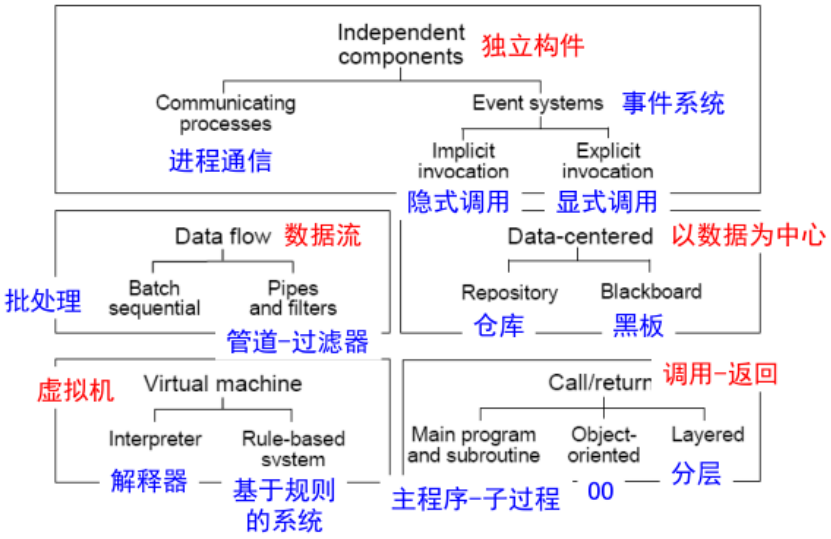
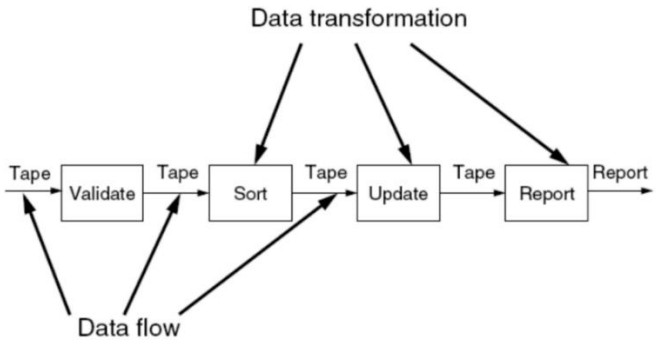


一、数据流风格

1、 总体描述:

【组件】数据流风格组件为计算模型，包括输入接口和输出接口，输入接口读取数据流的数据，通过计算模型处理数据得到结构，将其写入输出接口。【连接件】连接件为数据流，前一个组件的输出接口输出的数据通过数据流传输到下一个组件的输入接口，连接件主要起传输数据的作用。

2、 批处理风格【大量整体数据传输时适用】



【组件】批处理风格的组件是独立计算的程序，【连接件】连接件为某种传输数据的媒介

【优点】

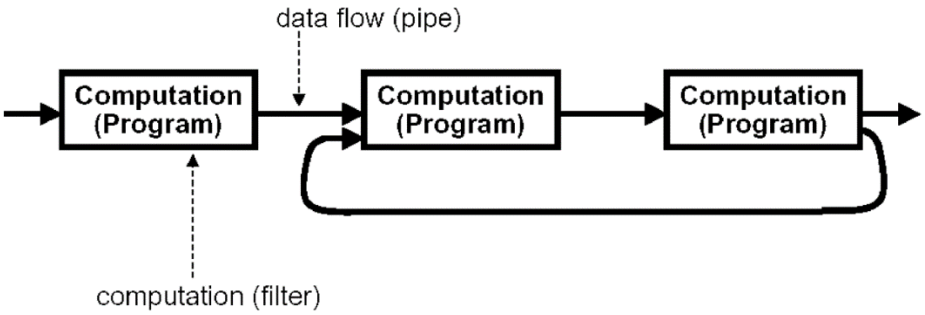
- 无需考虑同步问题：数据是完整的，以整体的方式在组件之间进行传输，因此无需考虑同步数据问题；
- 可以随机存取数据：由于数据是完整传输的，因此在获取数据时，可以完整的存取所有数据。

【缺点】

- 计算效率低，系统性能差，无法并行计算：批处理的处理单元必须在前一个处理单元完全处理完毕后再进行计算，因此系统计算效率低。
- 无法实时计算：每个组件之间的计算顺序是固定的，因此若实时计算，会存在计算延时现象
- 交互性差:每个组件都对完整的数据进行计算,因此在处理数据过程中,用户不易交互。

