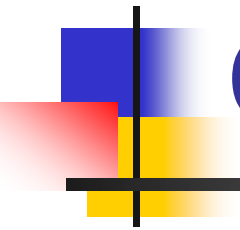
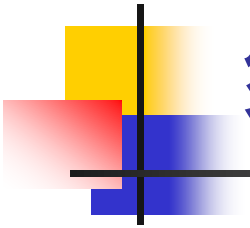


Chp2 Architectural Styles



纲要

- Architectural Styles的作用
  - Specific organizational principles and structure for certain classes of software
- 几种常见的体系结构风格及其优缺点
  - Pipes and filters
  - ADT and OO
  - Event-Based
  - Layered
  - Repositories
  - Interpeters
  - Process-Control

2.1 Architectural Styles

- 体系结构风格 → 模式系统中的词汇
  - 目前尚不完善
  - 每个风格可以视为一组部件的集合，以及部件间的交互（连接器）

    部件（Components）＋ 连接器（Connectors）
  - E.g. C/S结构中
    - 部件： Client, Server
    - 连接器： C/S间的通讯协议

A List of Common Architectural Styles

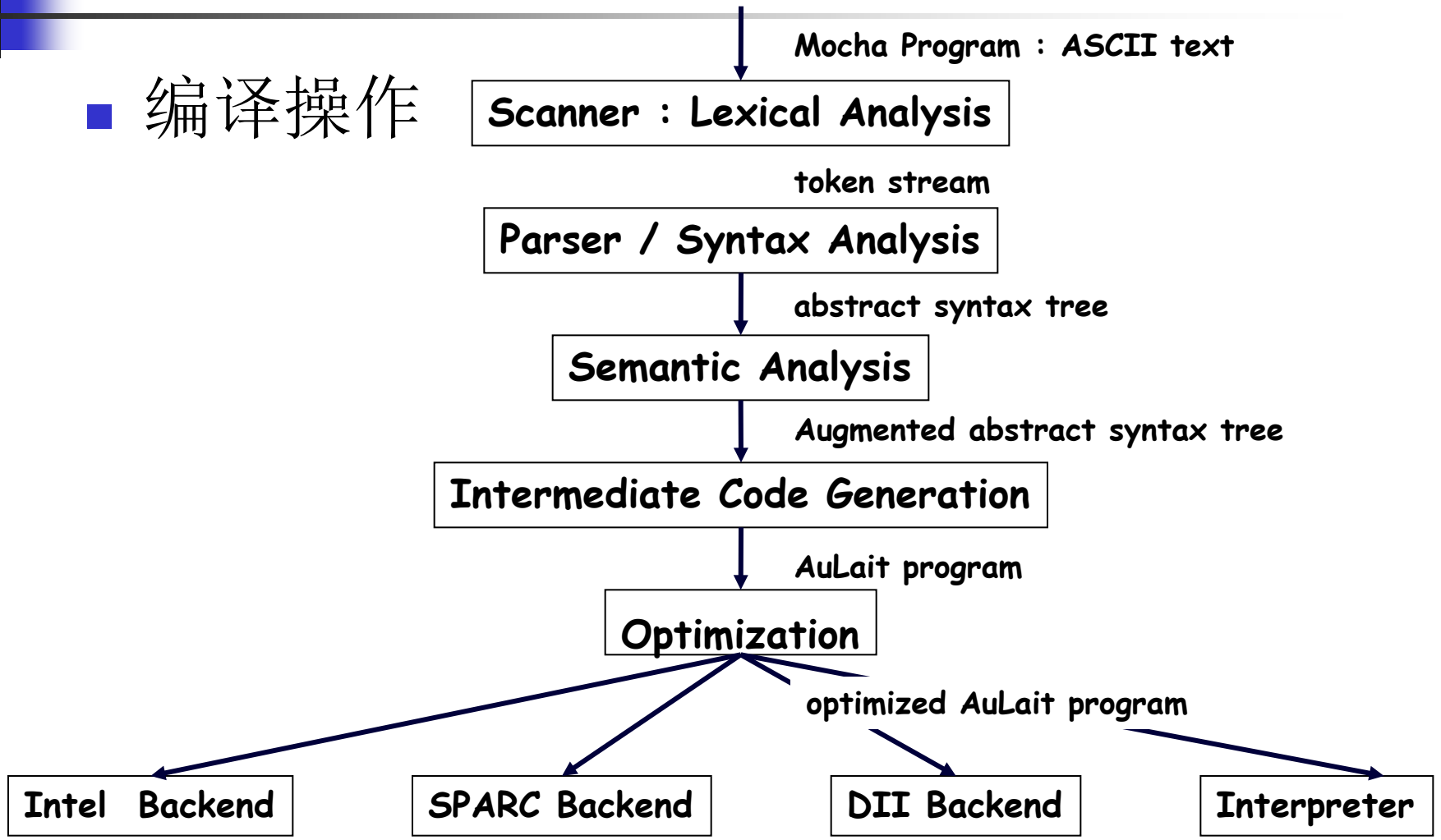
Dataflow systems Batch Sequential Pipes and filters	**Virtual machines** Interpreters Rule-based systems
Call-and-return systems Main program and subroutines OO Systems Hierarchical Layers	**Data-Centered systems(repositories)** Databases Hypertext systems Blackboards
Independent components Communicating processes Event systems	

