软件协同模型中多模式可配置通道的研究*)

韩 亮 赵 洁 陶先平 吕 建
(南京大学 软件新技术国家重点实验室 南京 210093)

摘 要 软件协同模型中的控制驱动模型很好地实现了计算与协同的分离,是一种重要的软件协同模型。通道在模型中表示协同部分,具有重要的地位。然而现有的控制驱动协同模型中,通道的设计较为简单,不能满足应用的需求。本文利用软件体系结构将连接子作为第一类实体的思想,提升通道在控制驱动协同模型的地位。首先对软件协同的独立要素进行分解,并使用 Interceptor 模型表示各个要素;然后对这些独立的协同要素进行组合,提出基于 Interceptor 的组合配置技术,以此对通道进行扩充,提出多模式可配置通道。多模式可配置通道表达能力得到加强,并且集成人员可以对其进行组合配置,以满足不同应用的需求。

关键词 控制驱动协同模型,通道,Interceptor,组合配置

Research on the Configurable Multi-Mode Channel in Software Coordination Models

HAN Liang ZHAO Jie TAO Xian-Ping LU Jian
(Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

Abstract Control-driven software coordination model is one important category of software coordination models, because of the fine separation of computation and coordination. Channel, which represents the coordination part in the model, plays a very import role. However, in the existed control-driven coordination models, channel is designed in a relatively simple way; therefore, they cannot meet the application requirements. This article adopts the idea of regarding connector as the first-class component in software architecture and enhances the status of channel in the model. Firstly, the independent concerns are decomposed from the software coordination and then modeled as the interceptors. Secondly, a composition method based on interceptors is presented for concern composition. The new channel is more expressive and configurable.

Keywords Control-driven coordination model, Channel, Interceptor, Composition and configuration

随着大规模并行计算和分布式程序设计技术的发展,人们对软件协同问题的研究兴趣日增。在计算机软件领域,影响最广泛的协同概念来自于 Linda 的作者 Carriero 和 Gelernter 从程序设计的角度对协同的定义[1]:

Coordination is the process of building programs by gluing together active pieces. (协同是通过将一组主动的片断粘合起来的方式来构作程序的过程)他们提出

Programming = Coordination + Computation 倡导在分布式程序设计中将(分布的)协同与(局部的)计算分离的思想,指出对协同的描述是与对计算的描述同等重要的软件组成成分。

在上述协同概念的基础上,新的协同模型不断出现。协同模型为主动独立的被协同实体之间的交互的表达提供了一个框架。Farhad Arbab 在综述[2]中将协同模型和协同语言分为数据驱动和控制驱动两类。数据驱动协同模型关注的主要是协同实体之间的数据交换。这类协同模型大多要用到共享的数据空间。协同实体除了需要完成自身的计算任务外,还需要通过调用协同原语从外界(共享数据空间)获取数据或者向外界提供数据,以此来与其它实体协同。数据驱动协同模型的典型代表是 Linda[3]。

控制驱动协同模型,又称作状态驱动协同模型,关注的是协同实体的状态变化与环境的变化。通常协同实体被视为一个具有良好接口定义的黑盒。协同实体通过端口(Port)接收别的协同实体发送的消息和接收广播事件感应外界环境的变化,从而引起实体状态的改变,并通过端口向其他实体发送消息和事件广播向环境发送事件,引起环境的改变。控制驱动的协同语言通常使用通道(Channel)来连接协同实体的端口。通道是两个协同实体间传输协同消息的媒体。文章的第 1 节将对控制驱动模型进行较为详细的介绍。控制驱动协同模型和协同语言的典型代表是 Manifold[4]、Darwin[5]。

数据驱动的协同模型将协同的代码与计算的代码混合,导致各自的语义不够明确,修改较为困难;而控制驱动的协同模型使用具有良好定义的接口的黑盒表示协同实体,独立的模块表示协同,各自语义明确,实现计算代码和协同代码完全的分离,有利于计算模块和协同模块的设计、调试、维护和复用。所以,控制驱动模型是一类非常重要的协同模型。

