基于层次消息总线的软件体系结构风格*

张世琨 王立福 杨芙清
(北京大学计算机科学技术系, 北京 100871)

摘要 随着软件系统规模和复杂性的增加, 系统总体结构设计的重要性已远远超过特定算法和数据结构的选择, 良好的体系结构是保证系统成功的关键. 以青鸟软件生产线的实践为背景, 提出了一种基于层次消息总线的软件体系结构风格 JB/HMB. 在该风格中, 构件模型由外部接口、静态结构和动态行为等 3 部分组成, 从多个视角对构件进行刻画. 在消息总线的支持下, 构件通过消息相互通讯, 可以较好地刻画具有分布和并发特点的系统. 支持系统的逐层分解、细化, 以及运行时刻的系统演化. 对 JB/HMB 风格的特点进行了总结, 并指出进一步的研究方向.

关键词 软件体系结构 软件构件 体系结构风格 层次消息总线

在软件工程实践中, 随着软件系统规模和复杂性的增长, 系统总体结构设计的重要性已远远超过了特定算法和数据结构的选择, 良好的软件体系结构是保证系统成功的关键. 软件体系结构目前已成为软件工程研究的热点之一.

软件体系结构设计位于系统开发的前期, 在这一层次上的设计主要包括以下内容: 系统中构件的描述、构件之间的交互、指导构件交互的模式以及施加在模式上的约束等[1]. 具体地说, 一个软件系统的体系结构刻画了系统的以下方面:

(ⅰ) 系统总体结构: 体系结构采用高层抽象的计算成分和它们之间的相互作用刻画了系统的结构(structure). 也就是说, 体系结构通过配置交互的构件, 为问题提供解决方案. 它既不是需求的表示(例如, 问题域元素的抽象关系), 也不是实现细节(例如, 算法和数据结构).

(ⅱ) 构件交互方式的抽象: 在体系结构设计中, 构件之间的相互作用反映了系统设计人员对构件交互方式所做的抽象. 构件之间的交互可以是非常简单的, 如过程调用和共享数据访问, 也可以是复杂并具有丰富语义的, 例如, 客户-服务器协议、数据库访问协议、异步事件广播(可以同时有多个接受者)和管道-过滤器等.

(ⅲ) 全局特性: 体系结构设计的一个主要作用是刻画系统的整体行为, 因此体系结构通常关注的是系统一级的行为特征, 例如端对端的数据传输率和响应时间, 系统的一部分对其他部分失效的适应程度, 或者当系统的某一部分修改以后, 所造成的波动影响范围等.

目前对软件体系结构的研究集中在以下方面: 各种体系结构风格的汇编和总结、体系结构描述语言(architectural description languages, 简称 ADLs)、体系结构的形式化基础、体系结构分析技术、基于体系结构的软件开发、体系结构恢复和再工程、支持体系结构设计的工具和环境及特定领域的软件体系结构等[1~4].

2000-09-15 收稿, 2001-06-18 收修改稿
* 国家“九五”重点科技攻关项目基金资助项目(批准号: 98-780-01)

青鸟工程在“九五”期间, 对基于构件-构架 [1)]模式的软件工业化生产技术进行了研究, 并实现了青鸟软件生产线系统[5]. 以青鸟软件生产线的实践为背景, 提出了基于层次消息总线的软件体系结构风格(Jade bird hierarchical message bus-based style, 以下简称 JB/HMB 风格), 设计了相应的体系结构描述语言[6], 开发了支持软件体系结构设计的辅助工具集, 并研究了采用 JB/HMB 风格进行应用系统开发的过程框架. 本文将集中讨论 JB/HMB 风格及其特点.

1 JB/HMB 风格的基本特征

软件体系结构风格有时也称为软件体系结构模式. 一种体系结构风格定义了关于构件和连接件类型的术语, 以及一组约束它们组合方式的规定. 许多风格还存在一个或多个语义模型, 指明如何根据各成分的属性确定系统的总体属性[1]. 概括地说, 一种体系结构风格刻画了一个具有共享结构和语义的系统家族. 目前存在许多不同类型的体系结构风格, 而且新的风格还在不断地出现, 比较典型的有: 数据流风格、面向对象风格、管道-过滤器风格、客户-服务器风格、基于事件的风格、层次系统风格、过程控制风格和基于消息广播且面向图形用户界面的 C2 风格等[1, 4].