2.2 Pipes and Filters

- 适用环境
  - 适用于对数据流的处理
  - The pipe and Filter architecture pattern provides a structure for systems that *process a stream of data*
- 组成部分
  - Components are filters 过滤器
    - each encapsulated a processing step
    - transform input data streams into output data streams
    - possibly incremental production of output
  - Connectors are pipes 管道
    - conduits for data streams between adjacent filters .
- *Recombining filters* allows you to build families of related systems .

使用示例

- 编译操作

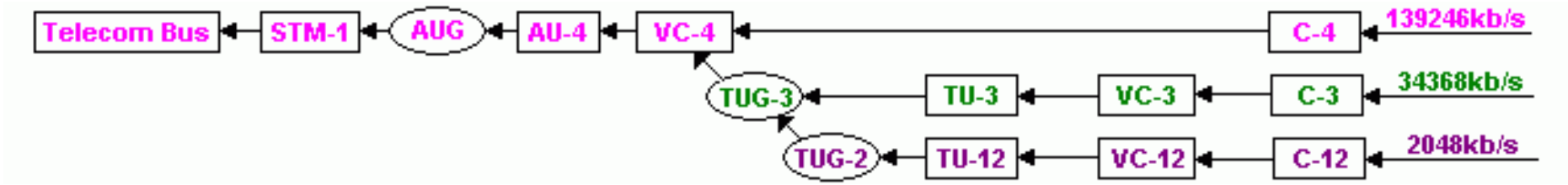

Mocha Program : ASCII text
Scanner : Lexical Analysis
token stream
Parser / Syntax Analysis
abstract syntax tree
Semantic Analysis
Augmented abstract syntax tree
Intermediate Code Generation
AuLait program
Optimization
optimized AuLait program
Intel Backend
SPARC Backend
DII Backend
Interpreter

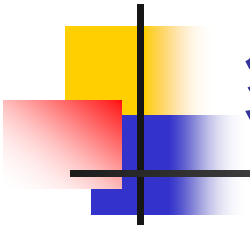
使用示例

- Unix系统中的管道过滤器结构

  ls –al | grep my

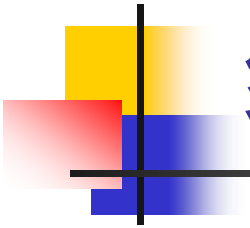
- 通讯协议的信息封装(e.g. SDH)





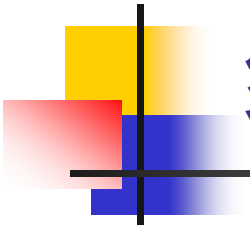
结构

- 过滤器的作用：对输入数据的处理
  - enriches : computing and adding info
  - refines : concentrating or extracting info
  - transforms : delivering data into some other representation
- 被动式过滤器（Passive filter）
  - adjacent pipes pulls/pushes output/input data from/into the filter
  - active either as a function (pull) or as a procedure (push)
- 主动式过滤器（Active filter）
  - filter is active in a loop , check the pipes for data
  - processing on its own as a separate program or thread



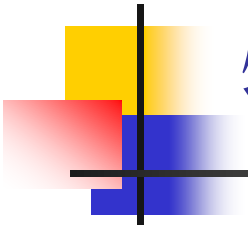
结构

- Data Source （数据源）
  - input data stream to the system , for example
    - A file consisting of lines of text
    - A sensor delivering a sequence of numbers
  - data can be pushed or pulled into first processing stage
- 管道 Pipes
  - connections between filters , between data source and the first filter , between the last filter and the data sink
  - synchronizes joined active filters , for example , by a FIFO (first-in-first-out) buffer
  - for passive filters , the pipes can be implemented by a direct call
    - Make the filter recombination harder
- Data Sink （数据池）
  - consumes output data



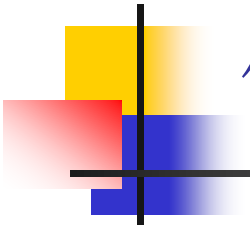
结构

- 类型
  - *pipelines* — linear sequences of filters
    将过滤器严格限制为单输入、单输出的类型
  - *bounded pipes* — limited amount of data on a pipe
  - *typed pipes* — data strongly typed
  - *batch sequential* — data streams are not incremental