按照计算与协同之和等于程序的思想,在控制驱动协同模型中,黑盒表示的协同实体表示了计算部分,而通道表示了

*)基金项目:国家重点基础研究发展规划 973 资助项目(2002CB312002);国家自然科学基金资助项目(60273034);国家 863 高科技发展计划资助项目(2002AA116010);江苏省基础研究计划(BK2002409)。韩 亮 硕士生,主要研究领域为对象技术,移动 Agent 技术;赵 洁 硕士研究生,主要研究领域为对象技术,移动 Agent 技术;陶先平 博士,副教授,主要研究领域为对象技术,移动 Agent 技术;吕 建 博士,教授,博士生导师,主要研究领域为软件自动化与形式化方法,移动 Agent 技术。

协同部分。用以表示协同部分的通道的表达能力的强弱直接影响协同模型表达能力的强弱，所以通道在控制驱动协同模型中具有举足轻重的地位。

然而，通过研究，我们发现当前所提出的控制驱动协同模型对于连接协同实体端口的通道研究，仅仅局限于简单的连接和动态重定向[2,12]。简单的通道不能满足以下需求：

1. 功能需求。在基于协同与计算分离思想的系统的开发中，端口的不匹配(比如数据格式的不匹配)成为阻碍通道顺利连接的因素，即功能异构。

2. 实际应用系统对通道提出非功能性需求，比如加密通道、实时通道、可靠通道等。

3. 软件系统的集成人员希望能够对通道进行组合配置，以满足应用的具体需求。

显然，当前的控制驱动协同模型和语言不能达到要求。而软件体系结构为协同模型中通道的研究提供了新的思路。软件体系结构一般是指该系统的构件组成、构件之间的相互关系和连接方式，所形成的系统结构配置，及其动机、模式和约束等[6,7]。一般的系统结构描述语言(ADL)包含下列基本建模成分：构件(Component)、连接子(Connector)和体系结构配置(Architecture Configuration)。

文[8]提出连接子是模型中的一个重要的组成部分，应该与构件有同等的地位，属于第一类(First-class)实体。近十年来，学者对连接子做了大量的研究，包括不匹配问题[9,14]，配置问题等。在综述[10]中，作者对连接子进行了分类。

本文的主要贡献在于利用软件体系结构将连接子作为第一类实体的思想，提升通道在控制驱动协同模型的地位，使得通道的表达能力得到加强，集成人员可以对其进行组合配置，以满足不同应用的需求。本文的主要工作是首先对软件协同的独立要素进行分解，并使用 Interceptor 模型表示各个要素；然后对这些独立的协同要素进行组合，提出基于 Interceptor 的组合配置技术，以此对通道进行扩充，提出多模式可配置通道。

本文第 1 节介绍和分析控制驱动软件协同模型，同时在分析现有控制驱动协同模型中通道能力的不足的基础上给出研究动机；第 2 节提出基于 Interceptor 技术的多模式可配置通道；第 3 节是一个应用实例的分析；第 4 节简要介绍 Artemis M²C 系统及多模式可配置通道在其中的应用；第 5 节进行相关工作比较；最后总结全文并提出进一步工作。

1 控制驱动协同模型及研究动机

1.1 控制驱动协同模型

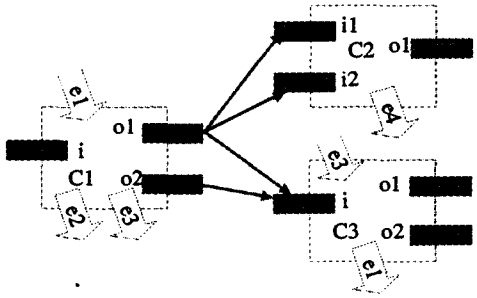

图 1 基本模型图

控制驱动协同模型中包含的实体有四类：协同实体(Entity)：协同实体是一个具有良好定义端口的黑盒模型，实体通过端口与外界进行信息交换。端口(Port)：端口是协同实体用来与外界交换信息的单元，类似于 I/O 系统中的 Read 和 Write。不失一般性，端口可以分为读端口和写端口。实体通过读端口读取信息，通过写端口发送信息。通道(Channel)：通道实现端口之间的连接。通道接收写端口发出的消息，然后通过底层的网络连接发送至读端口。不同的消息体现了不同的协同要求。消息在通道里面一般来说采取异步的消息传递方式(Message Passing)。事件(Event)：事件机制是控制驱动协同模型中另一个交换信息的方法。实体可以进行事件的监听和广播。

