DTSArch:一种基于分散式元组空间的软件体系结构模型

郑 翔 覃 征 邢剑宽
(清华大学计算机科学与技术系 北京 100084)

摘 要 针对协作系统中缺乏描述系统高度动态性、自组织性和协作性的方法,利用分散式元组空间模型的时空解耦特性,提出了一种软件体系结构模型DTSArch。在对分散式元组空间进行刻画的基础之上,从软件体系结构的角度建立了描述构件行为和系统配置的形式化语义,解决了分散式元组空间模型难以直接应用于实际系统开发的问题。实现了一个基于DTSArch的可视化开发工具以进行软件体系结构设计。并通过实例证明DTSArch使开发人员能够快速建立基于分散式元组空间模型的系统结构,从而提高系统构件和系统开发方法的可重用性。

关键词 分散式元组空间,软件体系结构建模,协作系统
中图法分类号 TP311.1 **文献标识码** A

DTSArch: A Software Architecture Model Based on Decentralized Tuple Space

ZHENG Xiang QIN Zheng XING Jian-kuan
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract The collaborative systems lack descriptional support for dynamism, self-organization and collaboration. A new software architecture model named DTSArch was proposed based on decoupling of decentralized tuple space model. DTSArch provided formal semantics to describe component behavior and system configuration, dealing with the difficulty in applying decentralized tuple space model into system development. A visual develop tool based on DTSArch was implemented. The practice shows that developers can quickly build system architecture on decentralized tuple space model, which improves the reusability of components and development approaches.

Keywords Decentralized tuple space, Software architecture modeling, Collaborative system

文献[1]中首先提出元组空间(Tuple Space)模型,又称为Linda模型。它将并行系统中多个进程的复杂的数据访问、通讯和协同抽象为在一个元组空间上进行元组(Tuple)的写(out)、读(rd)和删除(in)。元组空间模型最大的特点是时间和空间的解耦性:时间上的解耦是指通讯的双方不需要同时存在就可以通讯;空间上的解耦是指通讯与通讯参与者所处的逻辑位置或者物理位置无关。典型的商用元组空间系统包括JavaSpaces[2],IBM TSpaces[3]和Giga Spaces[4]等。它们的共同特点是集中式的(Centralized),即物理上存在集中的服务器来负责元组空间的工作。

但诸多领域的迅速发展,如计算机支持的协同工作(CSCW)、群决策系统、移动自组织网应用系统等,迫切需要对高动态性、自组织性和协作性的支持。目前大多数的研究是从网络传输、应用体系结构与中间件、协同应用构建等基础层面上展开的。本文认为分散式元组空间模型是有效解决以上问题的途径。所谓分散式元组空间(Decentralized Tuple Space)是指这样一种元组空间:物理上不存在实际的集中元组空间服务器,所有的空间参与者共同维护元组空间这个逻辑抽象。目前已存在一些分散式元组空间原型系统,如Lime[5],TOTA[6],PeerSpaces[7]等。它们有的给出了自身系统模型,有的提出了负载平衡的算法。但是它们均没有给出针对这一类系统的软件体系结构模型,从而使得这一特定领域系统构件和开发方法难以重用。目前也不存在专注于这个主题的软件体系结构研究。

本文的目标是为基于分散式元组空间的软件系统建立一个软件体系结构模型DTSArch,一方面可以利用元组空间模型的优势特性来支持系统的高动态性、自组织性和协作性;另一方面,将抽象的元组空间模型转变成软件开发人员更容易理解的软件体系结构元素,如构件、连接器、配制、演化规则等,以改善本领域软件系统开发的可重用性。在本文中采用高阶π演算作为软件体系结构的建模基础,以支持构件类型化和构件行为描述。本文基于DTSArch实现了一个可视化软件体系结构开发工具,用于支持系统设计和系统框架代码自动生成。