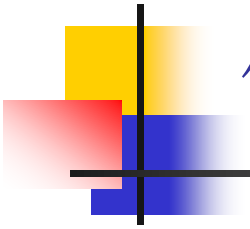
特性

- 过滤器是独立运行的部件
  - 非临近的过滤器之间不共享状态
  - 过滤器自身无状态
- 过滤器对其处理上下连接的过滤器“无知”
  - 对相邻的过滤器不施加任何限制
- 结果的正确性不依赖于各个过滤器运行的先后次序
  - 各过滤器在输入具备后完成自己的计算。完整的计算过程包含在过滤器之间的拓扑结构中。



优缺点

- 优点
  - 可以进行重组
  - 更换和添加部件，从而进行维护和升级
  - 部件的重用性好
  - 支持快速原型系统的设计实现
  - 自然的并发特性
  - 拓扑结构清晰，容易进行性能方面的分析



优缺点

- 缺点
  - 共享状态信息的代价高而且不灵活
  - 不适应交互式应用系统的设计和运行
  - 需要进行数据格式的设计和转换；需要对数据的句法或语义进行分析
  - 通过并行运行获得高运行效率往往行不通
    - 独立运行的部件间的数据传送的效率低
    - 过滤器通常在消耗所有的输入后才产生输出
    - 在单CPU上进程的切换代价高
    - 通过管道对过滤器进行同步控制，可能导致频繁启动和停止过滤器
  - 难以进行错误处理

2.2 Data Abstraction and Object-Oriented Organization

- 基本思路
  - 用对象（Object）来进行管理
    - Object：一个ADT的实例（Instance）
    - 封装了数据及其相关的基本操作
    - 相互之间通过函数或者过程调用完成交互
  - 特点
    - 一个Object需要保证其表示数据的完整性
    - 对数据的抽象对其他Object是不可见的

Fig 2.3 Abstract Data Type and Objects

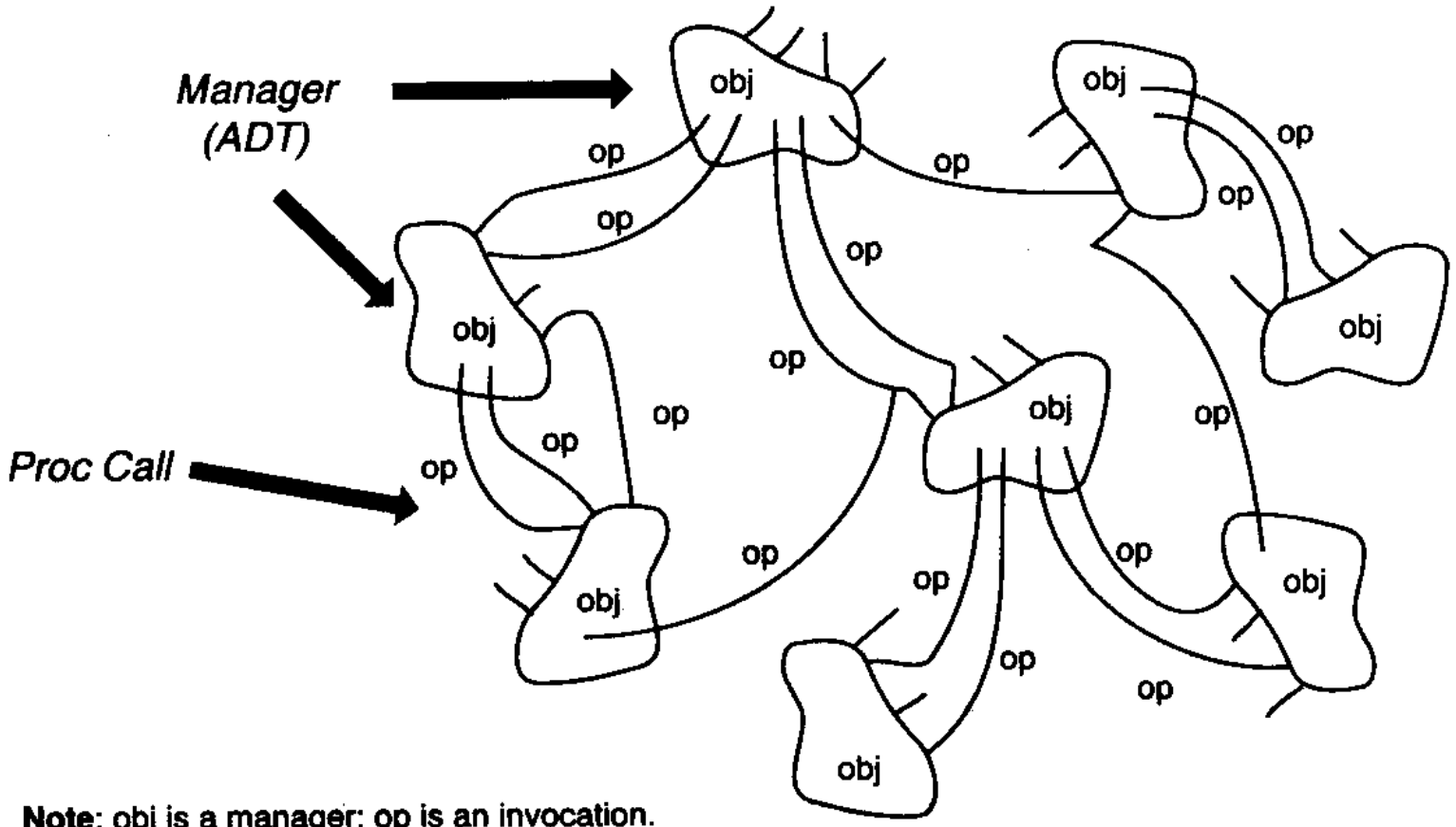
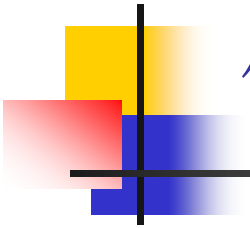