【应用】编译器，CASE 应用程序；基于 eclipse 的代码重复检测工具

3、 管道过滤器【流式数据传输并且知道流向时适用】



【组件】管道过滤器风格的组件是过滤器, 过滤器将源数据转换为目标数据, 功能包括增加，删除，转换，合并，分解。【连接件】管道过滤器风格的连接件是管道，管道的主要作用在于在过滤器之间传输数据。两个过滤器之间通过管道进行数据的传输。

【优点】

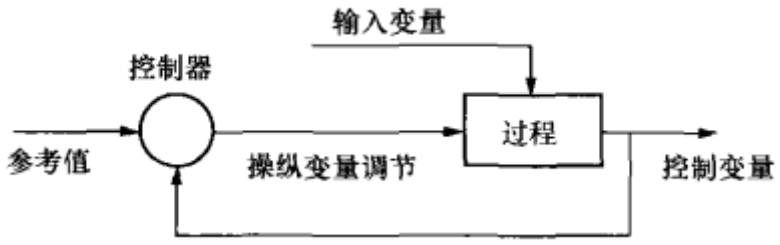
- 可以并行计算：由于管道过滤器输入数据为流式数据，因此对于不同的数据流可以在不同的过滤器上进行并行计算。
- 支持软件的复用：过滤器之间只要数据格式满足输入输出要求，任何两个过滤器均可以进行连接。
- 组件具有良好的隐蔽性，高内聚，低耦合的特点：整个系统的输入输出可以看作内部多个过滤器行为的合成。
- 系统易于维护和扩展：系统中仅需要通过增加过滤器便可以进行系统功能的扩展；同时也仅需要维护某一个过滤器便可以进行功能的维护。

【缺点】

- 性能不高：过滤器之间数据的输入输出格式往往没有统一的标准，因此在过滤器之间数据进行传输的过程中，过滤器要对数据进行解析才能继续使用数据，系统大量事件用户数据格式的转换与解析，因此实际功能的效率较低。
- 不适合用户交互处理：当系统需要增量改变时，管道过滤器往往无法进行很好的工作。

【应用】编译器，图像处理程序，语音处理，数据处理

4、 过程控制（闭环和开环）【存在外部阻力且阻力对于系统目标的更改不确定时适用】



【核心要素】基于过程控制模型，过程控制包括三个要素：过程，数据元素和控制器。【过程】过程主要是操纵过程变量的相关机制；【数据元素】数据元素主要是更新的过程变量，包括输入变量，控制变量，操纵变量，参考值；【控制器】控制器通过控制规则不断修正变量，将实际状态和理论状态进行对比，然后通过规则使得实际状态向着理论状态方向靠近。【组件和交互】将过程定义、过程变量以及传感器看作一个单独的子系统；将控制算法和设定点绑定在一起作为第二个子系统；有两种**交互**：一是**控制器从过程中获取过程变量的值**；

二是**控制器对过程中的被操纵变量的变化提供持续性的引导**。和许多线性的数据流体系结构不同，控制环路体系结构需要**有循环的拓扑结构。**

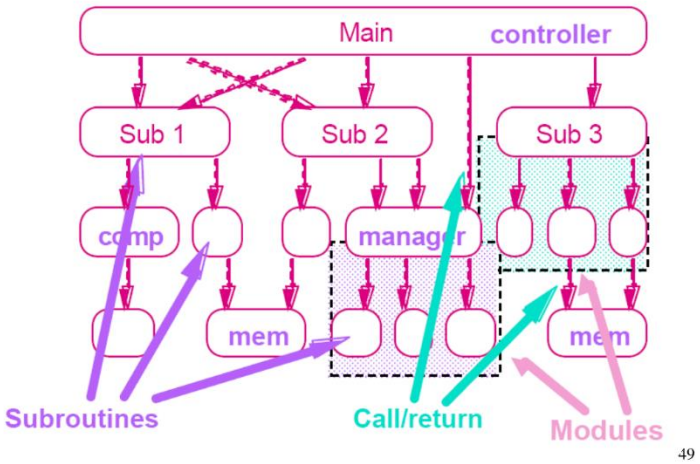
【优点】过程控制是连续的，易于通过各种控制规则和组件设计反馈控制系统，实现 g 俄中功能。可以处理复杂的自适应问题，通不断朝着理想状态前进。

