协调技术综述*)

A Survey of Coordination Technology

夏顺东　尤晋元

(上海交通大学计算机科学与工程系　上海 200030)

Abstract Coordination technology is an emerging research area in computer science. This paper gives the definition and taxonomy of coordination technology. Several state-of-the-art coordination models and languages are introduced. At last, future research issues are pointed out.

Keywords Coordination models, Coordination languages, Network computing

1 引言

近年来,计算机的性能、价格、体积在不断发生巨大的变化,对此人们思维中的概念也不断更新:从单一的符号处理、数字计算演变为代表个人组织行使职能,且具有和外界交互的能力。协调技术(Coordination Technology)[1,2]正是一个新发展起来的研究领域,研究如何管理这种交互,从而使计算系统有效地进行协同工作。

协调技术最早起源于20世纪80年代,其标志是Linda语言[3],是为解决大规模并行问题而出现的。20世纪90年代以来,并发系统被广泛研究。并发系统的设计主要关心的是合作模型(Cooperation Model),即组成应用程序的主动实体如何合作运行,由于系统支持的通信原语和并发程序在概念上有较大的差距,造成合作模型往往过于复杂。针对这个问题,Carriero和Gelernter正式提出了协调语言的概念和意义[4]。这种新范型认为并行/分布程序包括两个正交的部分:计算和协调。计算部分由一系列负责处理数据的进程组成,协调部分则负责这些进程间的通信和合作。这个思想在学术界被越来越多的人所接受,产生了许多各具特色的协调语言/模型。近几年,随着网络计算技术的发展,新应用(虚拟企业、电子商务等)和新技术(软件组件、代理/多代理、移动代理等)不断产生,协调领域的研究更加活跃,国外关于协调研究的会议、论文不断涌现。

2 协调的定义及分类

协调不仅仅是计算机科学中的概念。在文[4]中,协调被定义为管理行为间的依赖性,是计算机科学、经济学、运筹学、组织论、生态学等领域中的重要研究对象,是一个新发展起来的跨学科研究领域。

协调模型可以被看作一个三元组〈E,L,M〉,E代表被协调的对象,L代表协调介质,M代表模型的语义框架。协调语言是协调模型的语言形式体现。由于协调技术还处于萌芽期,当前产生了许多风格不同的协调语言和模型,它们一般被分为控制驱动类和数据驱动类,下面两节将分别加以介绍。

3 控制驱动协调

控制驱动协调(Control-driven Coordination)的中心是通信动作,通过控制程序间的交互拓扑结构来实现协调,普遍强调计算体和协调体的分离,用专门的协调语言来描述协调体。下面介绍几种有代表性的控制驱动协调模型和语言。

3.1 Manifold

Manifold[5]是管理独立、并发、合作运行进程间复杂演化的互连关系的协调语言。进程包括计算进程和协调进程。计算进程可以用任何传统语言编写,协调进程则由Manifold语言编写。Manifold中的语法抽象包括进程、端口、事件、流,通过响应事件来动态创建进程实例,通过流在这些进程的端口间建立或重新建立连接。此外,协调进程可以递归地管理其它协调进程,从而可以建立起具有复杂层次结构的协调关系。Manifold的可视化形式是Visifold[6],它可以通过图形方式创建、调试、监视协调过程。Manifold的形式语义基于二级转换系统[7],第一级转换系统定义了单个进程的语义,第二级转换系统定义了第一级转换系统间的交互。

Manifold是一种较完整的控制驱动协调技术,目前还处于发展之中,一般适合于并行环境,对解决分布环境中的协调问题有所欠缺。

3.2 Durra

Durra[8]程序包括一系列组件以及描述这些组件如何关联的一系列配置(Configuration)。组件包括任务和通道。任务含有输入/输出端口。运行时,任务创建进程,通道创建链接,不同进程通过连接输入/输出端口形成复合进程配置。Durra主要是用于协调资源,例如:在不同位置执行程序、转发数据、重新配置程序等。程序的结构同行为被严格区分开来。任务实现程序的功能,通道实现通信功能。