1 分散式元组空间模型

1.1 元组与元组模板

在元组空间模型中,数据类型均以元组的形式存在。如

到稿日期:2008-11-11 返修日期:2009-02-03 本文受总装装备预研基金(9140a15030308jw0108)资助。
郑 翔(1984-),男,硕士生,主要研究方向为软件体系结构等,E-mail:xiangmyself@gmail.com;覃 征(1956-),男,教授,主要研究方向为软件体系结构等;邢剑宽(1983-),男,博士生,主要研究方向为软件体系结构等。

果以 d 表示数据域，则元组 e 和元组模板 t 可以表示为：

$$e=(\vec{d}) \qquad t=(\vec{fd})$$
$$\vec{d}::=d|d;\vec{d} \qquad \vec{fd}::=f|\varphi|f;\vec{fd}$$
$$f::=null|d$$

其中，上箭头表示一个序列。$\vec{d}$ 表示元组数据域序列。$\vec{fd}$ 表示模板中的数据域序列。本文使用 $e[i]$ 和 $t[i]$ 表示元组或者元组模板中的第 i 个域（i 从 1 开始计数）。$null$ 表示通配符匹配；φ 表示进行元组扩展匹配。其定义如下。

定义 1（元组扩展匹配） 对于一个元组 $e=\langle d_1,\cdots,d_m\rangle$ 和一个元组模板 $t=\langle fd_1,\cdots,fd_m\rangle$，如果 t 匹配 e（表示为 $e\triangleright t$）当且仅当以下任何一条成立：

1）$m=n\wedge(d_i=fd_i\vee fd_i=null,1\leqslant i\leqslant m)$；

2）$m>n\wedge(d_i=fd_i\vee fd_i=null,1\leqslant i\leqslant n-1)\wedge fd_{n-1}=\varphi$。

此定义对经典的元组匹配定义进行了扩展。扩展匹配允许一个元组模板只指定一个待匹配元组的前缀而不必指定所有域。定义 6 中使用这个特性实现了元组类型的匹配。

1.2 分散式元组空间

分散式元组空间可以由分布在元组空间中的一组元组和访问元组空间的进程组成。分散式元组空间 DTS 可以如下定义。

定义 2（分散式元组空间）

$DTS=\{T,X,L,\gamma\}$

$T=\{e_1,\cdots,e_k\}$

$X=\{X_1,\cdots,X_m\}$

$L=\{l_1,\cdots,l_m\}$

$\gamma\in P((T\cup X)\times L)$

其中，T 表示系统中所有元组的集合；L 表示系统中所有位置的集合；X 表示访问进程集；γ 表示元组/进程到位置的映射集。

定义 3（分散式元组空间进程）

$X::=\text{nil}|X_1|X_2|\pi.X$

$\pi::=\text{out}(l,e)|\text{rd}(l,t)|\text{in}(l,t)|\text{update}(l,t,e)\quad l\in L$

在分散式元组空间中提供 4 个基本操作 out，rd，in 和 update，分别表示写入、读取、删除和更新。L 表示操作的目标位置。

以上的模型难以直接应用于基于分散式元组空间的应用开发。本文建立一个软件体系结构模型 DTSArch 来将抽象的元组空间模型和软件设计模型联系起来。DTSArch 是 DTS 的精化。

2 DTSArch 模型

在软件体系结构模型中，首先定义类型的概念。在 DTSArch 中，元组空间和元组以及元组使用者均被看作是不同构件类型的实例。

$Type::=V|T|B$

其中，V 代表基本数值类型，如整数、字符串、浮点数等；T 表示一个元组类型；B 是行为类型，如定义 3 所示。在类型的基础上定义元组类型，元组类型是一个类型序列。元组类型之间存在继承关系。

定义 4（元组类型）

$T::=Type|Type;\overrightarrow{Type}$

定义 5（元组类型继承） 对于两个元组类型 T_i 和 T_j，如果存在一个类型序列 $\vec{U}\in M(\overrightarrow{Type})$，满足

$T_i=T_j;\vec{U}$

则 T_i 继承于 T_j，记为 $T_i\Rightarrow T_j$。从定义 5 可以推断，如果 $T_i\Rightarrow T_j$，$T_j\Rightarrow T_k$，那么 $T_i\Rightarrow T_k$。