【缺点】PPT 上无

【应用】恒温控制系统

二、返回调用风格

1、 主程序和子程序



49

【核心原理】由单线程控制，主程序和子程序架构的核心是将系统功能分解为模块，模块通过子程序实现，通过主程序调用子程序实现功能。【组件】组件是主程序和子程序【连接件】连接件是主程序和子程序之间的过程调用，每个组件从其主程序获取控制和数据，并将其传递给其子程序，数据实质是隐式共享。【约束】子程序的正确性决定了主程序的正确性。

【优点】

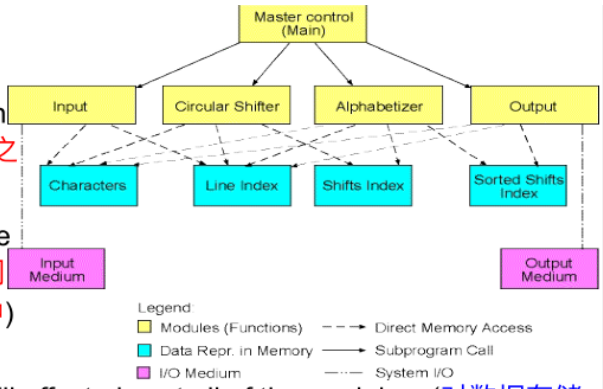
- 流程清晰，易于理解：主程序和子程序之间的调用具有一定的层次结构，层次结构使得系统符合功能分解，分而治之的思维，因此可以通过子程序的调用来清晰的理解系统的执行流程。
- 具有强控制性：主程序和子程序风格具有严格的层次分解和控制权转移，在程序实际执行的过程中具备很强的控制能力。子程序的正确性决定了主程序的正确性。

【缺点】

- 除了过程调用以外的连接件不易描述
- 不易于描述大规模的软件：大规模的软件存在很多的子程序调用，会造成系统程序调用混乱。
- 可复用的层次低：主程序和子程序风格不易于扩展和复用。
- 可测试性差：当代码量过多时，系统不易于测试。

【应用】KWIC 索引系统

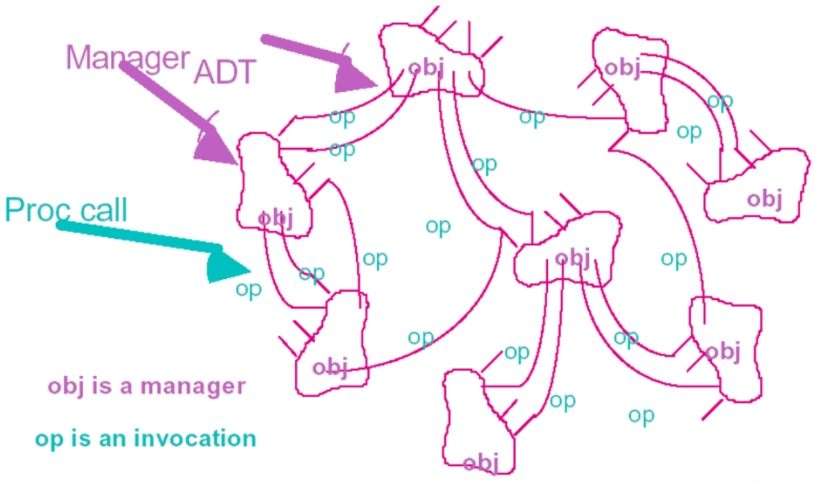
- **Advantages:**
  - Data can be represented efficiently, since computations can share the same storage. (模块之间的数据共享)
  - Distinct computational aspects are isolated in different modules (不同的计算功能被隔离在不同的模块中)



- **Disadvantages:**
  - A change in data storage format will affect almost all of the modules. (对数据存储格式的变化将会影响几乎所有的模块)
  - Changes in the overall processing algorithm and enhancements to system function are not easily accommodated. (对处理流程的改变与系统功能的增强也很难适应，依赖于控制模块内部的调用次序)
  - This decomposition is not particularly supportive of reuse. (这种分解也难以支持有效的复用)

26

2、 面向对象的风格



【组件】面向对象的风格组件是类和对象，类将属性及其相关操作进行封装，对象是类的实例，具有信息隐藏的特点【连接件】对象之间通过方法和函数调用实现交互。
面向对象风格主要具有封装，继承，多态，交互，动态绑定，复用和维护的特点。
【优点】

- 具有易于复用和维护的优点：通过封装和聚合提高了系统的效率和生产力。对其中一个类的修改不会影响到其他的客户。
- 反应现实世界：面向对象的风格架构具有模拟现实世界的特点，因此在现实世界中可能存在实体与系统的对象是对应关系，使得系统更易于理解。
- 易于分解一个系统：将一组例程和其控制的数据绑定在一起，允许设计人员将其分解为交互代理的集合。

