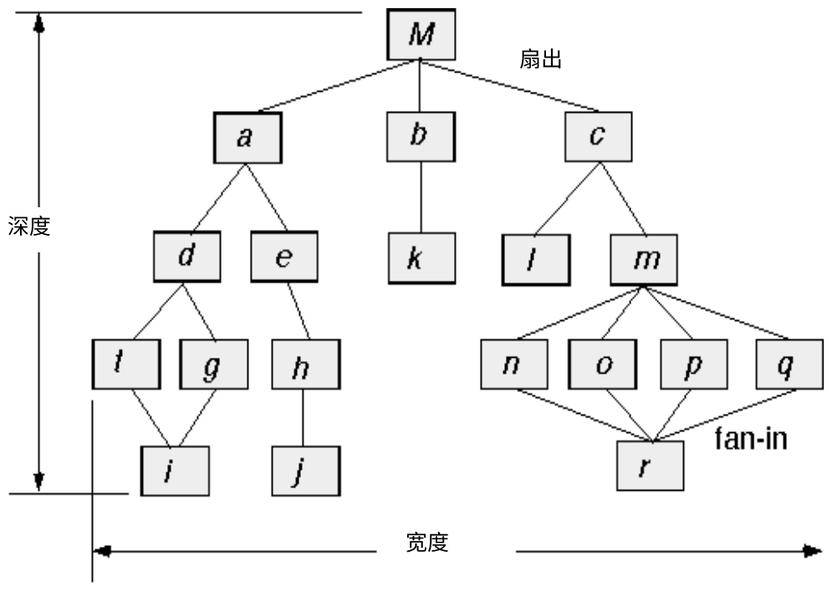


(a) 管道与过滤器

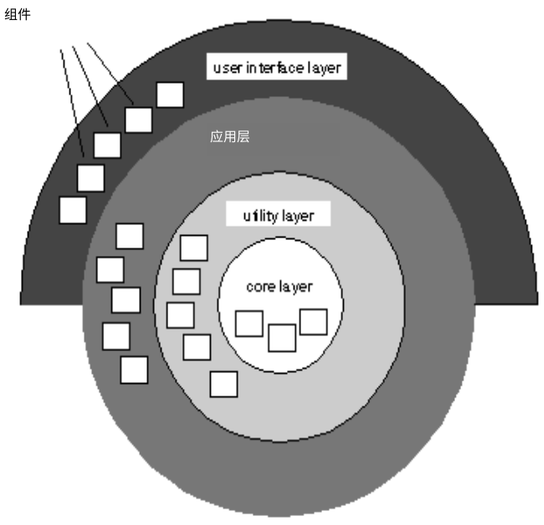


(b) 批处理顺序

- 调用与返回架构



- 分层架构



•架构模式

•并发——应用程序必须以模拟并行性的方式处理多个任务

模拟并行性的方式

•操作系统进程管理模式

•任务调度器模式

•持久性——如果数据在创建它的进程执行结束后仍然存在，则称其具有持久性。常见的模式有两种：

进程创建的数据在执行结束后仍能留存。常见的有两种模式：

•数据库管理系统模式，将数据库管理系统的存储和检索功能应用于应用程序架构

•将数据库管理系统的存储和检索功能应用于应用程序架构的数据库管理系统模式

•应用程序级持久性模式，将持久性特性构建到应用程序架构中

•将持久性特性融入应用程序架构的应用程序级持久性模式

