一种基于消息机制的通用框架

李文烨　顾毓清
（中国科学院软件研究所　北京100080）

A Common Message-Based Framework

LI Wen-Ye　GU Yu-Qing
(Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)
E-mail: liwenye@263.net

Abstract　Software architecture has changed the way of programming by shifting developers' focus from lines-of-code to coarser-grained architectureal elements and their interconnection structure. Furthermore, a flexible architecture makes it easier to identify and evolve the elements involved in a change. In this paper, we present a common messge-based framework which simplifies programming and is flexible to accommodate architecutral changes. A case study suggests that our framework effectively aids in both program constructiong and software evolution.

Keywords　Software architecture, Component, Connector, Message

1　引言

正如 Perry 和 Wolf[8]所预想的那样，软件体系结构近年来成为了软件工程领域研究的焦点。他们提出了一个模型，将软件体系结构定义为一个具有特殊形式(form)的结构元素(element)的集合，并用一个基本理论(rationale)的集合加以详细描述。在文[3]中，软件体系结构被粗略地定义为系统结构：一个特定系统的体系结构是由一组用于计算的构件以及用于描述这些构件之间交互作用的连接子组成的。这个定义在文[9]中被进一步扩充为：软件系统的体系结构以描述构件和构件之间的交互的形式定义了整个系统。关于这个定义的进一步详细解释参见文[10]。

研究软件体系结构的目的在于降低软件开发的代价，并且发现更多的相关产品线的不同应用之间潜在的共性[5,8,10]。基于通用体系结构的软件开发使得人们可以从考虑体系结构元素(软件构件及连接子)及其内部关联出发来进行软件的构造。在软件体系结构中明确提出了连接子(connector)的概念，使得人们将一个系统中用于通讯的元素同用于计算的元素区分开来，从而更利于简化程序的构造。

本文提出了一个框架，借助于连接子的支持，实现了支持分布的构件协同工作的目标。设计该框架的另外一个目标是尽可能增加构件之间连接的灵活性，藉此达到更好的通用性。另外，这种框架很好地支持了结构经常变化的应用系统的特殊需要。我们将这种框架应用在一个公文流转系统的系统原型中，取得了很好的效果，从而进一步验证了这种框架的灵活性，可以很好地支持应用程序结构的变化。

除了上述的目标之外，这种框架也同时支持(或者潜在地支持了)一些其他的目标，使得使用该框架构成的系统的对外接口被标准化，系统可以同时以集中和分布式方式运行，组成系统的构件可以分别实现，然后集成。

2　相关工作

在过去几年中，在结构化设计领域一个重要的进展就是在系统结构设计方面提出和使用大量惯用模式，形成了不同的体系结构风格，体系结构风格又以结构组织模式的形式定义了一系列系统[10]。

在软件体系结构领域另外一个重要的进展是体系结构描述语言(ADL)及其相关支撑工具的提出，这些语言和工具支持了基于体系结构的程序开发过程。简单地讲，“体系结构描述语言着重于应用程序整体的高层结构而不是任何源程序模块的具体实现”[12]。在过去的几年里，体系结构语言已经成为软件体系结构领域研究的重点。

人们已经提出了一些体系结构描述语言，并且用在一个特定领域或者作为一个通用的体系建模语言来描述、建模、校验和实现软件体系结构。这些体系结构描述语言包括：Aesop, ArTek, C2, Darwin, LILEANNA, MetaH, Rapide, SADL, UniCon, Weaves, Wright 等[6]。除此之外，人们还设计了一种体系结构交换语言 ACME，这种语言作为软件体系结构设计工具的一种通用交换格式，专门用来实现不同软件体系结构描述语言之间的相互集成。在文[6]中给出了一个分类框架，区分并比较了这些体系结构描述语言，并且指出了各种不同语言的主要特性。

不同的体系结构描述语言着重于不同的应用领域、不同的体系结构风格以及用它们所建模的体系结构的不同方面[6]。因为本文所提出的框架采用了 C2体系结构描述语言所涉及的某些概念，所以首先简单地介绍一下这种体系结构描述语言。关于 C2的详细资料可以参阅文[4～11]。

C2体系结构风格最初主要被设计用来支持具有图形用户界面(GUI)的一类特殊应用，并且具有支持一些其他类型的应用的潜在能力。这种风格下的应用程序，其结构可以简单地描述成为一个由不同的并发构件(Component)所组成的一个网络，这些并发构件通过连接子(Connector)进行连接，连接子在这个网络中起消息路由的作用[1]。每一个构件和连接子都被指定一个确定的上部和下部，每个构件的上部可以同一个连接子的下部进行连接。每个构件的下部可以同一个连