【缺点】

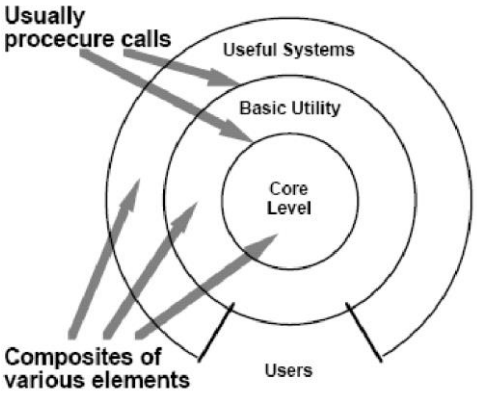
- 若系统复杂，则会产生过多的对象，对象过多，不易于管理，需要额外的结构来存储对

象

- 需要知道对象的身份，才可以进行对象的交互。
- 继承会使系统逻辑变得更加复杂，因此在系统中谨慎使用。

【应用】KWIC 索引系统

3、层次架构风格



【核心原理】系统由若干层次组成，每个层次由若干组件组成，每一层为上一层提供服务，使用下一层的服务。【组件】组件是各层次内部所包含的构件【连接件】连接件是各层之间的交互协议。【拓扑结构】分层【拓扑约束】相邻层次之间的交互的约束

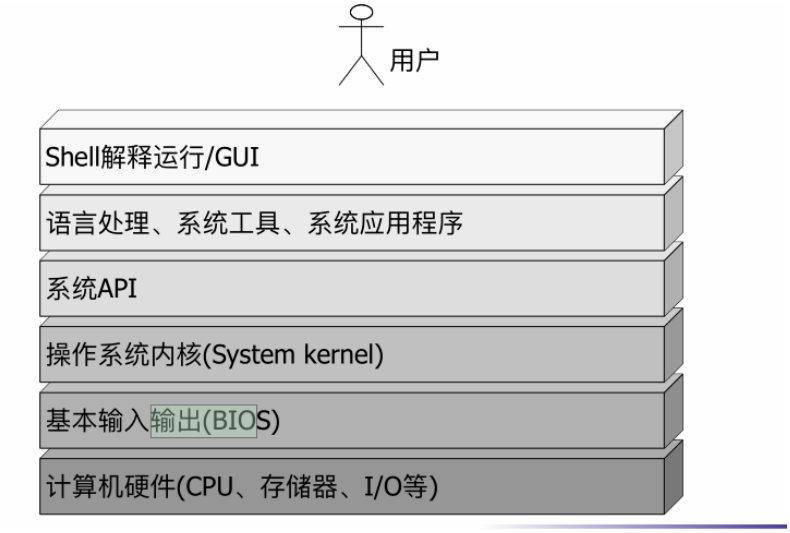
【优点】

- 支持基于抽象程度递增的系统设计：有利于设计者对一个复杂系统进行分解
- 局部依赖性：因为每一层至多和相邻的上下层交互，因此功能的改变通常影响相邻的上下层
- 可复用性：若某层保证了功能完整性并提供文档化接口，便可以在多个语境中复用
- 可替换性：当每一层的服务接口不变时，通过一组标准接口，可以实现相同接口，不同实现的互换使用。便于系统的开发
- 标准化设计：具有抽象程度高且普遍适用的层次接口有利于标准化开发。
- 可测试性：具有明确的层接口以及交换层接口，提高了系统的可测试能力

【缺点】

- 不是每个系统都易于分层：即使一个系统的逻辑结构是层次化的，出于性能考虑，系统设计师需要将一些功能综合起来。
- 系统效率低：分层系统的雄安率一般低于整体系统的效率；高层的服务可能会需要低层功能的支持，因此需要等低层提供相关服务后才能进行高层的运转。
- 难以找到正确层次抽样方法：层数太少，分层不能完全发挥风格的可复用性、可更改性和可移植性上的潜力；层数过多，则引入不必要的复杂性和层间隔离冗余以及层间传输开销

【应用】OSI 网络模型，操作系统的分层架构，物联网的四层体系结构



- 在由上而下方案中
  - 较高层直接调用较低层，因此高层依赖于低层。
- 在由下而上方案中
  - 较低层通过事件(event)、回调(callback)或委派(delegate)来与较高层通信。
- 由上而下的信息和控制通常被描述成请求(request)；
- 由下而上的方式被描述为通知(notify)。