- 分布——系统或组件的分布方式

在分布式环境中，系统内各组件之间相互通信的方式

环境

•代理充当客户端组件和

服务器组件之间的“中间人”。

架构设计

•软件必须置于特定环境中

•设计应明确软件与之交互的外部实体（其他系统、

设备、人员）以及交互的性质

•交互

•应确定一组架构原型

•原型是一种抽象（类似于类），它代表

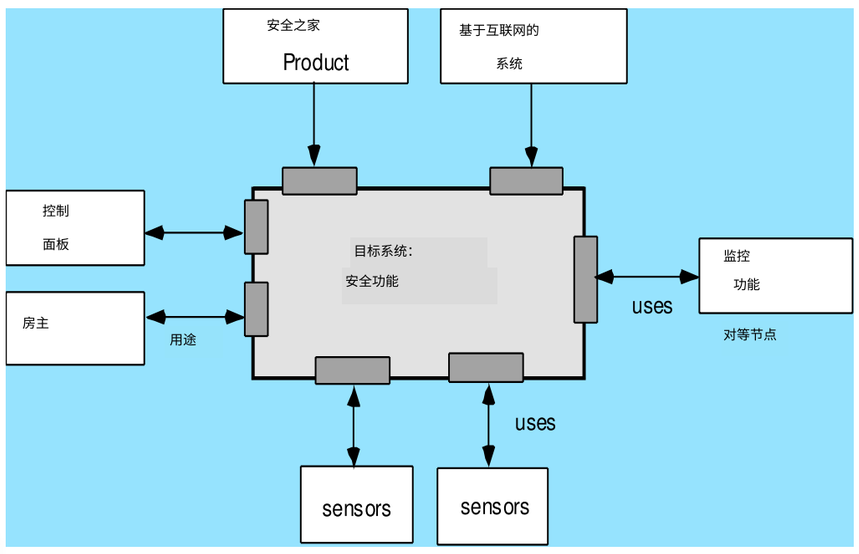
系统行为的一个要素

•设计师通过定义来指定系统的结构

并细化实现每个

原型

•架构上下文



- 原型

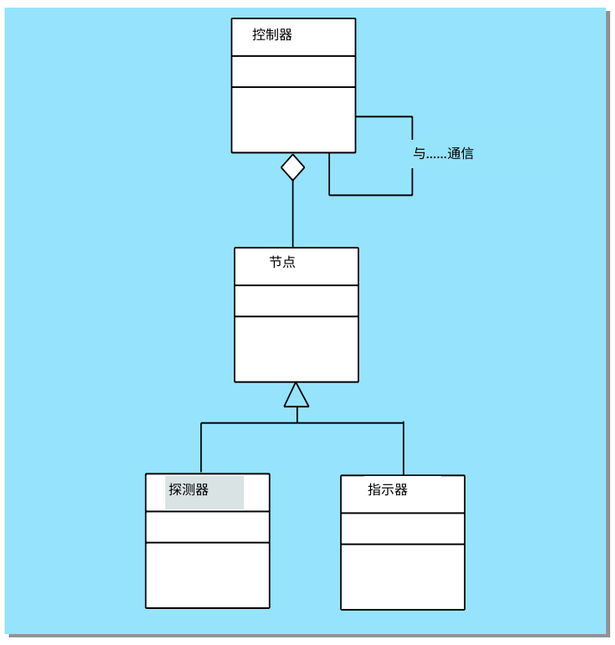
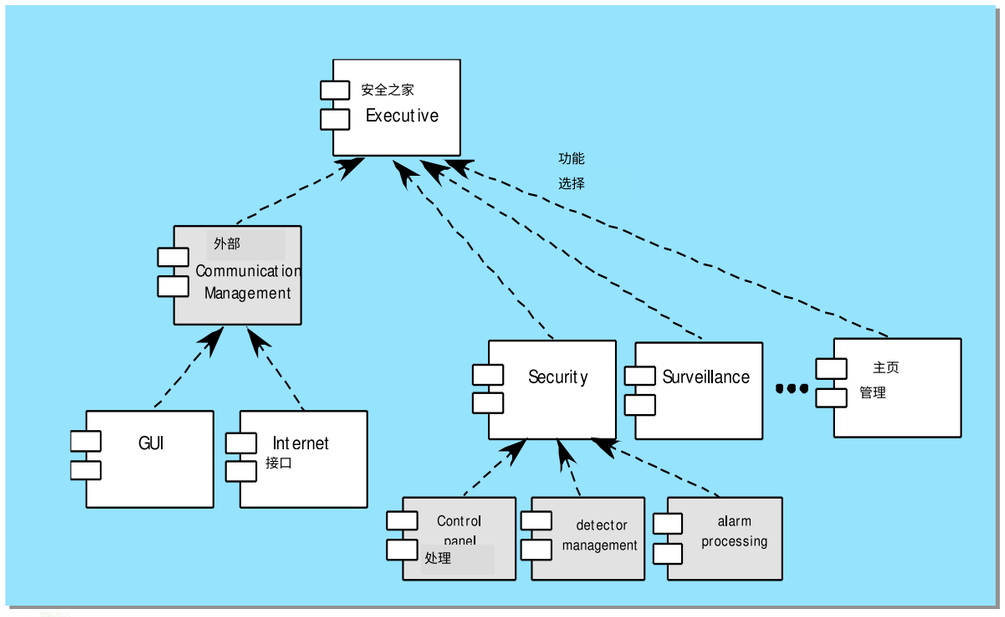
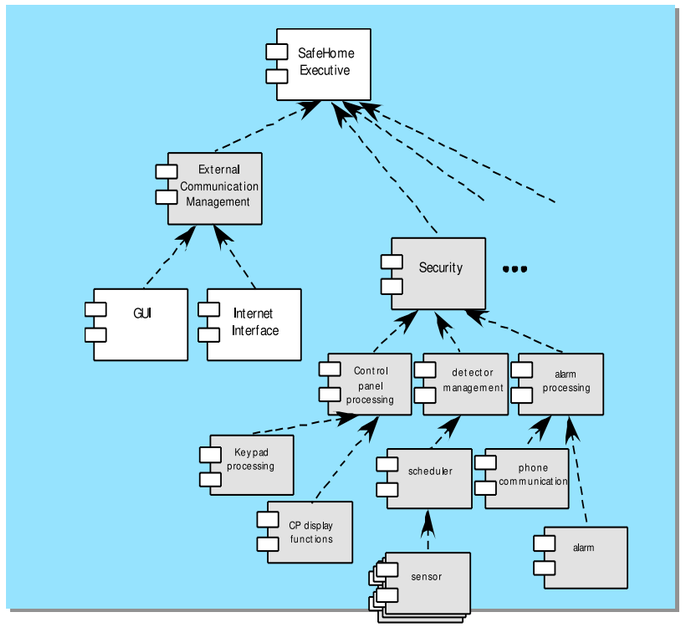