FIGURE 2.3 Abstract Data Types and Objects



优缺点

- 优点
  - 易修改（信息隐藏）
  - 便于设计者将问题分解为

    collections of interacting agents
- 缺点
  - 进行过程调用时，必须知道被调用者对象的名字
  - 一个对象的名字发生变化，需要修改所有相关联的对象
  - Side-effect ： A,B对象同时操作C时

    e.g. 购票系统

2.4 Event-Based, Implicit Invocation

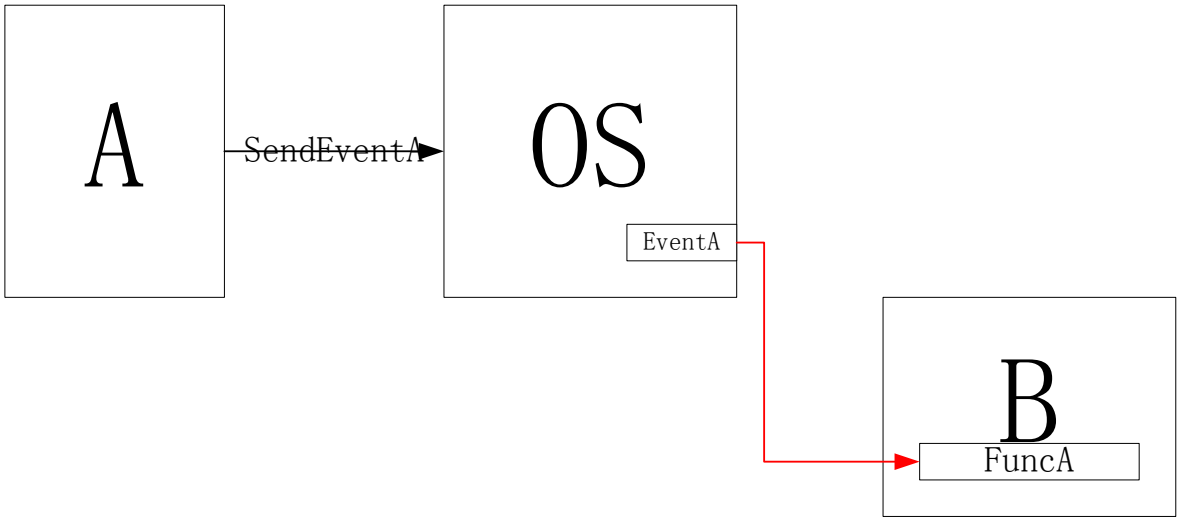
- 显式调用与隐式调用
  - 显式调用 e.g. wina->DoAct();
  - 隐式调用 e.g. SendMessage(…, …);

Implicit invocation

Reactive integration

Selective broadcast

基本思路



Step1：B向OS注册

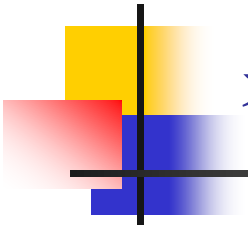
Step2：A向OS发出事件

Step3：OS根据注册信息找到需要执行的代码

e.g. 1. Debugger

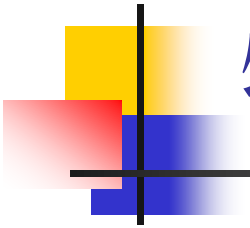
2. Windows环境中的消息处理（Message）

责任链



基本思路

- 在隐式调用风格中，部件的接口
  - A collection of procedures
  - A set of events
- 过程调用
  - 方式1：普通调用
  - 方式2：将过程与某个Event关联起来
    OS接收到此Event时，将自动触发该过程

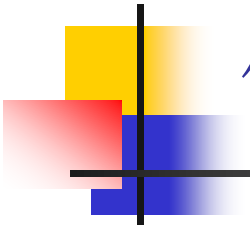


特点

- 事件的触发者对此事件的后果无知
  - 不知道那些部件会被影响到
  - 不能确定该事件被处理的过程

  不知道是否有人处理、不知道谁会处理、不知道怎样处理。

- 因此：仍然需要使用显式调用作为补充



优缺点

- 优点
  - 重用性好
  - 便于进行系统修改
    - 可以更换部件而不影响其他部件
- 缺点
  - 缺少控制
  - 如何进行数据交换（特别是共享数据处理）
  - 正确性判定困难