控制驱动协同模型关注实体与环境状态的变化，协同实体通过接收消息和事件感受外界变化，并通过发送消息和广播事件改变环境。实体通过输入端口接收消息，触发状态改变，然后通过输出端口向外界发送消息。端口之间的连接可以是一对一的，也可以是多对一、或者一对多的。端口也可以通过广播的方式向外界发送事件消息，同时可以接收广播事件。很多协同语言对事件进行参数化，携带部分数据。

图 1 为一个简单的示例[2]。示例中包含三个协同实体，C1、C2 和 C3。C1 包含一个输入端口 i，两个输出端口 o1、o2。C2 和 C3 类似。C1.o1 通过通道连接到 C2.i1、C2.i2 和 C3.i；C1.o2 通过通道连接到 C3.i。C1 通过广播接收事件 e1，发送事件 e2 和 e3；C2 和 C3 类似。

1.2 研究动机

通道在控制驱动协同模型中表示了协同部分，具有举足轻重的地位。然而通过研究，我们发现当前所提出的控制驱动协同模型对于连接协同实体端口的通道(Channel)研究，仅仅局限于简单的连接和动态重定向[3,12]。而实际的应用对通道提出更高的要求，简单的通道不能满足以下需求：

1. *功能需求* 在基于协同与计算分离思想的系统的开发中，协同实体都是独立开发的，甚至是为了复用而开发的。独立开发的构件对外界环境作自己的假设[14]；在协同模型中，协同实体对外界所作的假设主要包括数据格式和传输协议两个方面。假定之间的不匹配，导致协同实体之间存在异构性，不能完成信息的交换。

数据格式方面包括：数据类型、数据组织(比如组织成一个复杂数据类型传输和多个简单数据类型)、传输顺序。传输协议的不匹配指的是协同实体可能使用不同的协议通信。可能的协议有基本的 TCP、UDP 协议，应用层的 HTTP 协议、SOAP 协议等，或者用户自定义的协议。

2. *非功能需求* 不同的应用系统对实体间的连接有着不同的需求。这些需求不是功能方面的需求，它不影响交互的成败，但是影响其他方面，比如安全性、实时性、可靠性、容错性等方面，我们称之为非功能需求。

例如在军用系统中，信道的安全要求很高，这时希望通道是加密通道；同时对可靠性要求较高，希望通道是可靠通道。

3. *静态和动态组合配置需求* 系统集成人员可以对一个通道在功能上和非功能上进行配置，进行各种组合，而且通道应该具有良好的重用性。功能需求的满足保证了交互的成功；非功能需求的满足使得交互具有一些特定的性质。同时，软件集成人员希望集成平台能够提供支持自适应的机制，使得系统可以在运行时可对自身的行为进行评测，并通过改变自身的行为，以满足变化的功能要求，或者获得更好的性能。配置可以是在系统运行之前配置，即静态配置；也可以在运行之中进行配置，即动态配置。

为了满足上面提出的三类要求，我们需要对原有系统的

通道进行改进。首先，对通道的功能进行扩充，并且对这些功能进行分类，抽象出具有某种独立功能的简单处理逻辑；在此基础上，对简单处理逻辑的组合进行研究，从而能够表达复杂的处理逻辑。

2 多模式可配置通道

根据关注分离的原则，我们把软件协同模式分解为若干协同要素（如数据格式、时间、安全性、可靠性等）；实体间的交互实际上是这些交互要素的组合。因此我们提出使用 Interceptor 表示这些独立的交互要素，并使用 Interceptor 组合配置技术表示交互要素的组合。