李文烨　硕士研究生，主要研究领域为软件工程技术，数据库技术；顾毓清　教授，博士生导师，主要研究领域为软件工程技术。

接子的上部进行连接。不允许构件与构件之间不经过连接子而进行直接的连接，每个连接子可以连接任意数目的构件或者其他连接子。在C2体系结构中，所有的通讯都是通过消息的交换来完成的。图1是一个C2体系结构的例子[5]。

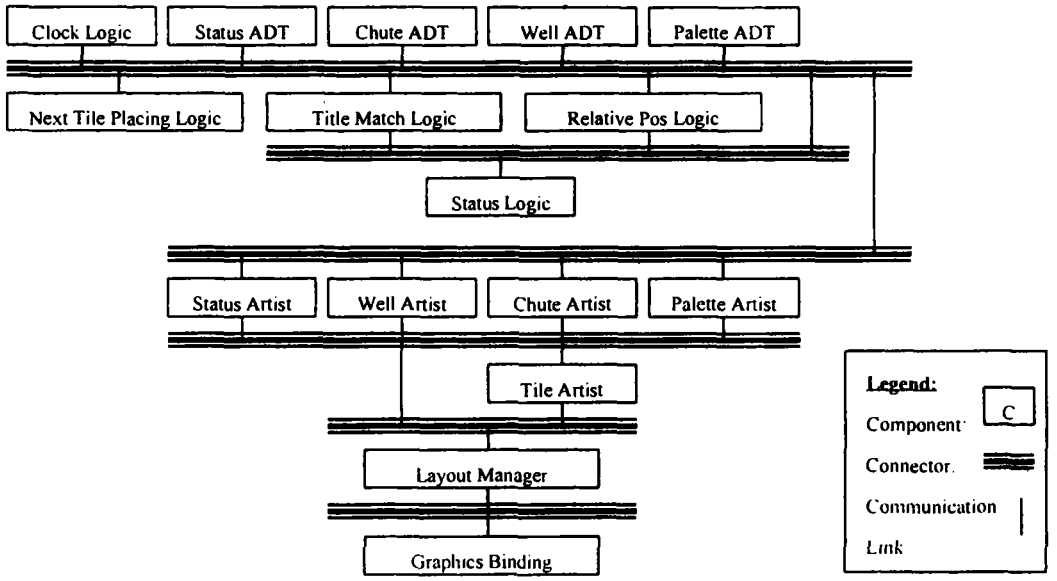

图1 C2体系结构示例

3 应用框架

C2体系结构很好地支持了GUI类应用以及一些其他类型的应用程序。但另外一个方面，我们发现一些系统并不适合采用C2体系结构风格进行表达，比如传统的数据流系统中的管道模型[10]，原始的数据必须经过许多分离的处理步骤后才能最终完成处理过程。

为了充分地表达数据流及其他模型，我们设计并实现了一个通用的程序框架，还采用了与C2体系结构风格类似的某些概念。这个框架同样可以被看作一个由连接子所连接的许多构件所组成的一个网络。构件和连接子分别定义了明确的前向和后向端口：对于构件而言，可以有任意数目的前向或者后向端口；但是每个连接子有且仅有一个前向和一个后向端口。连接子的前向端口可以连接到一个构件的后向端口，连接子的后向端口可以连接到一个构件的前向端口，但是不允许构件到构件或者连接子到连接子的直接连接（参见图2）。

C2结构风格中消息（message）的概念也被引入了我们的框架，包含两类消息：请求和通告。请求和通告消息在框架中沿着不同的方向进行传递（关于消息在框架中传递的方向，参见3.4）。连接子负责进行构件之间的消息路由。