2.5 Layered Systems

- 基本思想
  - 按层次结构进行组织
  - 每一层向上层提供服务，同时是下层的Client
  - 与Virtual machine的关系
  - 按照规定的协议进行层之间的交互，这种交互只发生在相邻层之间

Fig2.4 Layered Systems

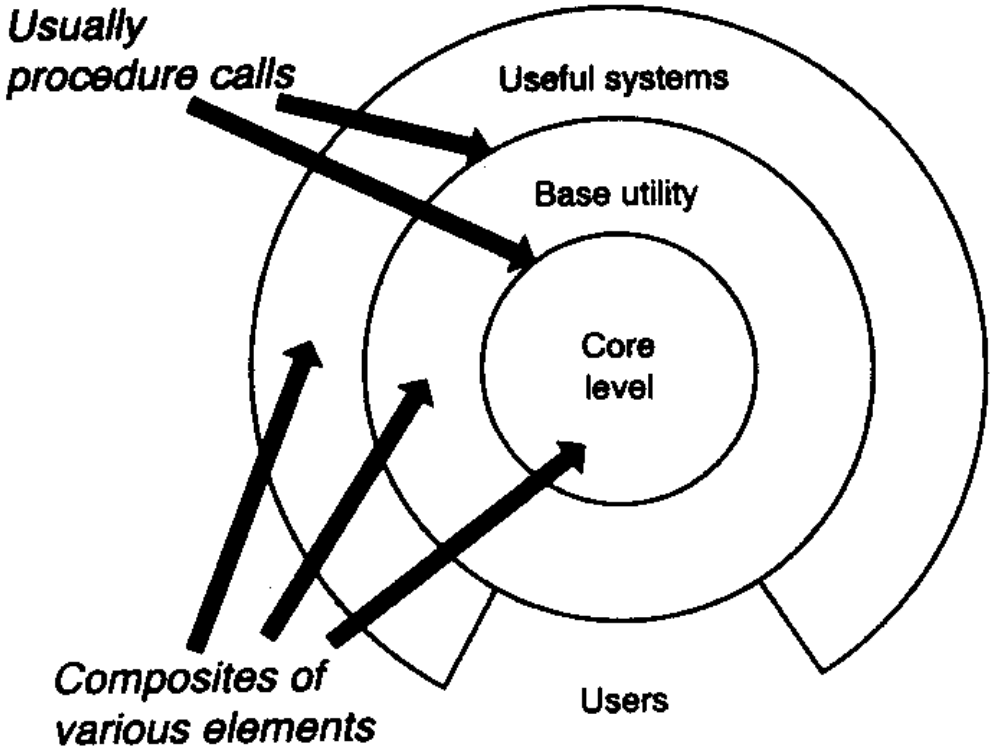
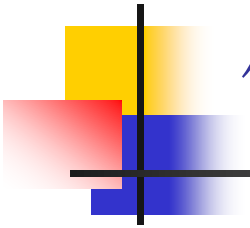


FIGURE 2.4 Layered Systems



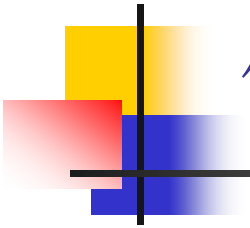
优缺点

- 优点
  - 方便按照递增抽象层次进行设计

    e.g. 对具体硬件设备的逐步抽象层次化
  - 便于修改（修改一层最多影响上下两层）
  - 支持重用

    e.g. OSI层次

    显卡驱动



优缺点

- 缺点
  - 有的系统无法用层次模型表达
  - 层次与效率

    e.g. GUI vs DirectX

  - 难于确定right levels of abstraction

    e.g. OSI vs TCP/IP

2.6 Repositories 仓库

- 仓库系统的构成
  - Central data structure
  - 一组独立部件的集合（用来对核心数据进行操作）

- 分类
  - 输入流触发动作　　database
  - 当前的数据状态触发动作　　blackboard

黑板系统 Blackboard

- 适用问题
  - for problems without deterministic solution strategies
  - several knowledge source assemble to build a approximate or possibly partial solution
- 限制条件
  - complete search of solution space not feasible
  - need experiment with different algorithms for same subtask
  - different algorithms to solve partial problems
  - different representation for data
  - an algorithm usually works on the results of other algorithms
  - uncertain data & approximate solutions involved
  - employing disjoint algorithms includes potential parallelism

解决方法（Solution）

- 基本思想
  - 独立程序集合
    - 对公共数据结构进行协同操作
    - 每个程序专门解决一个子问题
    - 不存在互相调用
    - 不存在事先确定的操作顺序
  - 中心控制部件（所谓的“黑板”）
    - 反映整体问题求解过程的状态
    - 状态变化时，中心控制部件对信息进行评价，协调各程序工作
    - 可以试探调用各个可能的求解算法

