计算机科学2002Vol. 29№. 6

软件连接件的分类及其应用研究*)

On the Classification of Software Connectors and their Applications

严昌浩　刘云生　张文彬

(华中科技大学计算机科学与技术学院　武汉430074)

Abstract That a connector's structure should consist of an implementation structure and a protocol structure is suggested in this article. For the practice purposes, all connections are divided into six styles in the view of implementation structure. For control connection styles, they consist of synchronous and asynchronous connections by time, and soft-connection and hard-connection by space. And for data connection styles, they only consist of soft and hard connections. Specially, the difference between soft connection and hard connection, though it could not be obviously distinguished by some formal ADL(Architecture Description Language), could influences the software architecture intensely. From it, a new method of metricating the complexion of software architecture is proposed.

Keywords Software architecture, Connector, Synchronous connection, Asynchronous connection, Soft-connection, Hard-connection

1　引言

连接件在软件体系结构中占有重要的地位。绝大部分的软件体系结构分析方法，均把连接件作为一个单独的实体进行分析。把原本分散在系统各个部分的连接成分，组成一个单独可见的实体，并把它提到与组件相同的重要程度，这是软件体系结构分析的重要成果之一[1,2]。当前，国内外对体系结构研究的主要方面有：

1)体系结构形式化描述。如：Wright[3]，ACME[4]等，给出了连接件的标记式形式化描述方法，但其直观性差，不便实际使用。本文将指出，Wright 标记法[8]的局限性。

2)体系结构风格的归纳整理[5,6]。通过归纳，提炼出若干种模式(pattern)和风格(style)，形成类似工程手册一样的经验，供后来的设计者使用。这是一种实用化的研究路线，但没有对连接件本身进行分类。

3)特定领域软件体系结构研究[7]，其通用性差。

4)其它。如：文[8]提出，采用主动连接件这一种连接形式，有助于连接件的实体化。但为了提供足够的灵活性，将连接件与主动机制联系，其结果必然导致本已十分复杂的连接件更加复杂。

到目前为止，少有论文对软件体系结构的重要部件——连接件，直接进行分类研究。本文将进行这项工作。通过分析，我们提出连接件的六种形式—控制连接：同步与异步和软连接与硬连接组合的四种；数据连接的软连接与硬连接两种。通过且仅通过这六种形式的连接方式，就可以构造各种体系结构的形式。

采用该分类方法描述系统，可以清晰地说明一个系统的软件体系结构，与传统的线框图相比，能更好地暴露系统的非功能性需求在设计上的实现。基于此，我们提出一种新的评价一个软件体系结构复杂性的方法，并给出一个用于优化软件体系结构的设计示例。

2　连接件分类

2.1　连接件的概念

几乎所有的软件体系结构的结构模式观点都认为[2]：软件体系结构部件间的联系即连接件的作用与部件等同，这也是软件体系结构研究的重要成果之一。但是，对连接件并没有一个统一的定义，一般考虑其接口、类型、语义、限制及细化等问题[10,11]。

连接件从其完成连接的功能上讲，应该包含两个成分：连接方式和连接协议。前者表明一种构成连接的通路，是相对静态的，是一种实现上的结构，属于语法层。连接协议说明构成连接的内容，是相对动态的，与应用程序相关的，属于语义层。

例如：在实现 ftp 的客户和服务器连接的结构中，连接方式是指：在 ftp 客户和 ftp 服务器之间存在着控制连接和数据连接这两种连接通路，这两种通路各自有其自身的特点；而连接协议是指：具体在控制线路上出现何种控制命令，就会相应地在数据线路上出现何种数据，它们之间的顺序关系。

在比较常见的 Wright 形式化描述方法中，由于将连接方式和连接协议混在一起描述，造成了对连接件描述上的复杂。而事实上，通过对连接件的连接方式的细致分析后发现：连接方式其实并不多，且是固定的。真正复杂和多变的是与应用相关的代表语义部分的连接协议。对协议的描述，不应采用线框图，而应采用其它的描述形式(如：Petri 网)等更为强大的描述工具。