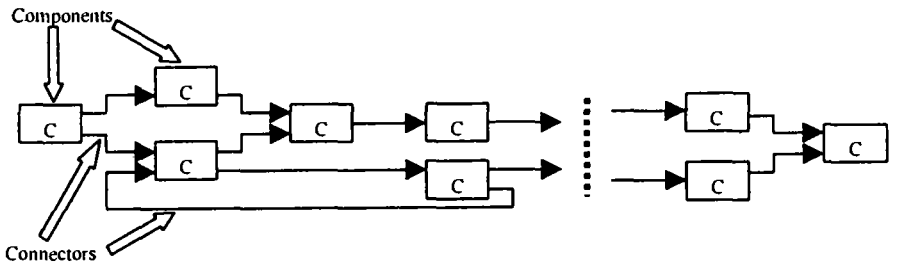

图2 框架示例

虚线部分代表框架中未画出的部分，连接子箭头的方向表示框架的正向

3.1 端口（Port）

端口可以被视为框架中不同元素（构件或者连接子）之间的连接点。每个与外界连接的元素都可以具有一个或者多个前向和后向连接点，两个不同元素之间的连接只能通过端口之间的连接进行。我们的框架支持两类不同的连接：构件的前向端口与连接子的后向端口之间的连接，也称为构件的前向连接或者连接子的后向连接；连接子的前向端口与构件的后向端口之间的连接，也称为构件的后向连接或者连接子的前向连接。图3给出了不同元素之间通过端口进行连接的情况。

为了方便后面的讨论，我们在这里也同时给出框架方向的概念。框架的正向，就是沿着构件的前向端口到达相连接的某个连接子的后向端口，然后通过该连接子的前向端口到达下一个构件的后向端口的方向；反之，我们则定义其为框架的逆向。

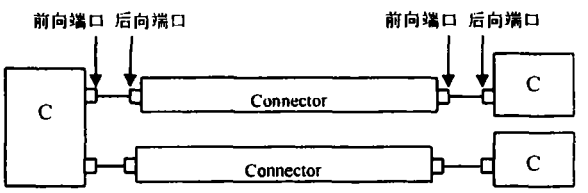

图3 不同元素之间通过端口的连接

因此，可以讲，一个端口标示了框架中的一个连接作用点。事实上，这个作用点也可能代表了一个很复杂的相互作

用。比如说，我们可以在这个连接点中定义一个过程调用的接口。尽管端口在框架构建的过程中起着重要的作用，但是绝大多数时候为了简化起见，这些端口对应用系统设计人员和具体编程人员来讲是透明的。在本文以后的图示中，也将隐含端口这个概念。

3.2 构件(Component)

在我们的框架中，构件表示了系统中用于计算的元素。一个构件可以拥有任意数目的前向端口和后向端口，其中的每一个端口与一个连接子的相应端口相连接。

框架中提供了复合构件(Composite Component)的概念。一个复合构件是由互相连接的构件及连接子组成的一个集合，这个构件集合整体上表现出了与单一的构件相类似的对外连接特性。在图4中，虚线框中的构件和连接子可以被视为一个复合构件，另外两个单独的连接子从前向和后向分别连接了这个复合构件到构件 C6和 C1。复合构件的作用既可以等同也可以区别于每一个单独的构件，框架中对于构件复合的支持使得我们可以使用该框架对于要建模的系统分层进行描述。

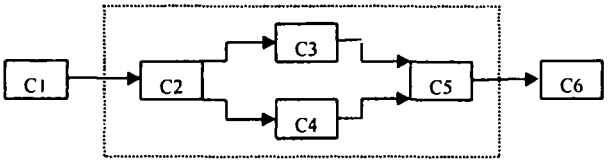

图4 复合构件

3.3 连接子(Connector)

连接子用于连接不同的构件或者同一构件，在构件的交互作用中充当了消息流动的路径。每一个连接子恰好有两个端口：一个前向端口，一个后向端口。连接子的前向端口可以连接到某个构件的后向端口之一，连接子的后向端口可以连接到某个构件的前向端口之一。

另外，连接子在框架中还具有消息过滤及路由的功能，用以确定是否让某条消息通过，或者确定如何将一条消息发送到一个相连接的构件。

由于其自身的特点所限，连接子并不直接支持消息的多点广播(Multicasting)。但是，多播的功能可以通过一个构件和多个连接子共同协作完成。在图5中，引入了一个辅助构件(Broadcaster)，用于协作实现多播，将构件 Component1产生的消息发送到多个相连接的构件。

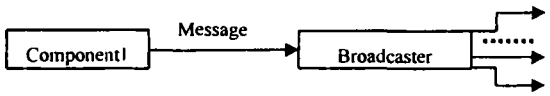

图5 多播模型

3.4 消息(Messages)

对于在构件之间信息的流动，是通过异步的消息机制进行的，所传播的消息遵从一个预先定义的一致的接口。框架支持两类消息：请求和通告。请求沿着框架进行正向的流动。通告沿着框架逆向流动。通常情况下，请求消息中包含了要被下一个构件处理的数据(或者指向数据的指针)，而通告消息则通常体现了构件的运行状态。相对于 C2结构而言，通告消息的作用被弱化，在某些时候甚至可以从框架中移除，这主要是因为在某些系统中构件的处理状态并不总是需要被其他构件所了解。