结构

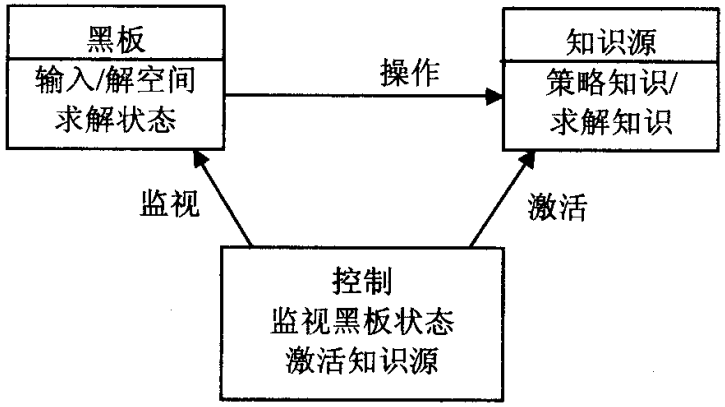


图 11.15　黑板体系结构关系

- 黑板部件
  - Central data store
  - Interface that enable knowledge source to read/write data
  - Hypothesis / blackboard entry : elements of solution space constructed by knowledge source , removed if rejected by further KS
- 知识源
  - Separate independent subsystems to solve specific aspects of the overall problem
  - Read / write blackboard
  - Condition part + action part
- 控制部件
  - Runs a loop : monitors changes on blackboard / decides the next action (use of knowledge source)
  - Heuristics could be used as control knowledge

示例

- HEARSAY-II
  - 第一个黑板系统结构的应用，early 1970s
  - 文献数据库的自然语言接口系统
    - To answer queries (as acoustic signals) about documentations
    - Docs retrieved from a collection of abstracts of AI publications
  - Generate-&-Test strategy：由知识源产生假想，然后由另外的知识源进行评价和证明。

- HASP/SIAP：
  - 探测敌方潜水艇
  - 输入：sonar signals collected by hydrophone arrays
  - 事件驱动：new information available
  - Blackboard as ‘situation board’ evolving over time : infor collected continuously

黑板结构的变体

- 产生式系统（Production System）
  - 知识源
    - 结构： “条件 – 结论/操作”的推理规则集合
    - 针对不同问题或方面，可定义和使用不同的知识源
  - 冲突裁决（Conflict Resolution）部件
    - 用于多条推理规则的条件同时满足时
  - 系统控制策略
    - 正向推理： 从输入出发，选择条件被满足的规则进行推理
    - 反向推理： 从假设触发，根据规则验证规则条件的满足
    - 同时进行
  - 规则的满足
    - 可采用“模糊”或近似
    - 推理规则中可定义使用模糊推理的计算参数