三、以数据为中心

1、 总节点

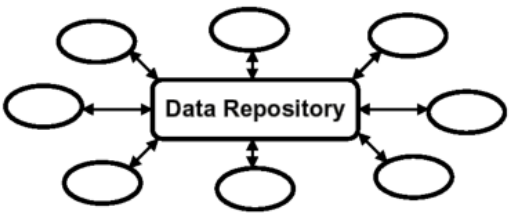
【组件】中心的共享数据以及对数据进行的生产者，消费者，修改者【连接件】数据与操作之间的交互过程。

【优点】系统开放性高，耦合性低；生产者，消费者，修改这易于动态加入；当生产者，消费者，修改者发生故障时，系统不会受到影响。

【缺点】

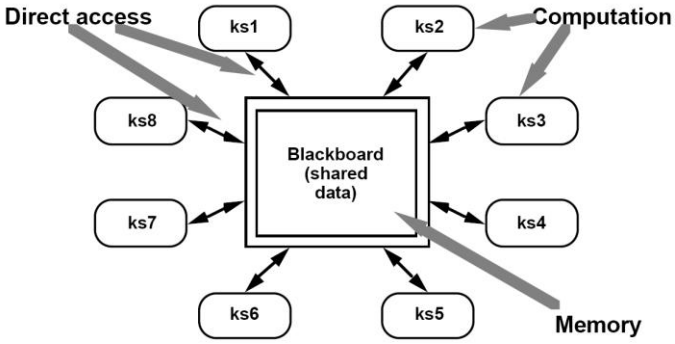
- 同步：当中心数据发生改变时，出现控制反转。最初生产者，消费者，修改者主动控制数据，中心数据被动改变，当中心数据修改后，需要告诉生产者，消费者，修改者其数据被改变。
- 配置和管理：外挂注册中心，生产者，消费者，修改者需要先注册配置才能进入系统
- 性能：各生产者，消费者，修改者之间无法直接通信，性能可能较低
- 具有原子性，持久性，一致性的特点

2、 仓库架构

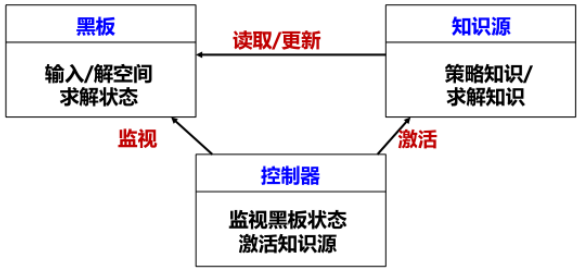


仓库是存储数据和维护数据的场所【组件】组件主要包括两个：中心共享数据和对中心

共享数据的操作；中心共享数据主要用于数据的存储；操作主要用户对数据的处理和改变。
【连接器】仓库与独立构件之间的交互主要分为两种：数据库方式和黑板方式；数据库方式的交互方式是输入流中的**事务类型触发需要执行的过程**；黑板结构交互方式是中心数据结构的当前状态触发并选择需要执行的过程
【优缺点同上】
【应用】数据处理系统，基于传统的数据库的决策系统，现代规范编译器结构
3、 黑板架构



一个大问题被分解为若干个子问题；每个子问题的解决需要不同的问题表达方式和求解模型，分别设计求解程序；
【组件】【连接器】



- 知识源：求解问题的策略；把问题分成几个部分，每个部分采用独立的知识源计算；根据黑板状态的改变调用不同的知识源
- 黑板结构：全局数据库包含解域的全部状态；知识源互相作用的唯一媒介
- 控制器：完全由黑板的状态驱动，黑板的状态的改变决定使用的特定知识；让知识源响应偶然事件。

【优点缺点】

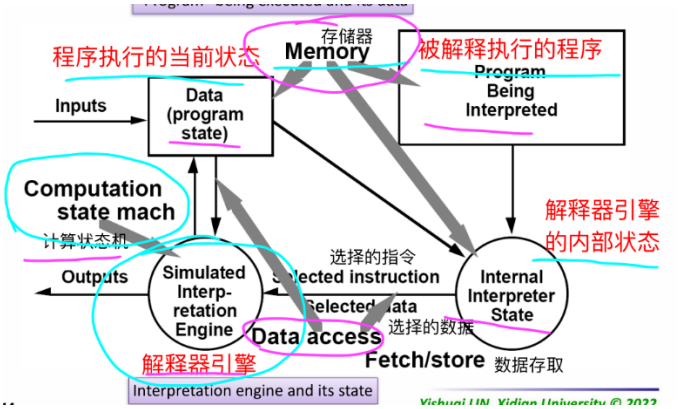
黑板系统	可更改性和可维护性；可重用的知识源；容错性和健壮性	测试困难；不能保证有好的解决方案；难以建立好的控制策略；低效；开发困难；缺少并行机制	在以数据为中心的基础上，使用中心数据触发业务逻辑部件	语音识别 模式识别 图像处理 知识推理