JB/HMB 风格的提出基于以下的实际背景:

(ⅰ) 随着计算机网络技术的发展, 特别是分布式构件技术的日渐成熟和构件互操作标准的出现, 如 CORBA, DCOM 和 EJB 等, 加速了基于分布式构件的软件开发趋势, 具有分布和并发特点的软件系统已成为一种普遍的应用需求.

(ⅱ) 基于事件驱动的编程模式已在图形用户界面程序设计中获得广泛应用. 在此之前的程序设计中, 通常使用一个大的分支语句(switch statement)控制程序的转移, 对不同的输入情况分别进行处理, 程序结构不甚清晰. 基于事件驱动的编程模式在对多个不同事件响应的情况下, 系统自动调用相应的处理函数, 程序具有清晰的结构.

(ⅲ) 计算机硬件体系结构和总线的概念为软件体系结构的研究提供了很好的借鉴和启发, 在统一的体系结构框架下(即总线和接口规范), 系统具有良好的扩展性和适应性. 任何计算机厂商生产的配件, 甚至是在设计体系结构时根本没有预料到的配件, 只要遵循标准的接口规范, 都可以方便地集成到系统中, 对系统功能进行扩充, 甚至是即插即用(即运行时刻的系统演化). 正是标准的总线和接口规范的制定, 以及标准化配件的生产, 促进了计算机硬件的产业分工和蓬勃发展.

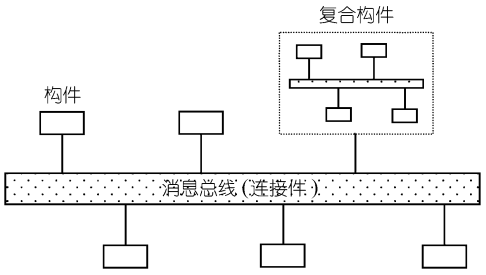

图 1 JB/HMB 风格的系统示意图

JB/HMB 风格基于层次消息总线、支持构件的分布和并发, 构件之间通过消息总线进行通讯, 如图 1 所示. 消息总线是系统的连接件, 负责消息的分派、传递和过滤以及处理结果的返回. 各个构件挂接在消息总线上, 向总线登记感兴趣的消息类型. 构件根据需要发出消息, 由消息总线负责把该消息分派到系统中所有对此消息感兴趣的构件, 消息是构件之间通讯的唯一方式. 构件接收到消息后, 根据自身状态对消息进行响应, 并通过总线

1) 在本文中, “构架”和“体系结构”两个词是同义的, 可以相互替换, 都对应英文单词 architecture

返回处理结果. 由于构件通过总线进行连接, 并不要求各个构件具有相同的地址空间或局限在一台机器上. 该风格可以较好地刻画分布式并发系统, 以及基于 CORBA, DCOM 和 EJB 规范的系统.

如图 1 所示, 系统中的复杂构件可以分解为比较低层的子构件, 这些子构件通过局部消息总线进行连接, 这种复杂的构件称为复合构件. 如果子构件仍然比较复杂, 可以进一步分解. 如此分解下去, 整个系统形成了树状的拓扑结构, 树结构的末端结点称为叶结点, 它们是系统中的原子构件, 不再包含子构件, 原子构件的内部可以采用不同于 JB/HMB 的风格, 例如前面提到的数据流风格、面向对象风格及管道-过滤器风格等, 这些属于构件的内部实现细节. 但要集成到 JB/HMB 风格的系统中, 必须满足 JB/HMB 风格的构件模型的要求, 主要是在接口规约方面的要求. 另外, 整个系统也可以作为一个构件, 通过更高层的消息总线, 集成到更大的系统中. 于是, 可以采用统一的方式刻画整个系统和组成系统的单个构件.

以下各部分详细讨论 JB/HMB 风格中的各个组成要素.

2　构件模型

系统和组成系统的成分通常是比较复杂的, 难以从一个视角获得对它们的完整理解, 因此一个好的软件工程方法往往从多个视角对系统进行建模, 一般包括系统的静态结构、动态行为和功能等方面[7]. 例如, 在 Rumbaugh 等人提出的 OMT(object modeling technology)方法中, 采用了对象模型、动态模型和功能模型刻画系统的以上 3 个方面[8].

借鉴上述思想, 为满足体系结构设计的需要, JB/HMB 风格的构件模型包括了接口、静态结构和动态行为 3 个部分, 如图 2 所示.