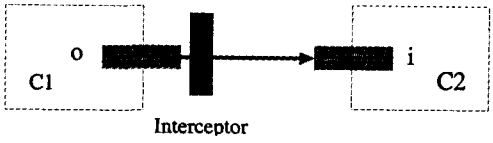

图 2 Interceptor

2.1 Interceptor 概念

如图 2，协同实体 C1 的写端口 o 向 C2 的读端口 i，发送消息。在 C1 端，消息被 Interceptor 截获，作相应的处理后，发送至 C2。

2.2 软件协同中的要素和 Interceptor 表示

我们对软件实体间的交互进行分析，分解出若干协同要素，包括数据格式、所使用的传输协议、时效性、安全性、可靠性、容错性等。

我们使用 Interceptor 描述这些要素。比如，我们使用数据格式转换 Interceptor 截获传输的数据，并对其格式进行转换；使用数据加密 Interceptor 对传输的数据进行加密，从而实现安全传输；使用异常处理 Interceptor 进行传输异常的处理，从而提高系统的容错性。

在软件协同中，对应的常用 Interceptor 包括数据转换 Interceptor、协议转换 Interceptor、可靠传输 Interceptor、数据加密 Interceptor、实时传输 Interceptor、数据压缩 Interceptor、监视器 Interceptor、异常处理 Interceptor 等。下面将对其中的一些进行简要描述，其它的 Interceptor 做法类似，此处不再赘述。

值得提出的是，软件协同的要素是复杂的，穷尽列举是很困难的，并且可能随着软件技术的发展不断地变化，而且在特定的应用中可能存在特殊的需求。因此集成人员希望可以对 Interceptor 的集合进行扩充。系统应该允许集成人员按照 Interceptor 的接口规范，自行开发处理逻辑，从而可以定制满足特定需求的 Interceptor。

2.2.1 *数据转换* Interceptor 主要用来屏蔽协同实体之间数据格式方面的异构性：包括数据类型、数据组织和传输顺序三个方面。对于单向通道来说，数据转换 Interceptor 可以单个出现。可以设置在发送端，也可以设置在接收端。

2.2.2 *可靠传输* Interceptor 主要用来保证消息传输的可靠性。造成消息丢失的原因是各种各样的，但是在网络系统中，主要是网络传输造成的。如果系统底层支撑采用非可靠的传输协议，若要保证可靠传输，需设置可靠传输 Interceptor。

一个简单的实现方法为：在可靠传输 Interceptor 中维持传输状态，在传输失败的时候进行重发。一般来说，可靠传输 Interceptor 设置在发送端。

2.2.3 *数据加密* Interceptor 主要用来保证消息传输过程中的保密性，在安全级别较高的通道中需要使用。一般来说，数据加密 Interceptor 成对出现，包括加密 Interceptor 和解密 Interceptor，分别设置在发送端和接收端。

2.2.4 *实时传输* Interceptor 主要用来保证消息传输的时效性。具体含义为，消息在一定时间内从发送端到达接收端，为有效消息，否则为过时消息。实时传输 Interceptor 也应该为成对出现，分别设置在发送端和接收端。

一个简单的实现方法为：在发送端 Interceptor 中设置发送时间戳和 TTL，在接收端 Interceptor 中进行比较后决定作为有效消息处理还是作为过时消息处理。

2.3 Interceptor 组合配置技术

单个的 Interceptor 只能完成简单的功能，而在实际的应用中，往往需要多种功能。比如在很多军用系统中，同时要求可靠和实时传输，同时还要进行加密；大多数系统中都要进行异常处理。而且简单的顺序组合逻辑在很多时候不能灵活地表达复杂协同逻辑，在下面将作详细阐述。

2.3.1 *组合逻辑*

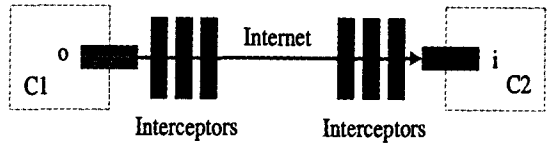

图 3 Interceptor 组

2.3.1.1 *简单组合* 简单的组合逻辑如图 3 所示。多个 Interceptor 通过简单的顺序组合，成为一个 Interceptor 组。Interceptor 组可以完成比单个 Interceptor 复杂的功能。

2.3.1.2 *使用 Control Interceptor 进行组合* 在控制驱动软件协同的基本模型中，一个构件的写端口可以将消息拷贝多份同时发送到多个读端口，如图 1 所示（C1.o1→C2.i1，C2.i2，C3,i），从而可以表示多个实体之间的 Observer 协同逻辑。同样，一个构件的读端口可以接收多个写端口发送的消息，如图 1 所示（C1.o1，C1.o2→C3.i）。