2.2　连接的类型

在软件体系结构中，部件只能通过连接件进行交互。这种连接关系从功能上可以分为两大类：控制类连接和数据类连接。

1　控制连接　从实现的角度上，控制连接是指：一个组件对另一个组件存在着激活/调用的联系。这种控制连接的关系可以按照不同的标准分为不同的类型。

*)本文课题的研究得到自然科学基金(60073045)和国防预研基金(00j15.3.3.JW0529)的资助。严昌浩　硕士，主要研究方向为实时操作系统设计，操作系统 I/O 结构分析。刘云生　教授，博士生导师，主要从事主动，实时等非传统数据库及集成的研究。张文彬　副教授，主要从事计算机语言原理，实时领域编译语言研究。

·从时间角度，控制连接发起者(caller)和被激活者(callee)之间相互协作的关系可分为同步连接和异步连接。同步连接是指：在连接发生时，即：caller 发起连接后，必须等待被 callee 处理请求；而后者也必须在执行完请求后，返回到前者的等待处。其中：caller 的等待方式也即 callee 的返回方式必须是事前约定的。异步连接是指：caller 发出连接请求，callee 被激活，caller 不用等待，同时，callee 也不用返回。例如：我们常见的过程调用是同步连接，中断例程的激活是异步连接。

·从空间角度，caller 和 callee 相互协作的关系可分为硬连接和软连接。区分它们的标准是：caller 和 callee 是否在同一个逻辑地址空间。若在同一个地址空间，则为硬连接，否则，为软连接。例如：一般常见的过程调用是硬连接；在 Windows 操作系统中，应用程序进程中的消息响应函数，被操作系统的外部消息激活，这样的连接方式是软连接。

从时间和空间的两种不同角度的分类是正交的。因此，可以产生如下四种连接方式：

(1)同步硬连接关系：如最常用的过程调用模式，在 caller 与 callee 之间，一方面，它们是同步的，另一方面，它们是"直接跳转"。

(2)同步软连接关系：RPC(远程过程调用)在语法和语义上与普通的过程调用很相似，但是在 caller 与 callee 之间，存在着命令的缓存机制。

(3)异步硬连接关系：硬件中断处理系统就是异步硬连接的典型例子，如硬件中断发生器与中断响应函数(ISR)之间的控制连接。

(4)异步软连接关系：对于某些不需要同步应答的连接，如一般的 Windows 下的 CallBack 类型的函数就是一个例子。

以上四种连接关系的结构特性如下：

(1)在硬连接中，连接发起者和连接被激活者必须在同一个逻辑地址空间中。也即：处于不同逻辑地址空间中的部件之间，连接的方式只能是软连接，而不可能用硬连接。

(2)软连接存在不确定的连接时延。连接时延是指：caller 发出连接请求的时刻与 callee 启动执行请求的时刻之间的差的绝对值。硬连接的时延是可以事先确定的。并且，一般硬连接的时延要小于软连接的时延。

(3)软连接在实现上，存在命令缓冲机制。因此，callee 必须存在消息泵机制。消息泵是指：在部件中存在的一种类似死循环的检索消息队列的循环结构。异步软连接中，calee 有消息泵，而同步软连接中，双方均应有消息泵。

(4)异步连接的双方，必须在不同的执行体调度单位中(线程/进程)。否则，如果在同一个调度单位内，那它们的执行序列将是确定的，而不是异步的。

(5)异步被激活者在执行时刻，不能对其处的执行单位(进程/线程)的上下文环境作任何的假设。

2 数据连接 只有两种类型：共享物理存储和数据消息机制。类似地，称共享物理存储的数据连接为硬数据连接，称数据消息机制的数据连接为软数据连接。

两种数据连接的结构特性有：(1)硬数据连接必须使得两个部件的数据共享区在它们的逻辑地址空间中。也即：若两个需要数据联系的部件之间，如果没有共享的逻辑地址空间，它们不可能使用硬数据连接，而只能使用软数据连接。(2)在数据消息机制的连接方式中，数据的接受方必须存在数据泵机制。因此，若是单向的数据传输，只需要在接受方有数据泵，若为双向的数据传输，则双方均需要数据泵。