【应用】自然语言处理、语音处理、模式识别、图像处理等

- HEARSAY-II (自然语言处理系统，系统输入是自然语言的语音信号，经过语音音节、词汇、句法和语义分析后，获得用户对数据库的查询请求)
- HASP/SIAP (在特定海域根据声纳阵列信号探测敌方潜艇出没的系统)
- CRYALIS (根据 X 射线探测数据推测蛋白质分子三维结构的系统)
- TRICERO (在分布环境下监视飞机活动的系统)

四、虚拟机架构

1、 解释器架构



【核心原理】解释器通常用来在**程序语言定义的计算**和**有效硬件操作确定的计算**之间建立对应和联系。【组件】组件主要包括解释器引擎和三个存储器，其中三个存储器主要存储执行程序执行的状态，被解释执行的程序，解释器内部的执行状态。【连接器】解释器引擎和存储区之间的数据访问

【优点】

- 功能：可以仿真自身不具有的功能。
- 测试：可以进行测试，模拟“灾难”模式仿真不好的环境释放病毒来测试一些安全软件的功能(例如，安全关键的应用程序)
- 灵活性：具有一定灵活性，是通用的工具。

【缺点】

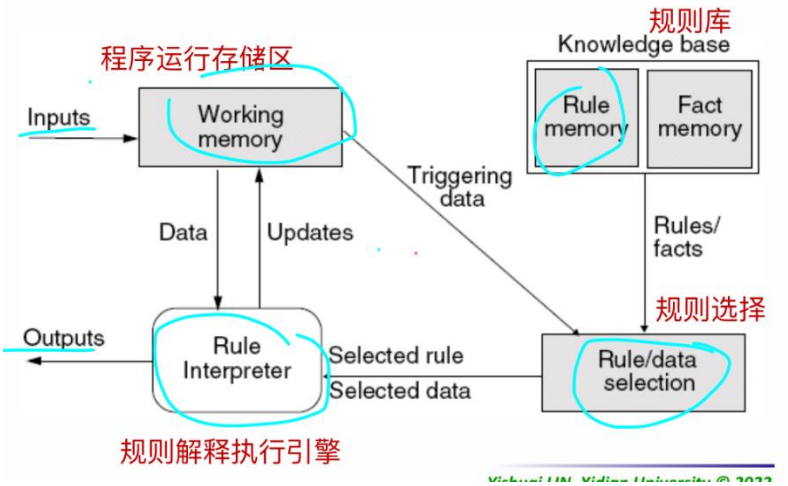
- 效率：增加了额外的一层，需要将运行的程序转入目标环境在进行运行，因此比硬件和编译好的系统慢的多，效率低
- 测试：既需要测试程序本身的正确性，还需要测试额外虚拟环境，测试过程复杂

【应用】JAVA 虚拟机

-包括各类语言环境、Internet 浏览器、数据分析与转换等;

-LISP、Prolog、JavaScript、VBScript、HTML、Matlab、数据库系统(SQL 解释器)、各种通信协议等。

2、 基于规则的系统



【核心原理】基于规则的系统是一个通过模式匹配搜索寻找规则并在正确的时候应用规则的虚拟机。基于规则的系统将业务逻辑中可能产生变化的部分分离出来。【组件】知识库(规则库)、规则解释器、规则与数据元素选择器、工作存储区。输入数据至工作内存，通过数据触发规则的选择，其中规则选择依赖于知识库，知识库：包括事实库和规则库（事实库是对现实世界的一种判定；规则库是在事实的前提下做出的一种行为），然后对规则进行选择，然后将选择好的规则输入解释引擎，解释引擎将结果进行输出。【连接器】过程调用