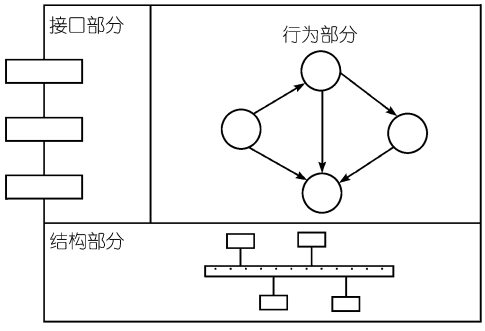

图 2　JB/HMB 风格的构件模型

在图 2 所示的构件模型中, 左上方是构件的接口部分, 一个构件可以支持多个不同的接口, 每个接口定义了一组输入和输出的消息, 刻画了构件对外提供的服务以及要求的环境服务, 体现了该构件同环境的交互. 右上方是用带输出的有限状态自动机刻画的构件行为, 构件接收到外来消息后, 根据当前所处的状态对消息进行响应, 并可能导致状态的变迁. 下方是复合构件的内部结构定义, 复合构件是由更简单的子构件通过局部消息总线连接而成的. 消息总线为整个系统和各个层次的构件提供了统一的集成机制.

3　构件接口

在体系结构设计层次上, 构件通过接口定义了同外界的信息传递和承担的系统责任, 构件接口代表了构件同环境的全部交互内容, 也是唯一的交互途径. 除此之外, 环境不应对构件做任何其他与接口无关的假设, 例如实现细节等.

JB/HMB 风格的构件接口是一种基于消息的互联接口, 可以较好地支持体系结构设计[9]. 构件之间通过消息进行通讯, 接口定义了构件发出和接收的消息集合. 同一般的互联接口相比, JB/HMB 的构件接口具有两个显著的特点. 首先, 构件只对消息本身感兴趣, 并不关心消

息是如何产生的，消息的发出者和接收者不必知道彼此的情况，这样就切断了构件之间的直接联系，降低了构件之间的耦合强度，进一步增强了构件的复用潜力，并使得构件的替换变得更为容易. 另外，在一般的互联接口定义的系统中，构件之间的连接是在要求的服务和提供的服务之间进行固定的匹配，而在 JB/HMB 的构件接口定义的系统中，构件对外来消息的响应，不但同接收到的消息类型相关，而且同构件当前所处的状态相关. 构件对外来消息进行响应后，可能会引起状态的变迁. 因此，一个构件在接收到同样的消息后，在不同时刻所处的不同状态下，可能会有不同的响应.

消息是关于某个事件发生的信息，上述接口定义中的消息分为两类: (ⅰ) 构件发出的消息，通知系统中其他构件某个事件的发生或请求其他构件的服务; (ⅱ) 构件接收的消息，对系统中某个事件的响应或提供其他构件所需的服务. 接口中的每个消息定义了构件的一个端口，具有互补端口的构件可以通过消息总线进行通讯，互补端口指的是除了消息进出构件的方向不同之外，消息名称、消息带有的参数和返回结果的类型完全相同的两个消息.

当某个事件发生后，系统或构件发出相应的消息，消息总线负责把该消息传递到对此消息感兴趣的构件. 按照响应方式的不同，消息可分为同步消息和异步消息. 同步消息是指消息的发送者必须等待消息处理结果返回才可以继续运行的消息类型. 异步消息是指消息的发送者不必等待消息处理结果的返回即可继续执行的消息类型. 常见的同步消息包括(一般的)过程调用，异步消息包括信号、时钟和异步过程调用等.

4 消息总线

JB/HMB 风格的消息总线是系统的连接件，构件向消息总线登记感兴趣的消息，形成构件-消息响应登记表. 消息总线根据接收到的消息类型和构件-消息响应登记表的信息，定位并传递该消息给相应的响应者，并负责返回处理结果. 必要时，消息总线还对特定的消息进行过滤和阻塞. 图 3 给出了采用对象类符号表示的消息总线的结构.