2.7 Interpreters
解释器 / 虚拟机

- Interpreters
  - 编译与解释

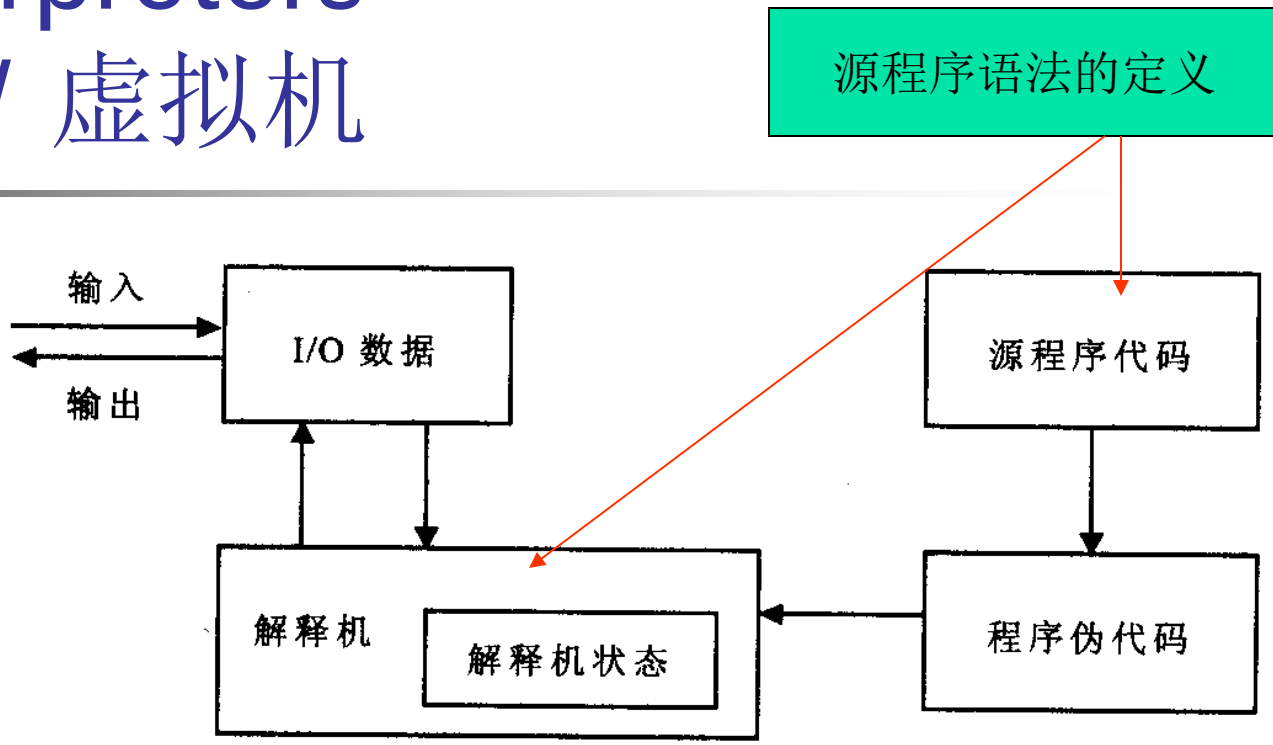


图 11.16　解释器构成

  - 组成
    - 源程序代码处理：编辑或产生源程序代码
    - 程序伪代码处理：将源代码转换成中间代码
    - 解释器输入输出控制：向解释器提供输入 / 输出
    - 解释机：读入中间代码，进行分析并执行

工作机制

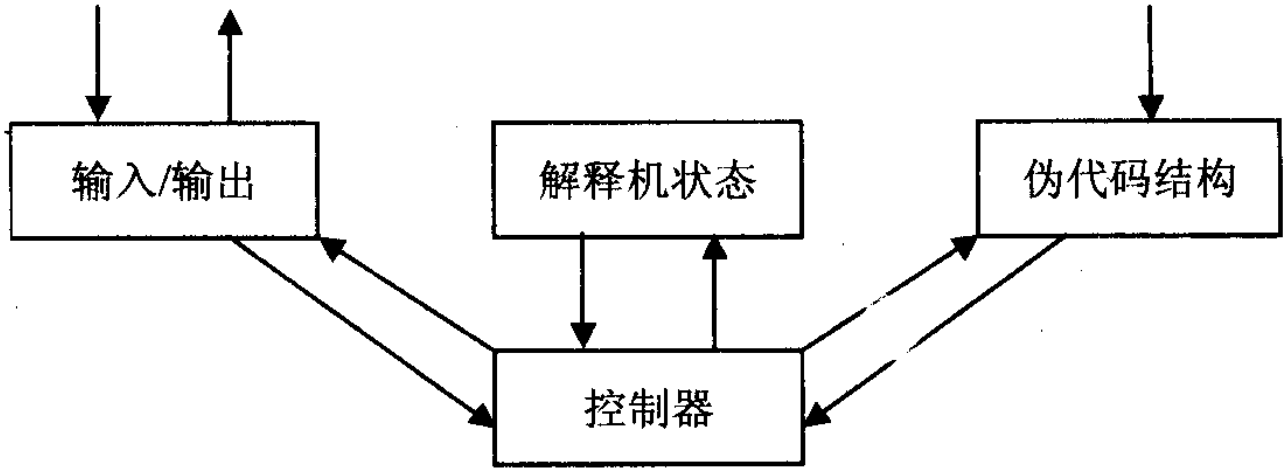


图 11.17　解释机的工作结构

控制器的工作循环：
读入伪代码；
读入解释机状态；
根据伪代码内容操作；
写入解释机新状态；
进入下一个循环。

- 虚拟机
  - 代表所有伪代码集合、可出现的不同状态集合、所有处理数据集合、所有操作集合、以及在以上集合上的状态转换关系。
  - E.g. Java虚拟机

2.8 Process Control

- 适用情况
  - 对被控制目标的检测、计算和控制的不断循环执行，以实时地对环境的变化做出反应。

2.8.1 Process-Control Paradigms

- Purpose
  - 将系统的指定输出值保持在set points上
- Open-loop system 开环
- Closed-loop system 闭环