当请求消息到达一个构件之后，构件负责处理这条消息：可能会更改消息中所包含的数据内容，然后决定是否继续正向传送该条请求，或者终止该条消息的传送，构件同时还要决定是否发送一条相应的通告消息。相应地，当通告消息到达一个构件之后，构件可能会更改消息中所包含的数据内容，然后决定是否继续逆向传送该条通告，或者中止该条消息的传送。如果构件中止了一条消息的传送，我们说该条消息被消化(consumed)了；如果构件接受了一条消息，并且继续传送，我们说这条消息被转发(transferred)了。

3.5 组合规则及消息流动

框架支持两种组合规则：

①一个构件可以在前向和后向连接任意数目的连接子；
②一个连接子的前向和后向分别最多只能连接到一个构件。

下面，我们按照不同构件在框架的组合中所处的不同位置对其做一次分类。

假定 front_connector 是一个构件前向连接的连接子集合，back_connector 是一个构件后向连接的连接子集合。按照组合规则，我们定义如下构件：

① $InitialComponent\ (C) \equiv C.front\_connector \neq \Phi \wedge C.back\_connector = \Phi$

② $IntermediateComponet\ (C) \equiv C.front\_connector \neq \Phi \wedge C.back\_connector \neq \Phi$

③ $EndingComponent\ (C) \equiv C.front\_connector = \Phi \wedge C.back\_connector \neq \Phi$

④ $IsolatedComponent\ (C) \equiv C.from\_connector = \Phi \wedge C.back\_connector = \Phi$

上述对于构件的分类，有助于加深对于系统框架的理解，并且有助于将系统框架简化，表述为一个图的形式：在这幅图中，顶点表示构件，有向边表示连接子，有向边的方向表示框架的正向。

在下面的陈述中，我们给出一个示例来解释不同元素之间消息流动的关系。图6中，“C_j”代表一个构件，“C_{b1}”，“C_{b2}”，…，“C_{bm}”代表了“C_j”后向连接的一组连接子，“C_{f1}”，“C_{f2}”，…，“C_{fn}”表示了“C_j”前向连接的一组连接子。

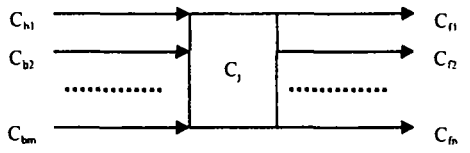

图6 消息流动示例

假定 *back_in* 是一个元素(构件或连接子)的后向端口所收到的请求消息的集合，*front_out* 是一个元素的前向端口发送的请求消息的集合，*back_out* 是一个元素的后向端口发送的通告消息的集合，*front_in* 是一个元素的前向端口所收到的通告消息的集合。为了保持框架中消息的正常流动，必须满足如下基本条件：

$$①C_j.back\_in = \bigcup_{i=1}^{m} C_{bi}.front\_out$$

$$②C_j.back\_out = \bigcup_{i=1}^{m} C_{bi}.front\_in$$

$$③C_j.front\_in = \bigcup_{i=1}^{n} C_{fi}.back\_out$$

$$④C_j.front\_out = \bigcup_{i=1}^{n} C_{fi}.back\_in$$

在上面的四个等式中，等式1反映了“C_j”的后向端口所收

到的请求消息来自于其后向连接的连接子的前向端口所发送的请求消息，等式3反映了“C_i”的前向端口所收到的通告消息来自于其前向连接的连接子的后向端口所发送的通告消息，等式2反映了“C_i”的后向端口所发送的通告消息至少被其后向连接的一个连接子的前向端口所接受，等式4反映了“C_i”的前向端口所发送的请求消息至少被其前向连接的一个连接子的后向端口所接受。如果这4个等式所陈述条件之一不满足，就意味着框架中流动的消息会发生丢失的情况。

在某些特定的情况下，需要对框架或者框架的某一部分做进一步的约束，使其满足或者部分满足如下4个等式：

⑤$C_j.\ front_out = C_j.\ back_in$

⑥$C_j.\ back_out = C_j.\ front_in$

⑦$\forall\ i, j\ 1 \leqslant i < j \leqslant m \Rightarrow C_{bi}.\ front_in \cap C_{bj}.\ front_in = \Phi$

⑧$\forall\ i, j\ 1 \leqslant i < j \leqslant n \Rightarrow C_{fi}.\ back_in \cap C_{fj}.\ back_in = \Phi$