可以在元组空间的元语操作中使用下面的式子表示向元组空间中写入一个类型为 T 的元组 e。其余的操作表示方式与其类似。

$\text{out}(l_d,e:T)$

为了在元组中维护其类型信息，需要将类型放在此元组中。在 DTSArch 中，如果 $\exists e:T$，则 $e[1]=T$，即 $e=\langle T,d_1,\cdots,d_m\rangle$。这样，就可以定义元组类型匹配模板的含义。

定义 6（元组类型匹配模板） 一个元组类型 T 的任何元组可以被其元组匹配模板 $\hat{t}_T$ 表示，$\hat{t}_T=\langle T,\varphi\rangle$。匹配规则见定义 1。

基于类型和元组类型的概念，可以定义基于分散式元组空间的构件。

定义 7（构件） 一个构件（*Component*）可以形式化地描述为

$Component=\{Tuples,Ports,Behavs,Prot\}$

$Tuples=\{e_1:T_1,\cdots,e_n:T_m\}$

$Ports=\{port_1,\cdots,port_k\}$

$Behavs=\{B_1,\cdots,B_r\}$

其中，$Tuples$ 表示此构件中包含的所有元组的集合；$Ports$ 是端口名，不同构件之间通过端口名通讯，基于端口的通讯只发生在同一个位置；$Behavs$ 表示所有构件行为的进程规约，定义了多个行为类型；$Prot$ 是端口上的通讯协议，用 π 演算[8]来描述，将在接下来的定义中说明。

在 DTSArch 中，最重要的构件就是元组空间节点（Tuple Space Node，TSN），它负责维护分散式元组空间一个物理主机上的元组空间分片。这个分片中所有元组组成的集合称为本地元组空间（Local Tuple Space，LTS）。一个 TSN 代表一个位置和在这个位置上管理元组的进程的综合体，即 TSN 本身就代表一个位置，即 $L=\{TSN_i\}$。其定义如下。

定义 8（元组空间节点）

$TSN=\{LTS,Ports_{TSN},Behav_{TSN},Prot_{TSN}\}$

$LTS=\{e_1:T_1,\cdots,e_n:T_m\}$

$Ports_{TSN}=\{in\}$

$Behav_{TSN}=\phi$

$Prot_{TSN}::=\text{via } in \text{ rec}(x:B).x.Prot_{TSN}$

可以看到，TSN 利用其定义的端口 *in* 来不断接收一个动作，并且执行该动作。上式中动作由变量 x 表示。这个动作被限制为元组空间允许的操作，如定义 3 所示。

对元组空间的操作可以定义为对某个具体的 LTS 集合的增减。以下用 $\overline{TSN}$ 表示属于此 TSN 的 LTS。$LTS\oplus e$ 表示本地元组空间中增加了一个元组 e；$LTS\odot e$ 表示移除一个元组 e。对元组空间的操作可以由对 LTS 进行集合操作来定义。

定义 9（分散式元组空间系统操作语义）

(1) $\text{out}(TSN,e)\Rightarrow\overline{TSN}\oplus e$

(2) $\text{rd}(TSN,t)\Rightarrow TSN,if\ \exists e\in\overline{TSN};e\triangleright t$

(3) $\text{in}(TSN,t)\Rightarrow\overline{TSN}\odot e,f\exists e\in\overline{TSN};e\triangleright t$

(4) update(TSN,t,e)⇒in(TSN,t). out(TSN,e)

可见,在 DTSArch 中,元组的分布由操作发起者确定,而不像结构化 P2P 系统中由系统内部确定,对外界透明。

TSN 将系统分为明确的两层:元组空间层和应用层,如图 1 所示。元组空间层由多个 TSN 组成,它们之间的通讯方式是网络传输元组。元组空间层和应用层之间使用端口进行通讯。应用层构件被称为代理(Agent)。应用层构件预先定义一组行为,然后通过端口交由 TSN 执行。一个简单的 Agent 的描述可以表示为