Durra的优点是可以支持分布异构程序的快速原型开发,也可以满足一定的实时性。原则上,Durra可以协调由任何计算语言编写的程序,实际上只较好地支持Ada语言,另外在动态创建任务实例上也有局限性。

3.3 LGI

LGI(Law-Governed Interaction)[9]是一种分布代理(Distributed Agents)间的交互机制,确保小组成员间的交互遵循显式规定的规则。成员间的交互是通过消息传送实现的。LGI的核心概念是$\mathscr{L}$-group,可以定义为一个六元组〈A,CS,M,V,L,E〉:A是该小组内的代理集合;CS是控制状态的集合$\{CS_x \mid x \in A\}$,按照小组的规则L进行演化,每个代理x对应

*)本论文受国家自然科学基金(69973032)资助。夏顺东　博士研究生,研究方向包括:协调技术、网络计算、信息安全、软件工程等。尤晋元　教授,博士生导师,研究方向包括:操作系统、分布对象计算、移动计算、软件工程等。

一个 C_x，由相应的控制器维护；M 是小组成员间受规则 L 控制的消息集合，也称为 L-Message；V 是小组成员中发生的并受规则 L 控制的本地事件集合；L 是模型中的规则；E 是规则强制机制。LGI 控制的四种事件包括：

1)sent(x,m,y)：代理 x 向代理 y 发送的 L-Message m 到达了 x 的控制器 C_x；

2)arrived(x,m,y)：代理 x 向代理 y 发送的 L-Message m 到达了 y 的控制器 C_y；

3)obligationDue(…)：强制执行某个义务(Obligation)；

4)exception(…)：发生了某个异常情况。

LGI 还定义了对控制状态、消息、小组、义务的原语操作。LGI 中有两种小组：显式的和隐式的。前者的成员由一个特殊代理 S_G 管理，后者则不需 S_G。

LGI 的贡献在于对分布式环境提供了一个强制性的、可扩展的、显式的协调机制。问题在于消息中含有位置信息，代理必须知道所要通信对方的名字，这样要引入复杂的名字服务机制。此外，LGI 在实现上代价较大，例如控制器的分布问题，若分布到各个代理所在的机器，必然牵涉到安全性问题，若分布到一些中介机构(如银行)，必然又牵涉到代理如何定位这些中介机构以及中介机构间如何安全通信的问题。

3.4 COCA

COCA (Collaborative Objects Coordination Architectu - re)[10]是一种为电子会议系统设计的协调基础设施。COCA 的思想与 LGI 相似，主要区别在于：

1)一小组内的成员被划分为角色，不同的角色由不同的规则集控制；

2) COCA 虚拟机相当于 LGI 中的控制器，通过特定的门(gate)监听消息。COCA 虚拟机分布在每个成员的机器上。

COCA 的贡献在于将角色的概念引入协调策略的描述中，为运行时的动态重配置(Dynamic Reconfiguration)提供了灵活性。但是协调规则分散在各个角色中，从而不能明显地看出 role 之间的交互关系。目前 COCA 正在朝异构工具的互操作、层次型协调策略及形式语义的方向扩展。