等式5和6描述了一个仅仅转发，但是自身不产生新的消息的构件“C_i”；等式7和8描述了“C_i”所产生或者转发的消息最多被一个连接子所接受的情形。

最后，按照构件在消息处理的过程中所发挥的作用的不同，我们给出另外一种分类方法：

①$Request\ Producer(C) \equiv \neg(C.\ front_out \subseteq C.\ back_in)$

②$Request\ Pr\ oducer^{*}\ (C) \equiv Request\ Producer\ (C) \wedge C.\ back_in = \Phi$

③$Notification\ Producer\ (C) \equiv \neg(C.\ back_out \subseteq C.\ front_in)$

④$Notification\ Producer^{*}\ (C) \equiv Notification\ Producer\ (C) \wedge C.\ front_in = \Phi$

⑤$RequestConsumer(C) \equiv \neg(C.\ back_in \subseteq C.\ front_out)$

⑥$RequestConsumer^{*}\ (C) \equiv RequestConsumer\ (C) \wedge C.\ front_out = \Phi$

⑦$NotificationConsumer(C) \equiv \neg(C.\ front_in \subseteq C.\ back_out)$

⑧$NotificationConsumer^{*}\ (C) \equiv NotificationConsumer(C) \wedge C.\ back_out = \Phi$

值得注意的是：采用如上方法划分的8类构件并不是完全正交的。例如：一个构件既可以是请求消息的生产者(RequestProducer)，同时可以是请求消息的消费者(RequestConsumer)。

3.6 框架与框架实例(Framework & Instance of Framework)

根据如上的讨论，我们首先给出框架实例的概念：按照组合规则的要求，连接起来的一组构件和连接子，构件之间通过连接子进行双向的消息传递。在框架实例定义的基础之上，我们定义框架为全体框架实例所组成的集合。为了充分阐释框架运作的机理，我们给出示例图7，它描述了一个 Client-Server 的框架实例。在图8中，我们采用 ACME 语言描述了这个实例。可以从中看出，在这个实例中一共涉及了3个构件和3个连接子：每个构件具有一个或多个端口(事实上，构件也可以没有任何端口，参见3.2)，每个连接子恰好有两个端口。构件和连接子通过端口连接在一起，构成了整个系统。

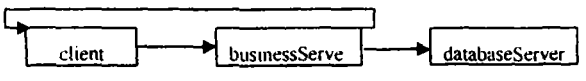

图7 Client-Server 结构

4 应用实例

为了测试本文所设计的这个框架的能力，我们构造了一个公文流转系统的原型。传统上，这类系统大多是运行在类似于 Lotus Notes 这样的平台之上的。在这儿，我们采用了“Browser-Server”的风格构造了这个原型，原型的结构如图9所示。

```
System cs Demo={
    Component businessServer={
        port displayArtist;//a front port
        port business Handler;//a back port
        port sqlRequestor;//a front port
    };
    component databaseServer={
        Port sqlHandler;//a back port
    };
    component client={
        Port businessRequestor;//a frontport
        Port displayRender;//a back port
    };
    connector conn1={
        Ports { frontPort;backPort}
    };
    connector conn2={
        Ports {frontPort;backPort}
    };
    connetor conn3={
        Ports{front Port;backPort}
    };
    Attachments{
        client. businessRequestor to conn1. backPort;
        businessServer. businessHandler to conn1. frontPort;
        businessServer. sqlRequestor to conn2. backPort;
        databaseServer. sqlHandler to conn2. frontPort;
        businessSever. displayArtist to conn3. backPort;
        client. displayRender to conn3. frontPort;
    };
};
```

图8 Client-erver 结构的 ACME 描述

图9的右侧是这个系统原型的核心，可以把它看作是一个模拟了传统的办公自动化系统中的真实公文流程的独立引擎。这个引擎由3部分构成：初始化装置、事件扫描器和公文流程模拟器。初始化装置主要负责引擎的初始化工作；事件扫描器负责观测外部环境，当预定义事件发生的时候，向公文流程模拟器发送相应消息；公文流程模拟器负责模拟真实的公文流程，它实际上就是一个本文所设计的框架的一个实例：公文流程中的每一步处理过程都被建模成为一个构件，公文信息以消息的形式通过连接这些构件的连接子在构件之间进行流动。