3 连接类型的分析比较

同步连接和异步连接在语义上的差别是巨大的，典型的例子是 Windows 和 Unix 程序设计上的重大差异。在 UNIX 下的编程风格是一种同步的，而在 Windows 下，是以消息响应为主，围绕消息响应函数而展开，是异步的，同时，这一编程结构上的差异，对程序员界面的影响也是巨大的。我们往往只承认这种差异，但却很少去考虑，是什么造成了这种差异。其实，若把应用程序和操作系统看作两个大的部件，则它们之间的连接图分别为图1和图2，其中：实直线表示同步硬连接，实折线表示异步硬连接，虚折线表示异步软连接。

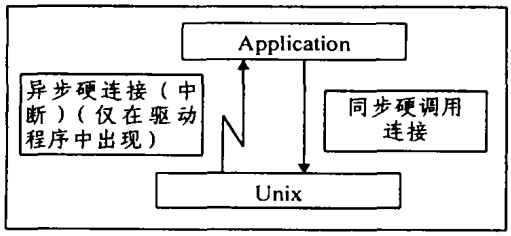

图 1

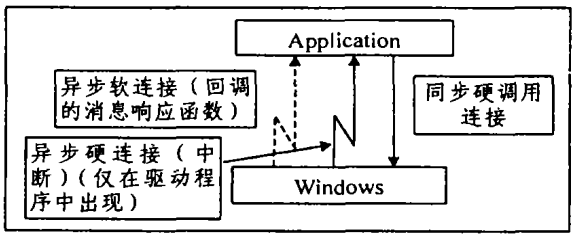

图 2

从中可以看出，Windows 下的应用程序和操作系统之间的连接比在 UNIX 下多出了一个异步软连接，(并且，它在实际的编程中占的比重很大)。因此，我们也就不难理解，它们的编程风格有很大的不同，而且，从体系结构上讲，Windows 编程要比 UNIX 复杂。其中，也可以看出，驱动程序，由于往往要考虑异步硬连接(硬件中断)，因此，它要比普通的应用程序在连接件的种类上又要多出一个，同样，驱动程序要比普通应用程序复杂。这一点可以从我们的直觉中得到印证。

直观上，同步连接和异步连接的区别是显著的，但软连接和硬连接的区别并不是那么明显，但为什么还要区分它们呢？对于同步连接，硬连接和软连接的典型例子分别是：直接过程调用和 RPC。它们虽然在语义上的区别不大，但是，在实现上，RPC 一般是由底层的 RPC 软件包实现，它与直接由计算机硬件实现的过程调用还是有些区别的。例如：在系统失效上，RPC 更容易出现问题，即使没有硬件故障，也可能出现软件通路失效；在系统分布上，由于不需要共享逻辑地址空间，RPC 显然比直接过程调用适应性强；在响应延迟上，硬连接显然要小且确定等等。这些差别，往往都是一个系统的非功能性需求的重要组成部分。我们在针对不同的系统需求，构造总体软件体系结构时，系统的功能性需求，一般取决于部件(component)，而非功能性需求，就体现在连接件上。也就是，不同的连接类型，将产生不同的系统非功能性需求。

在计算机系统中，采用软件或是硬件实现，在逻辑功能上，是没有什么差别的，但是，就是这个软硬件划分界面形成了不同的计算机系统结构。同样，在软件体系结构中，部件之间的连接采用软连接或硬连接，也会构成不同的软件体系结