Feedback / FeedForward Control

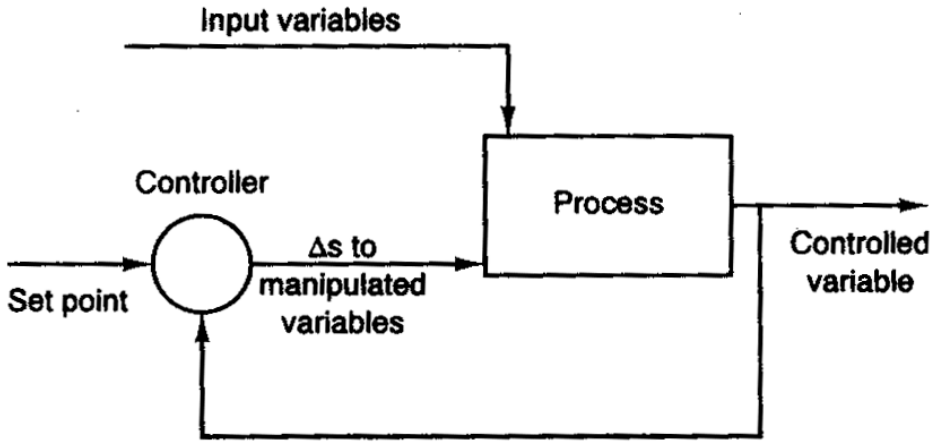


FIGURE 2.10 Feedback Control

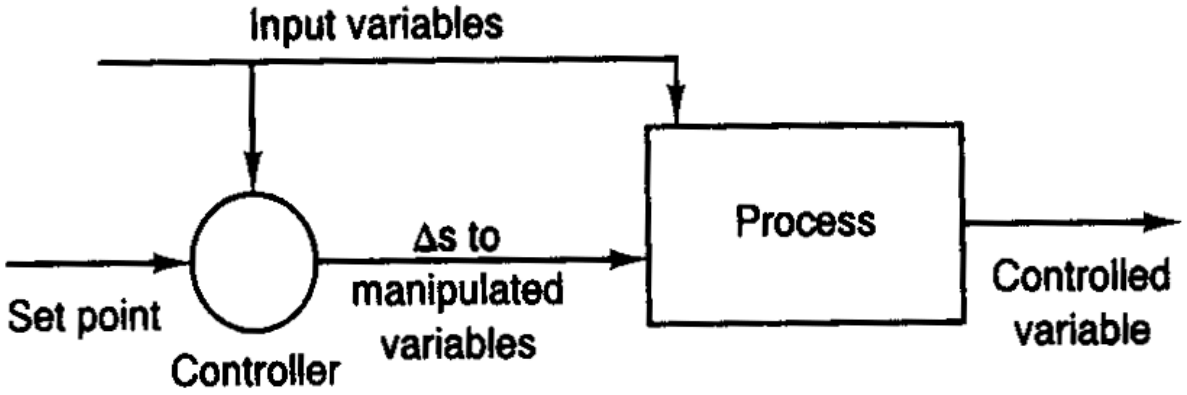


FIGURE 2.11 Feedforward Control

2.8.2 A Software Paradigm for Process Control

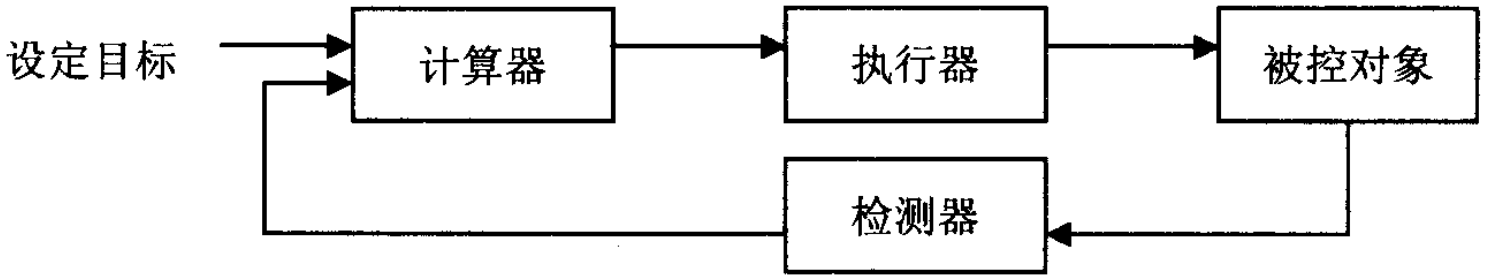
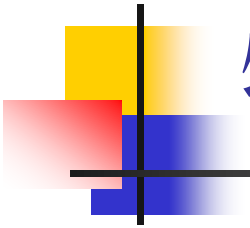


图 11.2　实时与连续计算控制系统

- 特点
  - 整个循环周期执行的实时性
  - 计算的连续性
- 分类：跟踪 、 稳态控制



特点

- Dataflow architecture
  - 部件间的交互通过传递数据来完成
  - 当所有输入数据准备完毕后，该部件即被执行

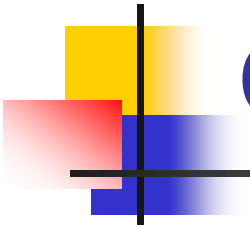
2.9 Other Familiar Architectures

- Distributed processes
  - E.g. C/S

    Server对Client无知;

    Client知道Server的identify

- Main program / subroutine organizations
  - Main作为所有subroutine的driver
  - 提供control loop
  - E.g. DOS环境下的编程（用户输入获取等）



Other Familiar Architectures

- Domain-specific software architectures
  - 与具体领域相关
- State transition systems
  - A set of states
  - A set of named transitions