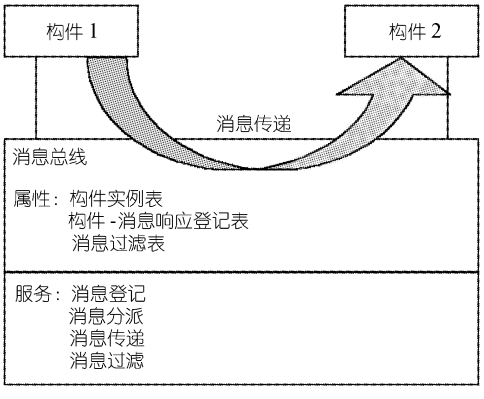

图 3 消息总线的结构

4.1 消息登记

在基于消息的系统中，构件需要向消息总线登记当前响应的消息集合，消息响应者只对消息类型感兴趣，通常并不关心是谁发出的消息. 在 JB/HMB 风格的系统中，对挂接在同一消息总线上的构件而言，消息是一种共享的资源，构件-消息响应登记表记录了该总线上所有构件和消息的响应关系. 类似于程序设计中的“间接地址调用”，避免了将构件之间的连接“硬编码”到构件的实现中，使得构件之间保持了灵活的连接关系，便于系统的演化.

构件接口中的接收消息集合意味着构件具有响应这些消息类型的潜力，缺省情况下，构件对其接口中定义的所有接收消息都可以进行响应. 但在某些特殊的情况下，例如，当一个构件在部分功能上存在缺陷时，就难以对其接口中定义的某些消息进行正确的响应，这时应阻塞掉那些不希望接收到的消息. 这就是需要显式进行消息登记的原因，以便消息响应者更灵

活地发挥自身的潜力.

4.2　消息分派和传递

消息总线负责消息在构件之间的传递, 根据构件-消息响应登记表把消息分派到对此消息感兴趣的构件, 并负责处理结果的返回. 在消息广播的情况下, 可以有多个构件同时响应一个消息, 也可以没有构件对该消息进行响应. 在后一种情况下, 该消息就丢失了, 消息总线可以对系统的这种异常情况发出警告, 或通知消息的发送构件进行相应地处理.

实际上, “构件-消息响应登记表”定义了消息的发送构件和接收构件之间的一个二元关系, 以此作为消息分派的依据.

消息总线是一个逻辑上的整体, 在物理上可以跨越多个机器, 因此挂接在总线上的构件也就可以分布在不同的机器上, 并发运行. 由于系统中的构件不是直接交互, 而是通过消息总线进行通讯, 因此实现了构件位置的透明性. 根据当前各个机器的负载情况和效率方面的考虑, 构件可以在不同的物理位置上透明地迁移, 而不影响系统中的其他构件.

4.3　消息过滤

消息总线对消息过滤提供了转换和阻塞两种方式. 消息过滤的原因主要在于不同来源的构件事先并不知道各自的接口, 因此可能同一消息在不同构件中使用了不同的名字, 或不同的消息使用了相同的名字. 前面我们提到, 对挂接在同一消息总线上的构件而言, 消息是一种共享的资源, 这样就会造成构件集成时消息的冲突和不匹配.

消息转换是针对构件实例[1)]而言的, 即所有构件实例发出和接收的消息类型都经过消息总线的过滤, 这里采取简单换名的方法, 其目标是保证每种类型的消息名字在其所处的局部总线范围内是唯一的. 例如, 假设复合构件 A 符合客户/服务器风格, 由构件 C 的两个实例 c1 和 c2 以及构件 S 的一个实例 s1 构成, 构件 C 发出的消息 msgC 和构件 S 接收的消息 msgS 是相同的消息. 但由于某种原因, 它们的命名并不一致(除此之外, 消息的参数和返回值完全一样). 我们可以采取简单换名的方法, 把构件 C 发出的消息 msgC 换名为 msgS, 这样无需对构件进行修改, 就解决了这两类构件的集成问题.

由简单的换名机制解决不了的构件集成的不匹配问题, 例如参数类型和个数不一致等, 可以采取更为复杂的包装器(wrapper)技术对构件进行封装[2].

5　构件静态结构

JB/HMB 风格支持系统自顶向下的层次化分解, 复合构件是由比较简单的子构件组装而成的, 子构件通过复合构件内部的消息总线连接, 各个层次的消息总线在逻辑功能上是一致的, 负责相应构件或系统范围内消息的登记、分派、传递和过滤. 如果子构件仍然比较复杂, 可以进一步分解. 图 4 是某个系统经过逐层分解所呈现出的结构示意图, 不同的消息总线分别属于系统和各层