构。

4 对 Wright 标记法不足的分析

为了更准确地说明问题，用比较常见的 Wright 标记法，形式化地定义各种连接，分析 Wright 标记法的不足之处。下面，下划线表示该事件为主动发出。

(1)同步硬连接和同步软连接

```
Style Procedure-call
Component Caller
Port p = call→ return → p ∏ §
Computation = internalCompute → p. call→
  p. return → Computation ∏ §
Component Definer
Port p = request →return→ p □ §
Computation = p. request → internalComputation → p. return →
  Computation □ §
Connector Procedure-call-Link //直通结构，其实现方式决定了连接的软、硬
Role Caller =call→ return → Caller ∏ §
Role Definer = call → return→ Definer □ §
Glue = Caller. call → Definer. call→ Glue □
  Definer. reply → Caller. return→ Glue □ §
//Constraints, Configuration Attatchment (略)
EndConfiguration
```

(2)异步硬连接

```
Style Interruptor-ISR
Component Interruptor
Port p = request→ recover → p ∏ §
Computation = internalCompute → p. request→
  p. recover → Computation ∏ §
Component ISR
Port p = request → recover→ p □ §
Computation = p. request → InternalComputation (p. recover) →
  Computation □ §
Connector Interruptor-ISR-Link
  //直通结构，硬件实现
Role c = request→ c ∏ recover → c ∏ §
Role s = request → s ∏ recover→ s □ §
Glue = c. request → s. request→ Glue □
  s. recover → c. recover□ §
//Constraints, Configuration Attatchment (略)
EndConfiguration
```

(3)异步软连接

```
Style PostMessage-GetMessage
Component PostMessage
Port p = request → p ∏ §
Computation = internalCompute →
  p. request → Computation ∏ §
Component GetMessage
Port p = request → p □ §
Computation = p. request → internalComputation → Computation □
  §
Connector PostMessage- GetMessage-Link
  //直通结构，且为单向直通，软件实现
Role c = request → c ∏ §
Role s = request → s □ §
Glue = c. request → s. request→ Glue □ §
//Constraints, Configuration Attatchment (略)
EndConfiguration
```

从中我们可以看出：同步和异步在连接件和部件的处理上，按 Wright 标记，存在着显著的差别，说明能够区分它们。但是，同步的软连接和同步硬连接在 Wright 标记中是完全一样的，而异步的软连接和硬连接也几乎是一样的。唯一的差别也仅仅是由于某些硬连接需要关闭连接请求，但关闭的时刻与请求的执行没有任何同步的要求。由此可见，在 Wright 表示法中，不能有效区分硬连接和软连接的差别。

软连接和硬连接存在对系统结构(特别是性能)有重大影响的可能，然而，Wright 标记法却不能容易区分它们，这说明它屏蔽了某些不应该屏蔽的细节。其实，Wright 标记方法重在描述组件之间的协议部分，即：进程间的逻辑通讯关系，而完全没有本文所指出的软硬连接的区分。

5 对软件体系结构复杂性的评价

基于上述连接件分类方法，对一个软件系统的体系结构复杂性，提出一个评价原则：

一个软件系统的结构复杂性，并不取决于系统中连接件的总数量，而是系统中连接件种类的数量。

应用这一原则，可以定性地分析软件系统的结构复杂性，如：前文中提到的比较 Windows 和 UNIX 应用系统的结构复杂性的评价，实际上，就是采用的这个原则。

除了可以定性地分析软件系统的结构复杂性外，我们还可以将该原则用于简化软件系统的设计结构。例如：在我们当前正在做的实时嵌入式微内核操作系统的 I/O 结构设计中，其中一个关键的问题是，设计一个良好的驱动程序界面，给驱动程序员(不是普通应用程序员)一个简洁、有效的编程界面，而这个问题也就是：如何设计驱动程序与操作系统内核、硬件、应用程序之间的连接关系？

传统操作系统(UNIX/Windows)和微内核操作系统的驱动程序与系统的连接关系分别如图3和图4。其中，在微内核操作系统中，模块间的连接均是采用消息通讯的方式，也就是本文中的软连接的类型。因此，在系统的结构上，它们与传统操作系统是不同的。而这一点，在一般的线框图中是无法区分的。