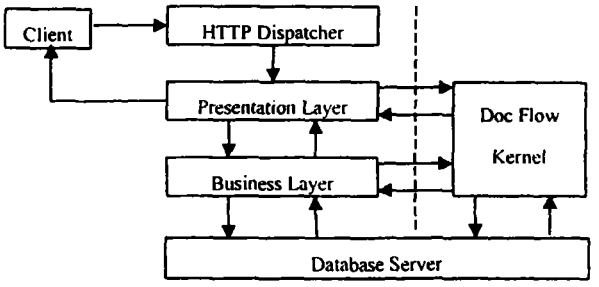

图9 公文流转系统结构

当应用程序启动的时候，初始化装置首先从数据库或者文件中读取公文流程配置信息，然后建立起公文流程模拟器的相应结构。与此同时，事件扫描器也被建立起来，并且持续扫描预先定义的事件源(在本原型中，将数据库中数据内容的变化作为预定义的事件源)。当预定义事件(数据库数据发生变化)发生时，事件扫描器将该事件以消息的形式通知公文流

程模拟器，然后模拟器开始处理公文的流转过程，处理的结果被保存在数据库中，或者以消息的形式对外宣布。外部环境（比如图9左侧的表示层或者业务层处理逻辑）就可以获得当前正在处理的公文的状态信息，也可以通过预定义的消息接口通知各个构件如何处理公文。

这个原型的测试结果是相当成功的。在此之前，构造类似系统的一个主要难点就在于如何设计一个足够灵活的框架来适应公文流程变更的需要。在我们的系统中，由于公文流程模拟器是动态构建的，能够适应动态结构变更的需要，因而很好地解决了这个问题。总的来讲，本文所提出的框架提供了类似系统的设计和开发所需要的灵活性，用于适应程序结构变更的需要；另外一方面，完全基于消息机制的通讯方式也使得系统的内部和外部的接口得以规范和简化。

5 进一步讨论

5.1 与有向图的比较

我们在3.5节提到，本文所论述的框架的每个实例可以表示成一个有向图，其中顶点表示构件，有向边表示连接子，有向边的方向表示框架的正向。在这里，我们给出一个定理，用来描述本文所述框架与有向图集合的关系。

假定 F 是按照组合规则构建的全体框架实例的集合，G 是全体有向图的集合，则如下定理成立：

Theorem 1. $F \cong G$.

为了证明这个定理，我们仅仅需要建立一个从 F 到 G 的双射函数。很容易证明我们在本小节第一段所描述的构造方法满足双射函数的要求，因此我们在这里省略了证明步骤。

5.2 与结构化程序设计语言的比较

我们也可以从另外一个角度出发来考虑本文所提出的框架：可以把请求消息的流动过程与传统的结构化程序设计语言（比如 Pascal 或者 C 语言）的控制流程从结构上做一个比较，如图10所示。

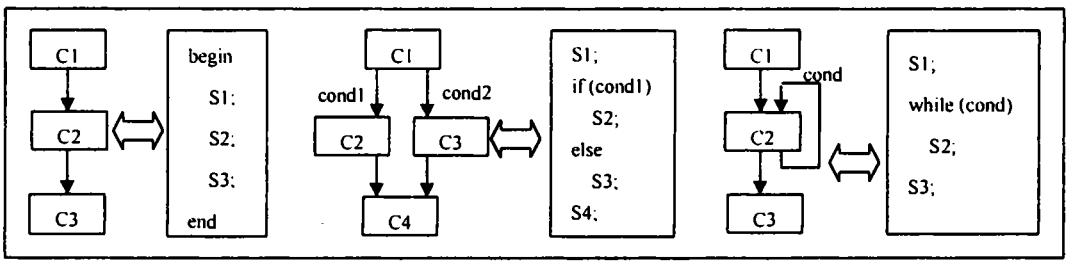

图10 与结构化程度设计语言在结构上的比较

在图10中，我们建立了从结构化语言的3种控制结构（顺序结构、分支结构、循环结构）到某个框架实例的对应。通过这种对应，我们可以看出，本文所涉及的框架可以从结构上表达类似于传统的结构化程序设计语言的语句流程的控制流转。这种流转是消息在不同的构件中的传递，相对于结构化程序设计语言不同语句之间的转移来讲，这种流转是在一种更高的层次进行的。

5.3 从集中到分布

本文所论述的框架，其设计目标之一就是同时支持集中和分布的运行环境。在集中的运行环境下，所有的构件运行在单一的进程空间中；在分布的运行环境下，每一构件运行在不同的计算机上（例如点到点的运行方式）。