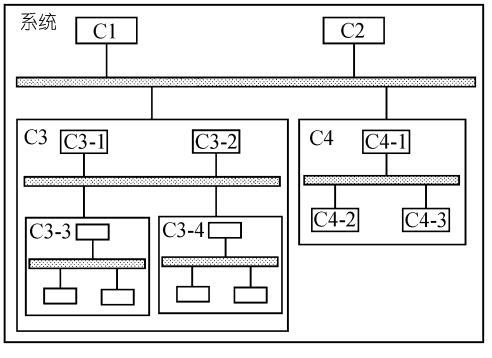

图 4　系统/复合构件的静态结构示意图

1) 在本文中, 构件和构件实例的关系类似于面向对象中的类和对象的关系. 在不引起混淆的情况下, 有时为了方便起见, 也把“构件实例”简称为“构件”

次的复合构件, 消息总线之间没有直接的连接, 我们把 JB/HMB 风格中的这种总线称为层次消息总线. 另外, 整个系统也可以作为一个构件, 集成到更大的系统中. 因为各个层次的构件以及整个系统采取了统一的方式进行刻画, 所以定义一个系统的同时也就定义了一组“系统”, 每个构件都可看作一个独立的子系统.

6 构件动态行为

在一般的基于事件风格的系统中, 如图形用户界面系统 X-Window, 对于同一类事件, 构件(这里指的是回调函数)总是采取同样的动作进行响应. 这样, 构件的行为就由外来消息的类型唯一确定, 即一个消息和构件的某个操作之间存在着固定的对应关系. 对于这类构件, 可以认为构件只有一个状态, 或者在每次对消息响应之前, 构件处于初始状态. 虽然在操作的执行过程中, 会发生状态的变迁, 但在操作结束之前, 构件又恢复到初始状态. 无论以上哪种情况, 都不需要构件在对两个消息响应之间, 保持其状态信息.

更通常的情况是, 构件的行为同时受外来消息类型和自身当前所处状态的影响. 类似一些面向对象方法中用状态机刻画对象的行为, 在 JB/HMB 风格的系统中, 我们采用带输出的有限状态机描述构件的行为.

带输出的有限状态机分为 Moore 机和 Mealy 机两种类型, 它们具有相同的表达能力[10]. 在一般的面向对象方法中, 通常混合采用 Moore 机和 Mealy 机表达对象的行为. 为了实现简单起见, 我们选择采用 Mealy 机来描述构件的行为. 一个 Mealy 机包括一组有穷的状态集合、状态之间的变迁和在变迁发生时的动作. 其中, 状态表达了在构件的生命周期内, 构件所满足的特定条件、实施的活动或等待某个事件的发生[11].

7 运行时刻的系统演化

在许多重要的应用领域中, 例如金融、电力、电信及空中交通管制等, 系统的持续可用性是一个关键性的要求, 运行时刻的系统演化可减少因关机和重新启动而带来的损失和风险[12]. 此外, 越来越多的其他类型的应用软件也提出了运行时刻演化的要求, 在不必对应用软件进行重新编译和加载的前提下, 为最终用户提供系统定制和扩展的能力.

JB/HMB 风格方便地支持运行时刻的系统演化, 主要体现在以下 3 个方面:

(ⅰ) 动态增加或删除构件. 在 JB/HMB 风格的系统中, 构件接口中定义的输入和输出消息刻画了一个构件承担的系统责任和对外部环境的要求, 构件之间通过消息总线进行通讯, 彼此并不知道对方的存在. 因此只要保持接口不变, 构件就可以方便地替换. 一个构件加入到系统中的方法很简单, 只需向系统登记其所感兴趣的消息即可. 但删除一个构件可能会引起系统中对于某些消息没有构件响应的异常情况, 这时可以采取两种措施: 一是阻塞那些没有构件响应的消息, 二是首先使系统中的其他构件或增加新的构件对该消息进行响应, 然后再删除相应的构件. 系统中可能增删改构件的情况包括: 当系统功能需要扩充时, 往系统中增加新的构件. 当对系统功能进行裁剪, 或当系统中的某个构件出现问题时, 需要删除系统中的某个构件. 用带有增强功能或修正了错误的构件新版本代替原有的旧版本.

(ⅱ) 动态改变构件响应的消息类型. 类似地, 构件可以动态地改变对外提供的服务(即接收的消息类型), 这时应通过消息总线对发生的改变进行重新登记.

(ⅲ) 消息过滤. 利用消息过滤机制, 可以解决某些构件集成的不匹配问题, 详见“消息过滤”一节. 消息过滤通过阻塞构件对某些消息的响应, 提供了另一种动态改变构件对消息进行响应的方式.

8　结论

以上讨论了 JB/HMB 风格的各组成要素, 下面对 JB/HMB 风格的主要特点作一总结.