【优点】

- 降低了修改业务逻辑的成本
- 缩短开发时间
- 将规则外部化，可在多个应用之间共享
- 对规则的改变将非常迅速并且具有较低的风险

【缺点】

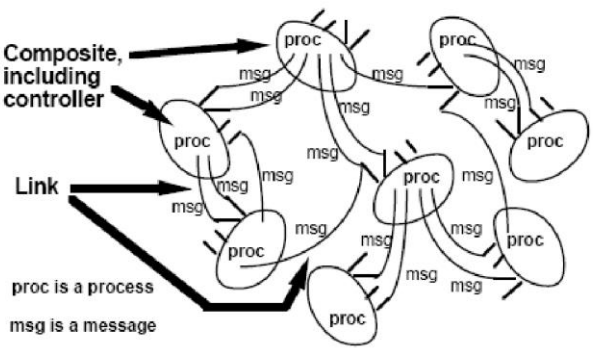
- 效率：增加了额外的一层，需要将运行的程序转入目标环境在进行运行，因此比硬件和编译好的系统慢的多，效率低
- 测试：既需要测试程序本身的正确性，还需要测试额外虚拟环境，测试过程复杂

进程通信：在命名参与者之间传递的消息

事件驱动系统：在未命名参与者之间进行隐式调用

应用：操作系统，分布式应用程序

5.1 进程通信



5.1.1 应用场景

此模式适用于涉及一系列不同的、在很大程度上独立的计算的应用程序，这些计算的执行应该独立进行。计算涉及数据的协调或对离散时间点的控制。因此，系统的正确性需要注意消息的路由和同步。

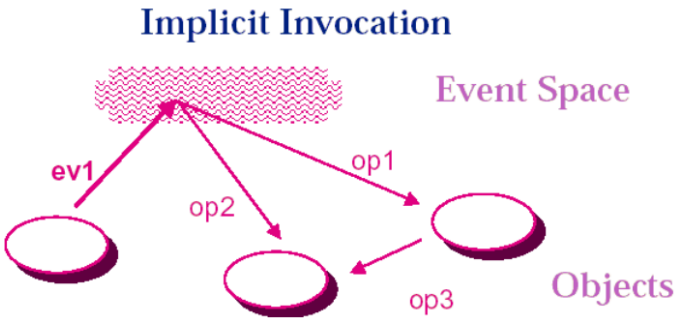
5.1.2 核心原理

- 系统模型：独立的通信进程
- 组件：向明确选择的接收方发送和接收消息的进程
- 连接器：与已知通信伙伴的离散消息(没有共享数据)。点对点或异步和同步。
- 控制结构：每个进程都有自己的控制线程，可以在通信点暂停或继续

5.1.3 优缺点

5.1.4 样例

5.2 事件系统



5.2.1 应用场景

此模式适用于包含松散耦合的组件集合的应用程序，其中每个组件执行某些操作，并可能在流程中启用其他操作。这些通常是反应系统。对于必须动态地进行可重新配置的应用程序(通过更改服务提供者或启用和禁用功能)，该模式特别有用。

5.2.2 核心原理

隐式调用系统通常需要一个事件处理程序，该处理程序注册组件对接收事件的兴趣，并在事件引发时通知组件。

- 系统模型：独立的响应式进程
- 组件：在不知道信号接收方的情况下发出重要事件信号的进程；接口定义了一组传入过程调用和一组传出的事件
- 连接器：连接器是事件-过程绑定。过程注册到事件中，组件通过在“适当”的时间宣布事件进行通信，当一个事件被宣布时，相关的过程被(隐式)调用，调用的顺序是不确定的。
- 控制结构：去中心；单个组件不知道信号的接收方

基于事件的隐式调用风格的思想是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

基于事件的隐式调用风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式

5.2.3 优缺点

优点：

1、问题分解：对象比显式调用更独立，交互策略可以从交互对象中分离出来，为软件重用提供了强大的支持。当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中

2、系统维护与复用：没有硬连接(静态)名称依赖，所以动态重新配置很容易-简化了系统演化，只需注册新对象就可以使用它们，简化了集成；为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其它构件的接口

3、性能：调用可以并行化

4、健壮性：一个组件的崩溃不会影响其他组件

缺点：

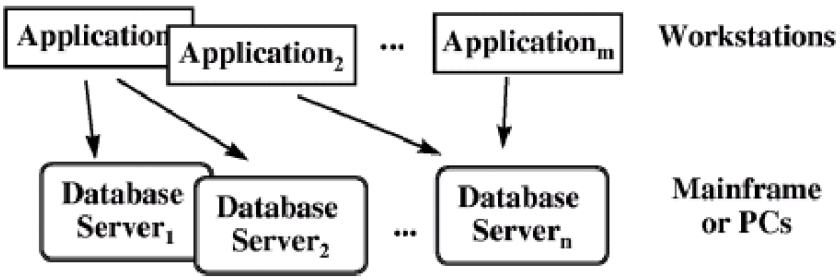
1、语义正确性问题：一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的构成，它也不能保证这些过程被调用的顺序；可能会出现函数语义问题和环路问题，正确性难以保证