3.5 CoLaS

CoLas[11]是一种主动对象的协调模型，在 Actalk 平台上用 Smalltalk 实现。CoLaS 与 LGI、COCA 的不同之处在于：

1) 协调动作可以影响组中所有成员；

2)协调规则可以通过状态谓词引用包括成员状态在内的协调信息。

CoLas 的核心是协调组，它是一个实体，规定并强制组中的成员执行某个任务。一个协调组由五个元素组成：

1)角色规范(Role Specification)：每个角色关联一个接口；

2)协调状态(Coordination State)：定义了组中协调所需的信息，可以是全局的，也可是局部(只对一个给定角色的参与者可见)；

3)合作协议(Cooperation Protocol)：定义了参与者动作间的连带关系；

4)多动作同步(Multi-Action Synchronization)：参与者交换信息的同步约束；

5)后活跃行为(Proactions)：根据当前的协调状态，协调组必须执行的动作。

CoLas 的贡献在于不但在协调策略中引入了角色，而且能清晰地描述角色间的交互关系。CoLas 的问题在于没有考虑安全机制，缺乏形式语义。

4 数据驱动协调

数据驱动协调(Data-driven Coordination)的中心是通信信息，通过交互中的信息交换实现协调。

4.1 Linda

Linda[3]是数据驱动协调技术家族中最早的成员，其核心是共享数据空间(Shared Dataspace)[12]。共享数据空间是一个公用的可访问的数据结构。所有进程可以间接地通过这个介质通信。它们可以向这个介质广播信息。检索有两种方式：移出、拷贝。由于进程间的通信通过共享数据空间，介质的内容同进程的活动历史是独立的，从而实现了进程在空间和时间上的分离。进程可以将数据发送到介质，然后执行其它任务甚至终止运行，此后其它进程可以异步地检索这个数据。生产者不需知道消费者的标识，反之亦然。这种通信方式也称为产生式通信(Generative Communication)。

Linda 提供了 4 个原语。out(t)：将元组 t 放入元组空间；in(t)：从元组空间里检索元组 t 并移出；rd(t)：从元组空间里拷贝元组 t；eval(p)：将一个主动元组 p(进程)放入元组空间。

其中 rd 和 in 是阻塞式的，out 和 eval 是非阻塞式的。后来又引入了两个非阻塞原语 rdp(t)和 inp(t)，它们在没有找到相应的元组时返回 FALSE。

元组是由类型域组成的序列。从元组空间里检索元组采用结合模式匹配(Associative Pattern Maching)机制。in，inp，rd，rdp 原语中的元组是一个可以含形参的元组模式(Tuple Schemata)，也称为模板。一个元组 T 和一个模板 M 匹配的条件是：

$$(\#(T)=\#(M)) \wedge (\forall j \in \{1 \cdots \#(T)\}: (t_j(M)=t_j(T)) \wedge ((V_j(M)=V_j(T)) \vee (V_j(M)=null)))$$

其中 $t_j(I)$返回元组或模板 I 的第 j 个域的类型，$V_j(I)$返回元组或模板 I 的第 j 个域的值，$\#(I)$返回元组或模板 I 所含域的个数。Linda 是独立于宿主语言的，因此可以和许多计算语言结合起来，这些语言包括 C，Fortran，SETL，Prolog，Gofer 等。

Linda 的贡献在于特有的空间/时间分离性的通信方式。缺点是：语言过于简单，仅表现为一些原语；与计算代码混合在一起；缺乏安全机制；不适于开放环境中的应用。

4.2 T-Spaces

T-Spaces[14]是 IBM 推出的一个软件包，用于协调 Java 对象。T-Spaces 的体系结构与 Linda 相似，元组空间集中在 Server 上，Client 通过原语访问元组空间。T-Spaces 还提供了 eventRegister()函数，允许用户将元组空间里感兴趣的事件与自己的回调方法关联起来。每个元组可以被指定有效期，过了期限后，服务器自动回收。若干元组操作还可以被定义在一个事务中，以实现原子性。T-Spaces 提供了 5 种类型的查询机制：MatchQuery，IndexQuery，AndQuery，OrQuery，XmlQuery。T-Spaces 将 XML 当作一个特殊的域处理，当 XML 域增加进元组空间，自动产生 DOM (Document Object Model)树，查询时遵循 XQL 规范。此外，T-Spaces 还允许用户修改元组的原语操作语义，以及增加新的原语操作命令。

4.3 JavaSpaces

JavaSpaces[15]是 Sun 开发的 Java 对象协调工具，同 IBM 的 T-Spaces 的不同之处在于：

1)JavaSpaces 不能扩展元组操作原语；

2)不支持 XML;

3)JavaSpaces 提出了快照(Snapshot)的概念,以防止 read/take 操作重复使用同一个元组。

4.4 WCL

WCL[16]用于支持 Internet Agent 的协调,可以嵌入任何宿主语言。WCL 在 Linda 的基础上增加了更为丰富的原语操作。增加的原语操作包括:

1)异步访问原语:out_async, in_async, rd_async, check_async(判断特定的 in/rd 异步操作是否返回了元组), touch_async;

2)堆原语(Bulk Primitives):copy_sync, move_sync, copy_async, move_async,它们可以一次对多个元组在两个元组空间之间进行操作;

3)流原语(Streaming Primitives):monitor, bulk_in_async, bulk_rd_async, touch_async, touch_sync,它们让多个元组在 Agent 与元组空间之间流动。

这些原语提高了对元组空间操作的效率。目前 WCL 正在安全性和容错机制上做进一步发展。

4.5 Jada

Jada[17]是一种通过共享对象空间协调 Java 并发线程及 Java 分布对象的工具。Jada 同 Linda 一样是一种最小集的协调语言,但在实现上有所区别,Linda 由于改变了宿主语言的语法需借助预编译器,而 Jada 则基于一些可以访问对象空间的类,允许用户保留标准的 Java 开发工具。为了访问远程对象空间,Jada 使用 Client/Server 方式,如图 1 所示。Tuple Client 位于本地,代表应用程序访问远端的 Tuple Server。Tuple Server 是一个多线程的服务器类,处理请求。Tuple Server 与 Tuple Client 通过 Soket 通信。

Jada 的远程访问牵涉到了寻址问题,Tuple Client 必须在程序中指定 Tuple Server 的地址,这样会降低协调在空间上的分离性。

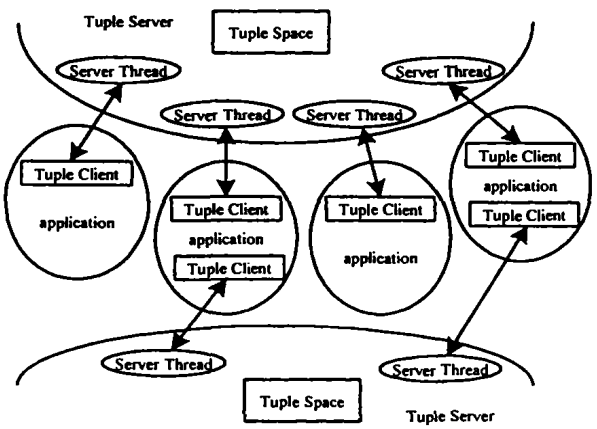

图 1 Jada 的 Client/Server 结构

4.6 Shada

Shada[18]是一种面向对象的协调语言,基于多集重写(Multiset Rewriting)。多集重写思想最先由 Gamma 语言[19]提出。多集即集合中允许出现多个相同的元素。一个重写规则由一对多集组成:头部和体部。当规则头与目标多集合中的一个子多集匹配时,则规则被激活,将匹配的子集从目标多集中删去,再将规则体中的多集写回。Shada 程序由类的集合组成,每个类含有方法,即协调规则。一个类可以通过继承的方法扩展,即加入新的规则。对象是含有状态类的实例,当协调规则被激活或对象接收到消息时,对象的状态发生变化。对象实际上是一个含有重写规则的元组空间,规则头用于触发规则及移出匹配的子多集,规则体用于执行动作,可执行的动作包括:

1)向对象的元组空间写入元素;

2)通过发送消息向其它对象的元组空间写入元素;

3)创建新的对象。

Shada 是对宿主语言的语法扩展,因此需要预编译。同 Jada 一样,Shada 没有实现协调在空间上的分离。

4.7 ACLT

为了增强 Linda 的表达能力,ACLT[20]在协调介质中引入了反应规则,这种被扩展的协调模型一般称为可编程介质(Programmable Coordination Media)。ACLT 最初是为了解决多组件软件系统的协调问题,后来又用于多代理系统的协调。它在 Linda 模型的基础上增加了逻辑元组空间(Logic Tuple Space)、反应式元组空间(Reactive Tuple Space)的概念。逻辑元组空间具有两面性,一方面象普通元组空间一样是一种协调设施,另一方面又是一个知识仓库(Knowledge Repository)。反应式元组空间是可编程介质思想的核心,具有对通信事件反应的能力,而标准的 Linda 只是改变通信状态。