(Ⅰ) 从接口、结构和行为方面对构件进行刻画. 在 JB/HMB 风格中, 构件的描述包括接口、静态结构和动态行为 3 个方面.

(ⅰ) 接口: 构件可以提供一个或多个接口, 每个接口定义了一组发送和接收的消息集合, 刻画了构件对外提供的服务以及要求的环境服务, 接口之间可以通过继承表达相似性.

(ⅱ) 静态结构: 复合构件是由子构件通过局部消息总线连接而成的, 形成该复合构件的内部结构.

(ⅲ) 动态行为: 构件行为通过带输出的有限状态机刻画, 构件接收到外来消息后, 不但根据消息类型, 而且根据构件当前所处的状态对消息进行响应, 并导致状态的变迁.

(Ⅱ) 基于层次消息总线. 消息总线是系统的连接件, 负责消息的传递、过滤和分派, 以及处理结果的返回. 各个构件挂接在总线上, 向系统登记感兴趣的消息. 构件根据需要发出消息, 由消息总线负责把该消息分派到系统中对此消息感兴趣的所有构件. 构件接收到消息后, 根据自身状态对消息进行响应, 并通过总线返回处理结果. 由于构件通过总线进行连接, 并不要求各个构件具有相同的地址空间或局限在一台机器上, 系统具有并发和分布的特点. 系统和复合构件可以逐层分解, 子构件通过(局部)消息总线相连. 每条消息总线分别属于系统和各层次的复合构件, 我们把这种特征的总线称为层次消息总线. 在系统开发方面, 由于各层次的总线局部在相应的复合构件中, 因此可以更好地支持系统的构造性和演化性.

(Ⅲ) 统一描述系统和组成系统的构件. 组成系统的构件通过消息总线进行连接, 复杂构件又可以分解为比较简单的子构件, 通过局部消息总线进行连接, 如果子构件仍然比较复杂, 可以进一步分解. 系统呈现出树状的拓扑结构. 另外, 整个系统也可以作为一个构件, 集成到更大的系统中. 于是, 就可以对整个系统和组成系统的各层构件采用统一的方式进行描述.

(Ⅳ) 支持运行时刻的系统演化. 系统的持续可用性是许多重要的应用系统的一个关键性要求, 运行时刻的系统演化可减少因关机和重新启动而带来的损失和风险. JB/HMB 风格方便地支持运行时刻的系统演化, 主要包括动态增加或删除构件、动态改变构件响应的消息类型和消息过滤.

软件体系结构的研究是青鸟软件生产线实践的重要组成部分, 目前已在针对商业领域的青鸟软件生产线示范工程中获得应用. 进一步的研究工作主要包括:

(ⅰ) 研究支持 JB/HMB 风格的运行机制, 并开发相应的支持平台.

(ⅱ) 针对不同的应用领域, 对 JB/HMB 风格及描述语言进行裁剪和扩充例如针对实时应用系统, 加入对时间的描述, 进一步增强 JB/HMB 风格在特定领域内建模系统的能力.

参　考　文　献

1　Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River: Prentice Hall, 1996

2 Bass L, Clements P, Kazman R. Software Architecture in Practice. Reading: Addison-Wesley Publishing Company, 1997
3 Buschmann F, Meunier R, Rohnert H, et al. Pattern-Oriented Software Architecture: A System of Patterns. New York: John Wiley & Sons Ltd, 1996
4 Taylor R, Medvidovic N, Anderson K, et al. A component- and message-based architectural style for GUI software. IEEE Transactions on Software Engineering, 1996, 22(6): 390~406
5 杨芙清. 杨芙清文集. 北京: 北京大学出版社, 1998
6 张世琨, 王立福, 常 欣, 等. 基于层次消息总线的软件体系结构描述语言. 电子学报, 2001, 29(5): 581~584
7 Pressman R. Software Engineering, A Practitioner's Approach. Fourth Edition. New York: The McGraw-Hill Companies Inc, 1997
8 Rumbaugh J, Blaha M, Premerlani W, et al. Object-Oriented Modeling and Design. Supper Saddle River: Prentice Hall, 1991
9 Luckham D, Vera J, Meldal S. Three Concepts of System Architecture. CSL-TR-95-674, Stanford: Stanford University, 1995
10 Hopcroft J, Ullman J. Introduction to Automata Theory, Languages, and Computation. Reading: Addison-Wesley Publishing Company, 1979
11 Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Reading: Addison-Wesley Publishing Company, 1998