$Ports$={in}

$Behav$={$Demo$}

$Demo$::=out(TSN_i,("Hello")). nil

$Prot$::=vin in send($self$,$demo$:$Demo$). nil

它向本地的 TSN 申请发送包含字符串"Hello"的元组到 TSN_i。$self$ 表示一个 Agent 隶属的 TSN,由 DTSArch 的 $Conf$ 来确定,见定义 10。

这样,整个系统的体系结构可以做以下定义。

定义 10(DTSArch)

$DTSArch$={$M(T)$,$M(TSN)$,$M(Agent)$,$Conf$}

其中,$M(T)$,$M(TSN)$,$M(Agent)$分别表示元组类型集合、TSN 集合和 Agent 集合。$Conf$ 描述系统配置,$Conf \in P(M(TSN) \times M(Agent))$。

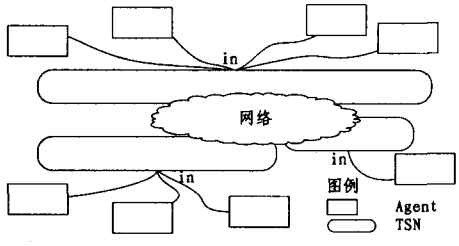

图 1 分散式元组空间系统模型图

至此,通过采用 DTSArch 对系统进行描述,就可以将软件系统结构同分散式元组空间模型结合起来。

3 实现和应用

3.1 实现

DTSArch 采用 EMF(Eclipse Modeling Framework)[9] 来实现模型。模型内容采用 XML 表示。然而对于比较复杂的系统设计,手工编写软件体系结构描述工作量大,缺乏直观,并且容易出错。DTSArch 利用 GMF(Graphical Editor Framework)[10] 构建了 DTSArch 可视化开发工具,如图 2 所示。利用此开发工具,可以对系统结构和行为进行完整的描述。代码生成工具能自动将体系结构设计描述转变为可编译和运行的代码。这些框架代码实现均通过了测试,此设计流程大大加快了系统开发的效率和正确性。

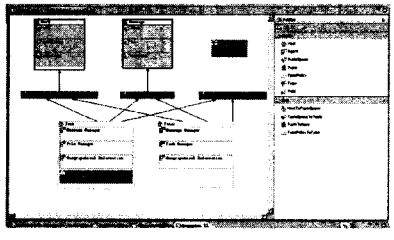

图 2 DTSArch 可视化编辑工具

3.2 应用实例

本节以地图协作标定为例介绍 DTSArch 的应用。假设存在两个协作者 u_1 和 u_2,各自借助移动设备在共享的地图上标定一些特殊位置,如危险位置、桥梁等。两个人可以共享地图标注。

首先定义一个 Agent:GeoManager 以及元组类型 Mark,DangerMark 和 BridgeMark(实际系统中有更多标注类型,这里仅举两例)。其定义如表 1 所列。

表 1 元组类型定义

元组类型	域类型	域含义
Mark	string	标定人
	double	经度
	double	纬度
DangerMark	int	危险级别
	string	危险描述
BridgeMark	double	载重量
	double	长度
	double	宽度

其中,DangerMark⇒Mark,BridgeMark⇒Mark。

在此系统中存在两个 TSN,分别命名为 TSN_1 和 TSN_2。TSN 的描述均是相同的,如定义 8 所示,不再赘述。TSN 是基本的可复用构件。

下面定义 GeoManager 构件语义。

$GeoManager$={$Marks$,$Ports$,$Behavs$,$Prot$}

$Marks=\phi$

$Ports$={in}

$Behavs$={$Display$,$MarkDanger$,$MarkBridge$}

$Display$::=in($M(TSN)-self$,$\hat{t}Mark$).