为了达到这个目标，我们采用了 C2的处理方式，使得每一个构件运行在一个单独的线程环境之下，构件之间不需要共享任何地址空间。这种方式提供了必要的前提，使得使用该框架构造的应用系统可以不加修改（或者只需要很小的改动）便可以运行于分布式环境下。为了充分利用这种特性，还有一些工作需要完成，还需要建立一些用于构件和连接子之间通过网络进行通讯的基础设施。另外，我们还可以做进一步的探讨：能否建立一种映射机制，可以把这种框架的任一实例在集中和分布的环境下自动进行双向的映射。

总结 本文提出了一种基于消息的通用框架，并且介绍了它的组成和工作原理。还介绍了这种框架的优点，其中最主要的就在于框架连接配置的灵活性。当然，还有一些其他的设计目标也被很好地实现。

为了充分探讨这种框架的特性，我们给出了这个框架的一个参考实现，并且利用这个参考实现完成了一个公文流转系统的原型。对于这个原型的测试结果是相当好的：由于框架自身易于动态构造的特性以及其基于消息的通讯机制，我们解决了构造此类系统的一个主要困难，构造出来的系统也是足够灵活和满足用户需要的，系统的对外接口也是规范的。

除此之外，我们还从其他方面揭示了框架的一些特性。将框架的结构与传统程序设计语言的语句流程进行了比较，从而也反映了框架在一定程度上的通用性。

致谢 我们工作的最初设想来自于很多方面，其中最主要的来自于 University of California，Irvine 计算机系设计的 C2体系结构风格。我们感谢 C2研究小组的成员们和他们所作的开放性研究。我们还要感谢北京亚士帝公司的技术人员，他们在本文完成的过程中提供了很多有益的帮助。

参 考 文 献

1 Dashofy E M，et al. Using Off-the-Shelf Middleware to Implement Connectors in Distributed Software Architectures. In：21st Intl. Conf. on Software Engineering，Los Angeles，CA，May 1999

2 Garlan D，Monroe R，Wile D. ACME：An Architecture Description Interchange Language. In：Proc. CASCON'97，Nov. 1997

3 Garlan D，Shaw M. An Introduction to Software Architecture. Ambiola & Tortola (eds.)，Advances in Software Engineering & Knowledge Engineering，vol. II，World Scientific Pub Co.，Singapore，1993.1～39

4 Medvidovic N. Formal Definition of the Chiron-2 Software Architectural Style：[Technical Report UCI-ICS-95-24]. Department of Information and Computer Science，University of California，Irvine，Aug. 1995

5 Medvidovic N，Taylor R N. Exploiting Architectural Style to Develop a Family of Applications. IEEE Proceedings Software Engineering，1997，144(5-6)：237～248

6 Medvidovic N, Taylor R N. A Framework for Classifying and Comparing Architecture Description Languages. In: Proc. of the Sixth European Software Engineering Conf. together with the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich, Switzerland, Sept. 1997
7 Medvidovic N, Rosenblum D S, Taylor R N. A Type Theory for Software Architectures: [Technical Report]. University of California, Irvine, Apr. 1998
8 Perry D E, Wolf A L. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 1992, 17(4): 40～52
9 Shaw M, DeLine R, Klein D V, et al. Abstractions for Software Architecture and Tools to Support Them. IEEE Transactions on Software Engineering, 1995, 21(4): 314～335
10 Shaw M, Garlan D. Software Architecture: Perspectives on an emerging discipline. Prentice-Hall, 1996
11 Taylor R N, Medvidovic N, Anderson K M, et al. A Component- and Message-Based Architectural Style for GUI Software. IEEE Transactions on Software Engineering, June 1996
12 Vestal S. A Cursory Overview and Comparison of Four Architectural Description Languages: [Technical Report]. Honeywell Technology Center, Apr. 1996
13 Pratt T, Zelkowitz M. Programming Languages: Design and Implementation. 3rd Edition. Prentice Hall, Englewood Cliffs, NJ, 1996

(上接第86页)

①最终选择 ISSP 方法的时机不同。在一次执行中，传统过程在选择 ISSP 方法之后才运用某种方法或方法组合进行企业调查和方案写作。而 ISSP 二次选择规范过程对 ISSP 方法进行了两次选择，首先使用定性分析的方法，结合各种 ISSP 方法的特点，选择两种或两种以上的方法或方法组合(下简称方法)作为预选方法。然后，战略规划组分别按照这几种方法进行企业调查和规划，再通过计算机仿真，在仿真结果的基础上所有组参与综合评价，对这几种方案进行第二次选择，将其中的一种作为最终方案。