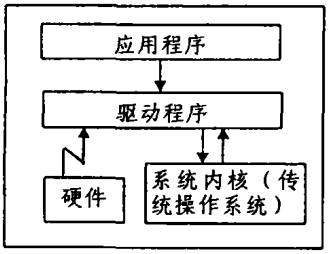

图 3

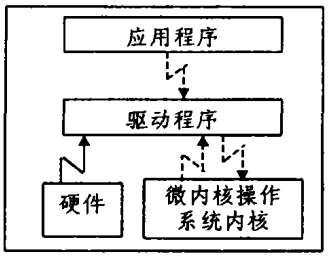

图 4

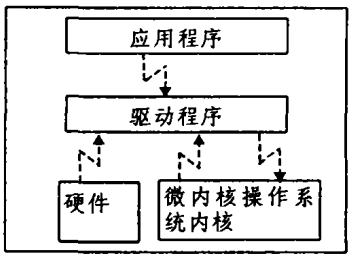

图 5

比较两者，我们发现，采用微内核结构原本是为了增强模块间独立性，但在驱动程序中却并没有减少它的结构复杂性(它们均存在两种连接方式)，为了简化系统结构设计，一种可行的方法是：将硬件和驱动程序间的硬异步连接，改为软的异步连接，如图5。

这样，驱动程序的结构复杂性就降低了，给驱动程序设计员的是这样一个简单的界面：处理各种不同的 I/O 请求(读/写/打开/关闭/中断响应)，其中，中断响应也简化为一个"普通"的需要处理的请求而已。当然，中断请求的优先级远高于一般的 I/O 请求，而这一点，由系统内核中的带优先级的 IPC 机制对用户(驱动程序员)屏蔽。当然，由于对硬件中断采

(下转第72页)

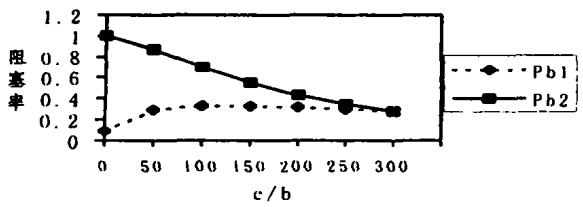

图4 代价因子变化时阻塞率的变化情况

第2组仿真结果表明了当 ρ1和代价因子 b 保持不变而 ρ2 变化时，点播阻塞率 Pb1，Pb2 的变化趋势，如图5所示。其中参数设定如下：C＝300，ρ1＝270，ρ2：0-350，c/b＝80。由图可以看出，当 c/b 保持恒定时，随着第二类点播请求到达率的增加，第二类点播请求的阻塞率迅速增加，而第一类点播请求的阻塞率增长幅度要远小于第二类点播请求。由上可知，在这种情况下基于代价因子的不公平接纳控制算法对第一类点播请求起到了很好的保护作用。

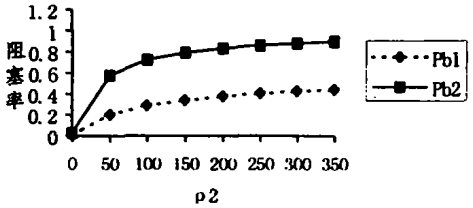

图5 ρ2变化时阻塞率的变化情况

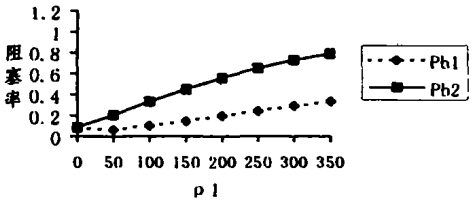

图6 ρ1变化时阻塞率的变化情况

第3组仿真结果表明了当 ρ2和代价因子 b 保持不变而 ρ1 变化时，点播阻塞率 Pb1，Pb2 的变化趋势，如图6所示。其中参数设定如下：C＝300，ρ1：0-350，ρ2＝100，c/b＝80。