($Marks \oplus x$). $sleep$. $Display$

$MarkDanger$::=$mark$(x:$DangerMark$).

out($self$,x). $ready$. $MarkBridge$

$Prot$::=via in send(md:$MarkDanger$|

mb:$MarkBridge$|dis:$Display$). nil

在这个描述中,同时完成了读取其他协作者标定、标定危险地点、标定桥梁的工作,分别由 $Display$,$MarkDanger$ 和 $MarkBridge$ 3 个进程来完成。通讯协议将 3 个进程利用并发关系连接起来,并交由本地 TSN 执行。其中,$Display$ 将利用 in 操作收到的标定元组加入到 Marks 集合中用以显示。in 的参数采用了 Mark 的类型的匹配模板,所以可以同时接受各种具体 Mark 类型的元组。

如果要增加一个标定协作者自身位置的功能,只需要增加相应的元组类型 SelfMark 和进程 MarkSelf。

$MarkSelf$::=$locate$(x:$SelfMark$).

$update$($self$,$tSelfMark$,x). $sleep$. $MarkSelf$

为了使 update 正常工作,必须在初始化阶段在 Marks 集合中添加初始自身位置元组。图 3 是此地图标定系统的实现效果。

从这个实例中可以看到 DTSArch 带来的优势:

1)协作双方的连接一旦中断,会使得 in 操作的执行被阻塞,直到连接恢复。in 被阻塞的同时,其他的进程可以正常执行。

2)系统易于理解。一个 Agent 的开发者只需要了解元组空间支持的 4 种基本操作即可。关于分散式节点底层通讯细

节可以不做任何考虑。

3)构件复用性得到了增强。因为仅仅利用元组空间进行通讯,Agent 不存在常规构件那样复杂的依赖关系。依赖限制的减少增加了 Agent 的可复用场景。

4)基于分散式元组空间的协作系统开发效率得到提高。基于开发的软件体系结构图形化开发工具,可以快速将设计转换为可运行代码。

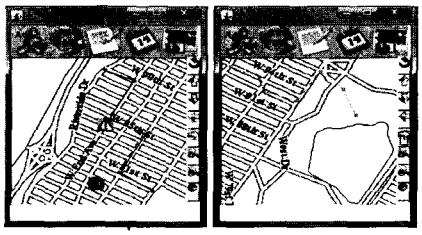

图 3 基于分散式元组空间的协作标定系统

结束语 为分散式协作系统找到一种方便、快捷、具有形式化语义支持的开发模式,其对于计算机支持的协同工作(CSCW)、群决策系统、移动自组织网应用系统等领域的发展十分关键。元组空间模型由于其固有特性,尤其适合作为此类开发模式的基础。本文提出了一种面向分散式元组空间的软件体系结构模型 DTSArch,用于支持此类系统的设计与实现。较目前已存在的软件体系结构模型,DTSArch 充分利用了分散式元组空间的优势,为分散式协作系统这个领域的开发实践提出了新的思路。

DTSArch 描述了系统的结构和行为。基于 DTSArch 的精确语义,开发人员能够快速建立实际协调系统的软件体系结构模型。本文还采用 EMF 和 GEF 框架对模型进行了实现,并建立了图形化编辑器以及相关的系统代码生成工具。利用它们设计者能够直观地描述或设计系统实例并生成相应的系统代码,对此类系统开发有较大实用价值。该模型和工具已经应用于移动协同平台中。

在后续的工作中,主要就以下两个问题进行进一步研究:1)丰富和扩展 DTSArch 的元语,为常见的协作行为提供更直接的支持;2)从动态体系结构的角度考虑,扩展 DTSArch 的描述能力,使其能够描述软件体系结构运行时的演化。

参 考 文 献

[1] Gelernter D. Generative communication in Linda [J]. ACM Transactions on Programming Languages and Systems, 1985, 7 (1): 80-112

[2] Freeman E, Hupfer S, Arnold K. JavaSpaces(TM) Principles, Patterns, and Practice[M]. Addison-Wesley Pub. Co., 1999

[3] Wyckoff P, et al. T spaces[J]. IBM Systems Journal, 1998, 37 (3): 454-474

[4] GigaSpaces. GigaSpaces enterprise application grid 4.1 documentation[S]. 2005