ACLT 将事件分为物理事件和逻辑事件两种,物理事件即传统的 Linda 原语操作,逻辑事件是内部定义的。可以通过 map(Operation, Event)将一个物理事件与一个逻辑事件映射起来。对物理事件的反应式形式是 reaction(Operation, Body),对逻辑事件的反应式形式 react(Event, Body),Body 即反应体,是若干反应目标(Reaction Goals)的合取。合取意味着整个反应体是原子性的,只要其中一个反应目标失败,整个反应体失败。反应目标可以是:状态原语、谓语项、通信原语。

ACLT 认为 in 和 rd 包括两个阶段:前阶段(pre phase),提交元组模板;后阶段(post phase),匹配中的元组返回给查询者。ACLT 不允许反应式嵌套,即执行一个反应式前,上一个反应式必须完成。

ACLT 的贡献在于:1)表达能力比 Linda 有了显著提高;2)实现了协调策略与计算代码的分离,这样协调策略的演化不用修改计算代码。

4.8 MARS

MARS (Mobile Agent Reactive Spaces)[21]是一种移动代理系统的协调体系结构,同 ACLT 一样,基于可编程介质思想。MARS 是在 JavaSpaces 的基础上开发的,增加了元级元组空间(Meta-level Tuple Space)的概念。一个元级元组可表示为(Rct, T, Op, I),含义是:若标识为 I 的代理对一个与 T 匹配的元组进行 Op 操作,则对象 Rct 的 reaction 方法被激活。

MARS 的反应模型基于元级元组,而不是象 ACLT 那样提供一个规范语言,其优点是不用引入新的语言,缺点是协调代码过于分散。

5 研究方向展望

控制驱动协调技术的特征是对计算体的行为在空间和时间上做很大的约束,适合于并行环境及相对封闭的分布式环境中的协调,例如同一组织内的工作流管理系统、电子会议等。

数据驱动协调技术的特点是把计算体行为间的依赖性映射到数据间的依赖性上，这些数据通常存放在一个共享空间中，而且有基于内容的匹配查询机制，从而实现了计算在空间和时间上的分离，更加适合开放式分布环境的协调。

网络计算是信息技术领域中的重要发展方向，网络计算环境[22]是分布式环境的一个特例，具有高度的开放性。随着网络计算的应用领域（如虚拟企业、电子商务、移动计算等）的不断扩展，需要有强大的协调技术作为支撑，因此数据驱动协调技术有更大的发展前景。目前数据驱动协调技术可以做的进一步研究工作包括：

1）协调语言

早期的数据驱动协调语言（如 Linda，Jada，Shada，T-Spaces，JavaSpaces）仅表现为基于协调介质的通信原语，协调代码和计算代码混合在一起，使协调策略难以演化和重用，也不符合安全性要求。可编程介质（如 ACLT，MARS）的提出，增强了数据驱动协调语言的表达能力，更重要的是使协调策略从计算代码中分离了出来。但是协调语言仍然不完善，不能清晰地表达计算行为间的依赖性，难以理解和维护，不适合描述复杂的协调问题。

2）形式化模型

形式化模型是对问题的数学描述，是对设计进行正确性验证的基础。目前的协调技术特别是数据驱动协调技术缺乏形式化模型。

3）安全机制

目前的数据驱动协调技术中的协调介质是一个开放的空间，计算体在上面的操作没有约束，如何保证有依赖关系的计算体间的交互合作不受干扰，是将数据驱动协调技术应用于网络计算环境的重要保证。目前的安全技术研究和协调技术是两个分离的研究领域，如何将两者结合起来是一项很有价值的研究工作。

结束语 协调技术是近几年来发展起来的新兴研究领域，随着网络计算技术的发展，愈加显示出其重要性。本文介绍了协调技术的发展状况，分析了一些有代表性的协调语言、模型，最后阐述了协调技术的发展趋势和待解决的问题。