分发器和综合器表示协同逻辑，而构件表示计算逻辑。根据关注分离的原则，二者应该分离。而基本模型中将两者混合在一起，带来很多问题，主要体现在以下几点：

1. 分发器、综合器内部处理逻辑不够灵活。基本模型的分发逻辑实际上是将消息复制后不加选择地发到所有的与之连接的读端口上。综合逻辑实际上是不加处理地直接将所有的消息递交给构件。而在很多情况下，需要更为灵活的处理逻辑，比如在一个拍卖逻辑中，到达的多个消息需要进行相应处理，留下最高的消息，然后再递交构件处理。

2. 分发器、综合器和其他具有协同逻辑功能的代码可组合性差。例如，如图 4 所示。

基本模型中，当需要将分发器和其他协同代码进行组合时，如图 4 左所示，需要将组合代码复制于多处，造成集成的不便和效率的低下。此外，当协同逻辑中包含影响全局状态的代码时，还可能造成错误。将分发器从构件中分离出来，成为独立的协同逻辑，解决了基本模型中存在的问题，易于和其他协同逻辑进行组合。如图 4 右所示。

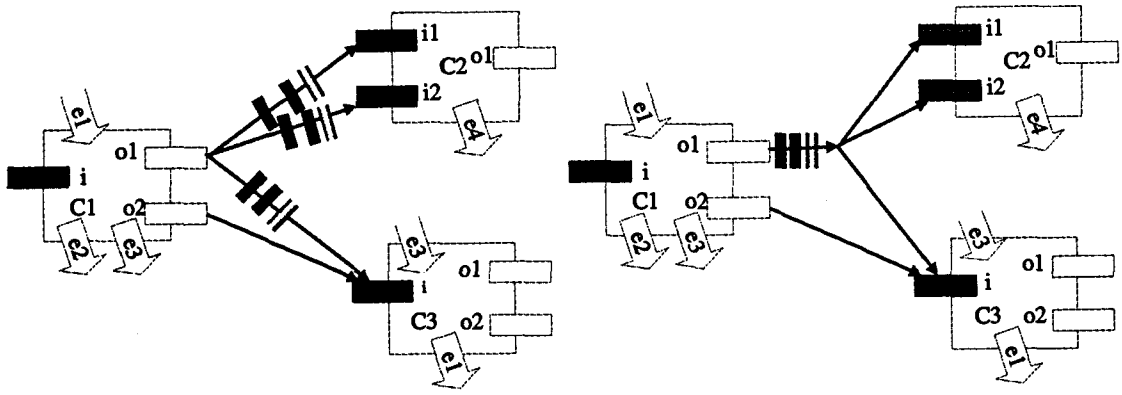

图 4　基本模型中分发器存在的问题示例

本文的基本做法是将分发器从写端口中分离出来，将综合器从读端口中分离出来，并将它们作为协同逻辑集成到通道中。允许集成人员在通道中的设置分发器、分发逻辑和综合器、综合逻辑。

同时，将分发器和综合器分离出来之后，通道可以按照 Composite 的设计模式[15]进行开发和组合，从而很好地实现了通道的复用。

本文的具体做法表述如下。

系统引进两类 Interceptor：Distributor 和 Integrator。如图 5 所示。

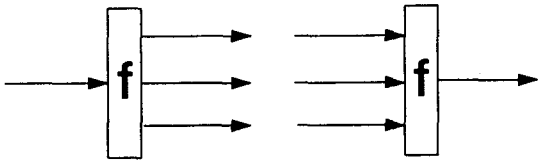

图 5　两类 Interceptor：Distributor 和 Integrator

图 5 左表示 Distributor。Distributor 包含一个 import 端口、多个 export 端口和一个分发函数 f()。f()通过判断，将 import 端口输入的消息进行复制，并传输到部分或者全部 export 端口。图 5 右表示 Integrator。Integrator 包含多个 import 端口、一个 export 端口和一个 Merger 函数 f()。f()通过读取 import 端口的数据，进行相应处理，处理结果从 export 端口输出。