[5] Murphy A L, et al. LIME: A coordination model and middleware supporting mobility of hosts and agents[J]. ACM Transactions on Software Engineering and Methodology, 2006, 15(3): 279-328

[6] Mamei M, Zambonelli F. Spatial Computing: The TOTA Approach[C]// SELF-STAR 2004. LNCS 3460. Berlin Heidelberg: Springer-Verlag, 2005: 307-324

[7] Busi N, et al. PeerSpaces: data-driven coordination in peer-to-peer networks[C]// Proceedings of the 2003 ACM Symposium on Applied Computing. Melbourne, Florida: ACM Press, 2003: 380-386

[8] Sangiorgi D, Walker D. The π-calculus: a Theory of Mobile Processes[M]. Cambrige University Press, 2001

[9] EMF. Eclipse Modeling-EMF Page[EB/OL]. http://www.eclipse.org/modeling/emf/

[10] GMF. Eclipse Graphical Modeling Framework Page[EB/OL]. http://www.eclipse.org/modeling/gmf

(上接第 131 页)

10%和 30%时修复拓扑所产生的成本。首先从成本趋势来看,两个协议的修复成本都随超级节点失效比例及网络规模的增加而增大,但 HOST 的修复成本增长趋势要慢于 SG-1。其次从数值上看,HOST 修复成本要明显低于 SG-1。例如在网络规模为 8 万、失效比率为 30%时,SG-1 的修复成本为 322730,而在 HOST 中该值为 53230,只占 SG-1 修复成本的 16.49%。差异原因在于:在 SG-1 中,超级节点失效后,其子节点需要在全局范围内通过谣言方式重新寻找其他超级节点加入,导致较高的失效修复成本。而在 HOST 中,通过有序的修复算法,候选超级节点会接管原有失效超级节点的角色,并将超级节点失效影响控制在本簇范围内,使修复成本大幅降低。

结束语 本文给出一种分级有序的 P2P 超级节点拓扑构造(HOST)方案。首先通过分裂和调整机制对拓扑进行控制,使拓扑层次和超级节点负载随网络规模增大进行自适应调整。其次通过有序的节点加入、退出以及失效修复算法,减少了拓扑构造的随意性。模拟结果表明,HOST 在降低拓扑构造成本、保持超级节点负载均衡以及减少失效修复成本 3 个方面均优于 SG-1 超级节点拓扑构造方案。

参 考 文 献

[1] Napster[OL]. http://www.napster.com

[2] The Gnutella Protocol Specification v0.4 [EB/OL]. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[3] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for Internet applications [C]// Cruz R, Varghese G, eds. Proceeding of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications(SigComm). New York: ACM Press, 2001: 149-160

[4] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems[C]// Guerraoui R, ed. Proceedings of the IFIP/ACM Int'l Middleware Conf. London: Springer-Verlag, 2001: 329-350

[5] Fasttrack[OL]. http://en.wikipedia.org/wiki/FastTrack

[6] Garc'es-Erice L, Biersack E W, Felber P A, et al. Hierarchical peer-to-peer systems[C]// Kosch H, Böszörményi L, Hellwagner H, eds. Proceeding of ACM/IFIP Int'l Conference on Parallel and Distributed Computing(Euro-Par 2003). Berlin: Springer-Verlag, 2003: 643-657

[7] 夏启志,谢高岗,闵应骅,等. IS-P2P:一种基于索引的结构化 P2P 网络模型[J]. 计算机学报,2006,29(4):602-610

[8] Gnutella protocol spec. v. 0.6 [EB/OL]. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[9] Liang J, Kumar R, Ross K W. Understanding KaZaA[OL]. 2004. http://cis.poly.edu/~ross/papers/UnderstandingKaZaA.pdf

[10] Montresor A. A Robust Protocol for Building Superpeer Overlay Topologies[C]// Caronni G, Weiler N, Shahmehri N, eds. Proceedings of the 4th International Conference on Peer-to-Peer Computing(P2P'04). Washington: IEEE Press, 2004: 202-209

[11] PeerSim Simulator[EB/OL]. http://peersim.sourceforge.net/