由图可以看出，当 ρ1较小时，第一类点播请求的阻塞率基本保持不变，而第二类点播请求的阻塞率迅速增大。当 ρ1大到一定程度时，系统容量的限制开始起作用，第一类点播请求和第二类点播请求的阻塞率都迅速增大。事实上在这种情况下，代价因子对第一类点播请求的接纳限制不大，系统容量才是主要的限制条件。

结束语 通过对上述仿真结果的分析，我们可以看出基于代价因子的不公平接纳控制算法通过对代价因子的调节实现了对不同类别节目的不公平接纳。上述算法对于提高基于多级存储的分布式 VOD 系统的整体服务指标和服务效益有很大帮助，因而具有较大的实用价值。

参 考 文 献

1 Burbeck D W，Rowe L A. Hierarchical storage management in a distributed void system. IEEE Multimedia，Fall 1996. 37～47

2 Siu-Wah L，John C S L. Scheduling and data layout policies for a near-line multimedia storage architecture. Multimedia System，1997，5：310～323

3 Vin H ，et al. An observation-based admission control algorithm for multimedia servers. In：Proc. of the 1st IEEE Intl. Conf. on Multimedia Computing and Systems，May 1994. 234～243

4 Reddy A L N，Wyllie J. Disk Scheduling in a Multimedia I/O System. ACM Multimedia，1993. 225～233

5 Diagle S J，Strosnider J K. Disk Scheduling for Multimedia Data Streams. ACM Multimedia Computing and Networking，1996，2188：212～223

6 Kaufman J S. Blocking in a shared resource environment. IEEE Transactions on Communications，1981，COM-29(10)：1474～1481

(上接第142页)

用软的连接处理，可能会带来中断响应延迟过大的问题，但至少，在系统的结构图上，可以清晰地看到这一点。

结论 本文的最初动机是为了回答一个问题；为什么操作系统的驱动程序比一般的应用程序在结构上要复杂。在尝试各种分析方法后，最终，从软件体系结构的角度来对其进行剖析；才引发了对连接间实现结构的细致分析。在本文中提出如下观点：

1)连接件在结构上，应分为两部分：连接结构和协议。其中，连接结构是静态的；协议是与应用相关的，动态的。并建议在描述连接件的整体结构时，将其两者分开描述。

2)连接件的实现结构限定为6种。控制连接(4种)为：同步硬连接、同步软连接、异步硬连接、异步软连接。数据连接(2种)为：软数据连接和硬数据连接。其中，软连接和硬连接这种差别对整个系统的结构存在着产生重大影响的可能。但某些形式化的标记方法很难直观地区分它们。

3)提出一种新的评价软件体系结构复杂性的方法。

参 考 文 献

1 Shaw M，Garlan D. Software Architecture：Perspectives on an Emerging Discipline. Englewodd Cliffs，NJ：Prentic-Hall，Inc. 1996

2 周之英. 现代软件工程(中). 北京：科学出版社，2000

3 Allen R J. A Formal Approach to Software Architecture. CMU-CS-97-144，1997

4 Garlan D，Monroe R，Wile D. ACME：an architectural interconnection language：[Technical Report CMU-CS-95-219]. 1995

5 Shaw M. Comparing Architectural Design Styles. IEEE Software，1995，12：11

6 Gamma E，et al. Design Patterns：Elements fo Resuable Object-Oriented Software. Addison Wesley Longman，Inc. 1995

7 黎伟建. 领域特定的软件体系结构. 中山大学学报(自然科学版)，1998. 6

8 张家晨，等. 基于主动连接件的软件体系结构及其描述方法. 软件学报，2000. 11

9 Hoare C A R. Communicating Sequential Process. Englewodd Cliffs，NJ：Prentic-Hall，Inc. 1985

10 Ducasse S，Richner T. Executable Connectors，towards Reusable design elements. In：Jazageri M，Schauer H，eds. Proc. of the sixth European Software Engineering Conf. (ESEC'97) Berlini：Springer-verlog，1997

11 Allen R，Garlan D. A formal basis for architectural connection. ACM Trans. on Software Engineering and Methodology. 1997，6(3)