参 考 文 献

1 Arbab F. What Do You Mean, Coordination? March'98 Issue of the Bulletin of the Dutch Association for Theoretical Computer Science (NVTI), Amsterdam, The Netherland, 1998. http://www.cwi.nl/NVTI/Nieuwsbrief/Nieuwsbrief98.ps.gz

2 Tolksdorf R. Coordinative Application, Structured Coordination, and Meta Coordination. In: Proc. of the 30th Hawaiian Intl. Conf. of System Sciences, Hawaii, U.S.A., 1997

3 Ahuja S, Carriero N, et al. Linda and Friends. IEEE Computer, 1986, 19(8): 26～34

4 Carriero N, Gelernter D. Coordination Languages and Their Significance. Communications of the ACM, 1992, 35(2): 97～107

5 Arbab F. Manifold version 2: Language Reference Manual: [Technical Report]. Amsterdam: Cetrum Voor Wiskunde en Informatica, 1996. http://www.cwi.nl/ftp/manifold/refman.ps.Z

6 Bouvry P, Arbab F. Visiflod: A Visual Environment for a Coordination Language. First International Conference on Coordination Languages and Models, Cesena, Italy, 1996

7 Bonsangue M, Arbab F, et al. A Transition System Semantics for a Control-driven Coordination Language: [Technical Report]. Amsterdam: Centrum voor Wiskunde en Informatica, 1998. http://www.cwi.nl/ftp/CWIreports/SEN/SEN-R9829.ps.Z

8 Barbacci M R, Weinstock C B, et al. Durra: A Structure Description Language for Developing Distributed Applications. IEE Software Engineering Journal, 1993, 8(2): 83～94

9 Minsky N H, Ungureanu V. Law-Governed Interaction: a Coordination & Control Mechanism for Heterogeneous Distributed Systems. ACM Transactions on Software Engineering and Methodology, 2000, 9(3): 273～305

10 Li D, Muntz R R. COCA: Collaborative Objects Coordination Architecture. In: Proc. of ACM CSCW'98, Seattle, U.S.A, 1998

11 Cruz J C, Ducasse S. A Group Based Approach for Coordinating Active Objects. Third Intl. Conf. on Coordination Languages and Models, Amsterdam, The Netherlands, 1999

12 Roman G C, Cunningham H C. Mixed Programming Metaphors in Shared Dataspace Model of Concurrency. IEEE Transaction on Software Engineering, 1990, 16(12): 1361～1373

13 Kielmann T. Designing a Coordination Model for Open Systems. First Intl. Conf. on Coordination Languages and Models, Cesena, Italy, 1996

14 Wyckoff P, Mclaughry S, et al. T-Spaces. IBM Systems Journal, 1998, 37(3): 454～474

15 Waldo J, et al. JavaSpaces Sepcification 1.0 Revision: [Technique Reports]. Palo Alto: Sun Microsystems, 1999

16 Rowstron A. WCL: A Co-ordination Langauge for Geographically Distributed Agents. World Wide Web, 1998, 1(3): 167～179

17 Ciancarini P, Rossi D. Jada: Coordination and Communication for Java Agents. In: Vitek J, Tschudin C. Mobile Object Systems: Towards the Programmable Internet. Berlin: Springer-Verlag, 1997. 213～228

18 Ciancarini P, Rossi D. Coordinating Distributed Applets with Shada/Java. ACM Applied Computing Review, 1998, 6(1): 1～12

19 Banatre J, LeMetayer D. Programming by Multiset Transformation. Communication of the ACM, 1993, 36(1): 98～111

20 Denti E, Natali A, et al. On the Expressive Power of a Language for Programming Coordination Media. In: Proc. of the 1998 ACM Symposium on Applied Computing, Atlanta, U.S.A., 1998

21 Cabri G, Leonardi L, et al. MARS: A Programmable Coordination Architecture for Mobile Agents. IEEE Internet Computing, 2000, 4(4): 26～35

22 Singh M P, Vouk M A. Network Computing. In: Webster J G. Wiley Encyclopedia of Electrical and Electronics Engineering, volume 12. New York: John Wiley & Sons, 1999. 114～132