利用 Distributor 和 Integrator 可以灵活地表示复杂的协同逻辑。如图 6 所示。

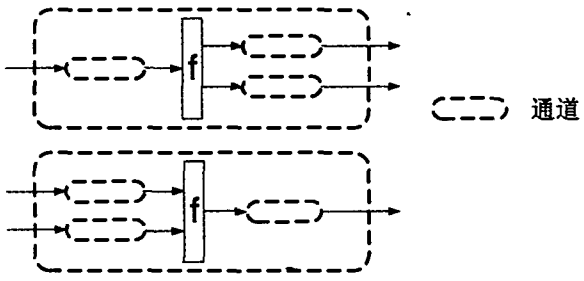

图 6　利用 Distributor 和 Integrator 进行通道组合

通过 Distributor 和 Integrator 的组合还可以提供自适应的设施，如图 7 所示。

一个 Distributor、一个 Integrator 和多个已有的通道组成一个自适应的通道。集成人员可以在 distributor 中设置方法对自身行为和环境进行检测，然后根据条件选择不同的通道(1、2 或者 3)。具体的应用场景如根据不同的环境使用不同的通信协议。

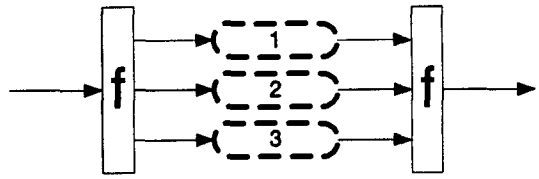

图 7　自适应支持

Interceptor 的组合配置技术为应用提供了强大和灵活的支持，但是，需要注意的是，并不是所有的 Interceptor 的组合都是语法和语义上正确的，主要原因是 Interceptor 之间并非完全正交。除了集成人员应力求保证组合的正确性，系统还需要提供相应的机制进行语法和语义的检查。

2.4　多模式可配置通道

小结第 2 节上述的内容，基于 Interceptor 的多模式可配置通道，结合了原有的基于消息传递的通道和本文提出的 Interceptor 技术、Interceptor 组合配置技术。

3　应用示例

一个简单的应用示例如图 8 所示。手机用户通过无线网络订购计算机并使用银行转账的方式进行付款。代理商负责配机。

系统中使用 Interceptor 对通道进行配置。通道 1 和通道 7 由于使用无线网络传输，带宽较窄，所以配置压缩 Interceptor；在应用需求中，手机用户认为在一定时间没能完成，就取消订单，所以在通道 1 中配置实时 Interceptor；由于不同的手机可能使用不同的通信协议，所以添加自适应的协议转换 Interceptor；为了安全要求，通道 7 中将用户的银行指令通过 Control Interceptor 做日志；通道 2、4、5 中的消息包含数字，涉及单位不一致问题，所以设置数据转换 Interceptor；通道 7 由于涉及银行系统，需要保证安全和可靠，所以配置加密和可靠传输 Interceptor，同时配置异常处理 Interceptor，保证异常情况下的正确处理。

4　Artemis M³C 系统

Artemis M³C 系统是南京大学软件研究所自行研制开发的一个基于 Internet 的网构软件多模式协同中间件。ARTeMIS_M³C 中利用双向的多模式可配置通道技术实现构件之间的多模式协同。