②组织机构不同。从图2可以看到，组织机构的设置比原来更加细化。传统的 ISSP 过程由企业信息系统领导小组管理，但是由于这个小组中大多数是高层领导和各部门领导，还有很繁重的日常工作，很难对 ISSP 过程进行有效的管理，而且，传统过程当中对新系统的部署有确定的管理模式，而规划却没有确定的管理模式，往往只有一个非常概括的时间计划。还由于缺乏管理，这个时间计划难以有效地执行。在二次选择规范过程中，由企业信息系统领导小组授权，项目管理部管理 ISSP 项目。在整个过程当中，信息系统领导小组不插手干涉具体事务，由项目管理部统一管理与 ISSP 相当的各种资源，这样才有可能进行有效的管理，使最后的规划结果避免受组织内政治活动干扰，得到尽可能科学的结果。

③对外包规划和自行规划的态度不同。传统过程中对外包规划和自行规划采取不同的态度。如果企业信息系统战略规划项目外包，则项目是所有者和承包商紧密协作的成果，承包商没有蓝图可循，因为承包商的目标就是提供蓝图。如果企业将信息系统战略规划外包给咨询公司，对于企业来说，是整个项目成果的所有者，而咨询公司是承包商。一般的外包项目中，所有者投入资金、提出要求，最后即能获得所需的成果。但是在信息系统战略规划这类特殊的管理咨询项目中，企业要配备一个专家小组配合承包商，这个专家组必须熟悉本企业的战略、组织结构、企业文化、信息管理等各个方面的情况。可以说，企业信息系统战略规划项目不可能进行完整意义上的外包，寄希望于承包商独立提出一揽子的解决方案是不现实的。这类项目中，管理咨询公司存在真正的意义在于结合自身的知识和经验，使用科学的方法和程序来诱导和协助企业专家小组得到最后的规划文件。另一方面，企业一般不具备自行进行 ISSP 规划的能力，必须引入外部专家。这样在 ISSP 二次选择规范过程的设计中，不截然划分外包和自行规划的界限，而是通过设立不同的小组，将外部专家和内部人员结合起来。根据企业情况的不同，可以在小组中安排不同比例的外部专家，通过参与环节的限制，使引入外部人员的成本得到控制。

对 ISSP 重视的程度不同。ISSP 二次选择规范过程比传统的 ISSP 过程复杂得多。而且，由于使用了更加复杂的组织结构，按照两种以上的方法得到规划方案，理论上会使 ISSP 的成本上升。但是，由于 ISSP 不是目的，而是手段，如果能够通过保证 ISSP 的质量来避免不正确的信息系统战略规划方案出台，保证提高企业很长一段时间内建设信息系统的效果，就可以有效地降低企业信息化的总成本。企业可以通过限制外部专家的数量，限制预选 ISSP 方法的数量来降低这一阶段的成本，但是绝不可以通过简化过程或者合并两个小组来降低成本。ISSP 二次选择规范过程充分重视了信息系统战略规划中的关键作用，并认为企业应当把更多的注意力和资源投入信息系统战略规划中。

结语 不论是学术界还是企业界，一直普遍对 ISSP 在整个企业信息化中的关键作用认识不足。除了 ISSP 方法以外，对 ISSP 的过程、管理模式一直缺乏研究。反观传统的 ISSP 过程，它最大的优点就是简单直观，但是直观的方法不一定就是好方法。这种传统的和直观的信息系统评价方法限制了对 ISSP 过程的改造。笔者分析了传统 ISSP 过程的缺点，并提出了信息系统的定量分析与仿真的方法作为信息系统分析和评价工具，使重新设计 ISSP 的过程成为可能。

本文提出的 ISSP 二次选择规范过程，是同信息系统定量分析与仿真、以定量仿真为基础的信息系统综合评价等方法、系统工程的思想和方法、项目管理模式等理论、方法和技术紧密结合的，其基本思想是实用主义的。由于对 ISSP 过程的研究还处在初始阶段，它仍有待在实践中得到进一步的发展和检验。

参考文献

1 林健，张玲玲. 中国 MRPII/ERP 实施现状及分析[J]. 五邑大学学报(自然版)，2001，3
2 张玲玲，林健. 企业管理中信息技术的影响力分析[J]. 企业经济，2001，4：61～63
3 张玲玲，林健. 企业 IS/IT 战略规划模型框架研究[J]. 系统工程，2001，2：33～36