图10.7 安全家庭安全功能原型的UML关系
（改编自 [BOS00]）

- 组件结构



•细化的组件结构



•架构考量

•经济性——最佳的软件应简洁明了，并依靠抽象来减少不必要的细节。
•可见性——架构决策及其原因应让后续审查模型的软件工程师一目了然。

•隔离性——在设计中分离关注点，且不引入隐藏的依赖关系。
•对称性——架构对称性意味着系统在其属性上是一致且平衡的。 - 涌现性——涌现的、自组织的行为和控制。

•架构决策文档

•确定每个决策所需的信息项。

•定义每个决策与适当需求之间的关联。

•当需要评估替代决策时，提供更改状态的机制。

•定义决策之间的先决条件关系以支持可追溯性。

•将重要决策与决策产生的架构视图关联起来。 - 记录并传达所有已做出的决策。

•架构权衡分析

•收集场景。
•获取需求、约束条件和环境描述。
•描述为应对场景和需求而选择的架构风格/模式：

•模块视图
•进程视图
•数据流视图

•通过单独考虑每个属性来评估质量属性。- 针对特定的架构风格，确定质量属性对各种架构属性的敏感性。- 使用步骤5中进行的敏感性分析来评判候选架构（步骤3中开发的）。

•架构复杂性

•通过考虑架构内组件之间的依赖关系来评估所提议架构的整体复杂性[Zha98]

•共享依赖表示使用相同资源的消费者或为相同消费者生产的生产者之间的依赖关系。- 流依赖表示资源生产者和消费者之间的依赖关系。- 约束依赖表示一组活动之间相对控制流的约束。

- ADL

•架构描述语言（ADL）为描述软件架构提供了语义和语法

•为设计师提供以下能力：
•分解架构组件
•将单个组件组合成更大的架构块并

•表示组件之间的接口（连接机制）。

•架构评审

•评估软件架构满足系统质量要求的能力并识别潜在风险

•有潜力通过早期发现设计问题来降低项目成本

•经常利用基于经验的评审、原型评估、场景评审和检查表

•基于模式的架构评审

•通过遍历用例来识别和讨论质量属性。

•结合系统需求讨论系统架构图。

•识别所使用的架构模式，并将系统结构与模式结构进行匹配。
•利用现有文档和用例来确定每个模式对质量属性的影响。

•识别设计中使用的架构模式引发的所有质量问题。

•简要总结会议中发现的问题，并对可运行框架进行修订。

•敏捷性与架构

•为避免返工，在编码前使用用户故事来创建和演进架构模型（可运行框架）

•允许软件架构师为不断演进的故事板贡献用户故事的混合模型
•运行良好的敏捷项目包括在每个冲刺阶段交付工作产品
•评审冲刺阶段产出的代码是一种有效的架构评审方式