在 Artemis M³C 集成平台上，集成人员通过可视化的方法对连接构件的通道进行配置。集成环境将独立的交互要素

组织成相应的 Interceptor，并利用简单的 Interceptor 组合配置方法，组织成表达应用需求的通道。

Artemis M³C 多模式协同运行支撑环境的基本结构和工作原理如图 9 所示。

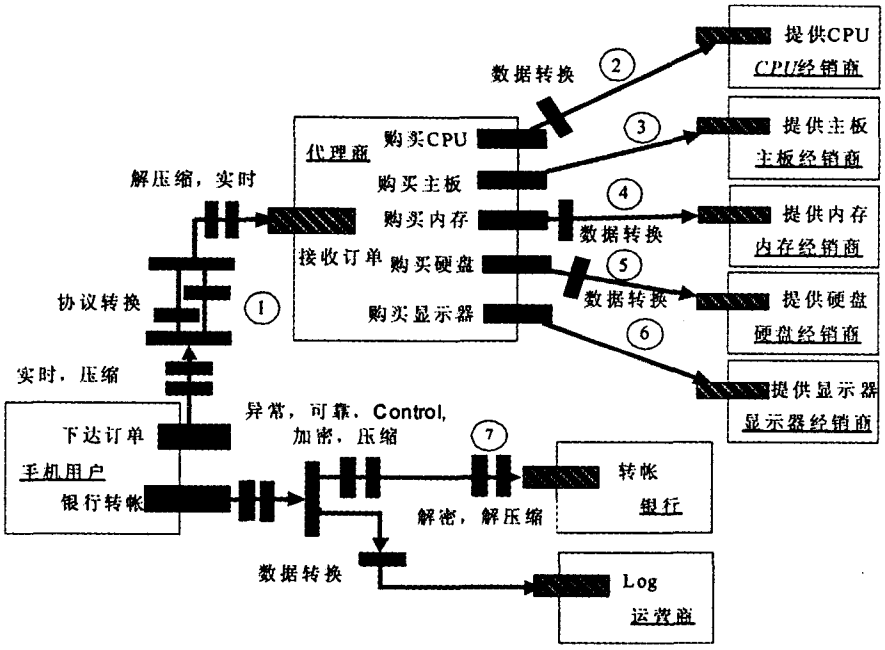

图 8　应用示例

图 9　多模式可配置通道在 Artemis M³C 系统中

系统基于 Web Service 搭建，发送方发送的消息在通道中首先经过客户端的 Interceptor 组处理，然后组织成 SOAP 消息通过网络发送到服务端；同样，消息在服务端通道中也需要经过 Interceptor 组的处理，然后递交构件。构件经过计算，返回的结果同样必须通过基于 Interceptor 的通道，然后到达客户端。

5　相关工作比较

表 1　通道对比

	功能异构处理	非功能特性	组合配置性
Conic	无	可重连接	无
Darwin	无	可动态建立	无
Durra	无	可设置部分参数	通过参数简单配置
TOOLBUS	对实体使用 Adaptor 进行数据转换	类似于 BUS	无
MANIFOLD	无	无	无
多模式可配置模型	通过 Interceptor 进行数据和协议的转换	通过加载 Interceptor 获得安全性、实时性、可靠性、容错性	通过简单组合和利用 control interceptor 组进行通道配置

Farhad Arbab 的综述[2]中提到的控制驱动协同模型和语言有：PCL，Conic，Darwin，Durra，CSDL，POLYLITH，The Programmer's Playground，RAPIDE，ConCoord，TOOLBUS，MANIFOLD。我们选取其中部分具有代表性的模型 Conic[11]，Darwin[5]，Durra[12]，TOOLBUS[13]，MANIFOLD[4]，从功能异构处理、非功能特性、组合配置性三个方面对其中的通道进行比较。

从表 1 可以看出，相对其他协同模型中的通道，多模式可配置通道使用 Interceptor，很好地解决了功能异构的问题，同时获得很多需要的非功能特性，并且用户可组合配置。

总结及进一步工作　本文的主要工作是在原有控制驱动软件协同模型中通道的基础上，首先对软件协同的独立要素进行分解，并使用 interceptor 模型表示各个要素；然后对这些独立的协同要素进行组合，提出基于 interceptor 的组合配置技术，以此对通道进行扩充，提出多模式可配置通道。多模式可配置通道可以很好地解决功能异构的问题，改善非功能特性，而且集成人员可以按照应用的需求灵活配置。开放的系统具有不可预测性，都面临系统动态配置的问题。对通道的基于 Interceptor 的动态配置的研究是非常有意义的。在上文中，提到 Interceptor 的组合需要满足一定的要求，以达到

（下转第 246 页）

1. 对扩展的模糊时间 Petri 网进行进一步的扩展，包括把模糊逻辑和各种时序逻辑、Petri 网结合起来，例如对时序逻辑的可能性分析，可以应用实时计算树逻辑(Real-Time Computation Tree Logic)，这项技术可以用于有限状态并发系统的状态转移图。