2、数据交换问题：有时数据可能同时背多个事件进行传递，因此，基于事件的系统需要一

个共享仓库进行数据的交互，此时系统的全局性能和资源管理存在问题。
3、性能：间接/通信意味着一些性能损失

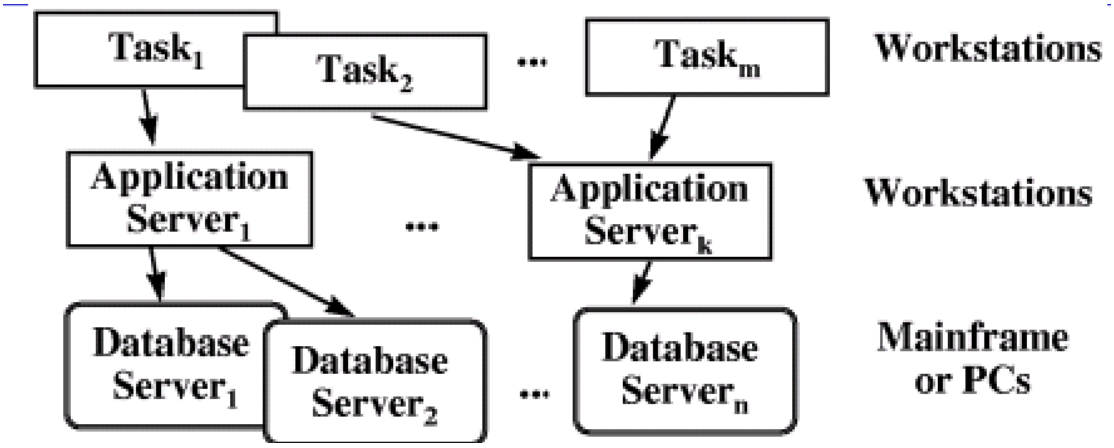
5.2.4 样例
在编程环境中用于集成各种工具，
在数据库管理系统中确保数据的一致性约束，
在用户界面系统中管理数据，
在编辑器中支持语法检查。例如在某系统中，编辑器和变量监视器可以登记相应 Debugger 的断点事件。当 Debugger 在断点处停下时，它声明该事件，由系统自动调用处理程序，如编辑程序可以卷屏到断点，变量监视器刷新变量数值。而 Debugger 本身只声明事件，并不关心哪些过程会启动，也不关心这些过程做什么处理
Smalltalk-80 Model-View-Controller (MVC)：模型，视图，控制组成，将试图注册到模型的事件中，当模型进行改变的时候，会触发事件然后相应视图显示的数据会进行改变。

1、 客户端服务器风格
（1）、两层 C/S 结构:



- C/S 体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络
- 服务器（后台）负责数据管理，客户机（前台）完成与用户的交互任务。（胖客户机，瘦服务器）【业务逻辑放在客户端】
- 缺点：
  - 对客户端软硬件配置要求较高，客户端臃肿，程序设计复杂
  - 数据安全性不好。客户端程序可以直接访问数据库服务器。
  - 信息内容和形式单一
  - 用户界面风格不一，使用繁杂，不利用推广使用
  - 软件维护与升级困难。每个客户机上的软件都需要维护

（2）、三层 C/S 结构：与二层 C/S 结构相比，增加了一个应用服务器。



- 整个应用逻辑驻留在应用服务器上，只有表示层存在于客户机上。“瘦客户机”
- 应用功能分为表示层、功能层、数据层三层
  - 表示层是应用的用户接口部分。通常使用图形用户界面
  - 功能层是应用的主体，实现具体的业务处理逻辑
  - 数据层是数据库管理系统。
  - 以上三层逻辑上独立。
  - 通常只有表示层配置在客户机中

（3）、B/S 结构（浏览器\服务器风格）：B/S 体系结构是三层 C/S 体系结构的特例

- 客户端有 http 浏览器即可：为增强功能，往往还需要安装 flash、jvm 及一些专用插件
- 使用标准 http/https 协议，省却很多麻烦
- 只能“拉”，不能“推”（只能客户拉取服务端升级后的数据，不能服务器推数据）
- 客户之间的通信只能通过服务器中转
- 对客户机资源和其他网络资源的利用受限
- B/S 结构的安全性较难控制(SQL 注入攻击…)
- B/S 结构的应用系统在数据查询等相应速度上，要远远低于 C/S 体系结构
- 服务器的负荷大，客户机的资源浪费：用 jvm、flash、ActiveX 等客户端计算技术解决