2. 把扩展的模糊时间 Petri 网和 UML 等系统分析与建模工具结合起来，使扩展的模糊时间 Petri 网可以应用更复杂的实时系统的建模。

3. 进一步应用扩展的模糊时间 Petri 网与复杂的实时系统的调度、网络 QoS 的分析、多媒体系统的建模，CIMS 系统及 P2P 系统等。

参 考 文 献

1 Marsan M A, Balbo G, Conte G. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. ACM trans. on computer system, May 1984, 2(2)

2 Merlin P, Faber D. Recoverability of communication protocols-implications of a theoretical study. IEEE Trans. Communications, Sept. 1976

3 Little T D C, Ghafoor A. Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks. IEEE Computer. Oct. 1991, 24(1)

4 Diaz M, Senac P. Time Stream Petri Nets: A Model for Multimedia Streams Synchronization. In: Proc. of the First Intl. Conf. on Multi-Media Modeling, Nov. 1993, Singapore

5 Valette R, Cardoso J, Dubois D. Monitoring Manufacturing Systems by Means of Petri Nets with Imprecise Marking. IEEE International Symposium on Intelligent Control, Albany, N. Y., USA

6 Murata T. Temporal Uncertainty and Fuzzy-Timing High-Level Petri Nets. Application and Theory of Petri Nets, Lecture Notes in Computer Science, Vol. 1091, Springer-Verlag, New York, June 1996

7 Zhou Y, Murata T. Fuzzy-Timing Petri Net Model for Distributed Multimedia Synchronization. In: Proc. of the 1998 IEEE Intl. Conf. on Systems, Man, and Cybernetics (SMC'98), La Jolla, Calif., Oct. 1998

(上接第 211 页)

语法和语义的正确。语法的正确性可以由系统的语法检查保证；语义上的正确性虽然从原理上讲，系统不能完成保证，但是可以提供一些手段帮助集成人员进行语义上的检查。

参 考 文 献

1 Carriero N, Gelernter D. Coordination languages and their significance. Communications of the ACM, Feb. 1992, 35(2): 97～107

2 Papadopoulos G A, Arbab F. Coordination models and languages. Software Engineering (SEN) SEN-R9834, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Netherlands, Dec. 1998

3 Carriero N, Gelernter D. Linda in Context, Communications of the ACM, April 1989, 32(4)

4 Arbab F, Herman I, Spilling P. An overview of Manifold and its implementation. Concurrency: Practice and Experience, Feb. 1993, 5(1): 23～70

5 Magee J, Dulay N, Eisenbach S, Kramer J. Specifying distributed software architectures. In: Proc. of the Fifth European SoftwareEngineering Conf. Sep. 1995

6 Shaw M, DeLine R, Klein D, et al. Abstractions for software architecture and tools to support them. IEEE Transaction on Software Engineering, April 1995, 21(4): 314～335

7 Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages. IEEE Transaction on Software Engineering, Jan. 2000, 26(1): 70～93

8 Shaw M. Procedure Calls are the Assembly Language of System Interconnection: Connectors Deserve First Class Status. In: Proc. of the Workshop on Studies of Software Design, May 1993

9 Shaw M. Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging. In: Proc IEEE Symosium on Software Reusability, April 1995

10 Mehta N R, Medvidovic N, Phadke S. Towards a Taxonomy of Software Connectors. ICSE '00, Limerick, Ireland, May 2000

11 Kramer J, Magee J, Finkelstein A. A Constructive Approach to the Design of Distributed Systems, Tenth Intl. Conf. on Distributed Computing Systems (ICDCS'90), Paris, France, 26 May - 1 June, IEEE Press, 1990. 580～587

12 Barbacci M R, Weinstock C B, Doubleday D L, Gardner M J, Lichota R W. Durra: A Structure Description Language for Developing Distributed Applications, Software Engineering Journal, IEEE, March 1996. 83～94

13 Bergstra J A, Klint P. The TOOLBUS Coordination Architecture. In: First Intl. Conf. on Coordination Models, Languages and Applications (Coordination'96), Cesena, Italy, April, LNCS, Springer Verlag, 1996. 75～88

14 Garlan D, Allen R, Ockerbloom J. Architectural mismatch, or why it's hard to build systems out of existing parts. In: Proc. of the Seventeenth Intl. Conf. on Software Engineering, Seattle, Washington, April 1995. 85～179

15 Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns - Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley. 1995