文章编号:1000-6788(2007)07-0077-08

基于 π 网的多 Agent 系统建模与分析

于振华[1,2],蔡远利[1],徐海平[3]

(1. 西安交通大学 电子与信息工程学院,西安 710049;2. 空军工程大学 电讯工程学院,西安 710071;
3. 马萨诸塞州立大学达特茅斯分校 计算机与信息科学系, 北达特茅斯 02747 美国)

摘要: 首先集成两种互为补充的形式化方法-面向对象 Petri 网(Object-Oriented Petri nets,OPN)和 π 演算,建立了一种通用的形式化建模方法——π 网.π 网利用 OPN 形象地描述系统的初始化模型及动态行为,利用 π 演算刻画系统的动态演化.然后以 π 网为语义基础,从软件体系结构的角度,建立了一种多 Agent 系统体系结构模型(Multi-agent Systems Architecture Model, MASAM).在 MASAM 中,将多 Agent 系统抽象为计算 Agent、连接 Agent 和配置等三个单元,并描述了多 Agent 系统的动态演化;研究了系统演化后体系结构一致性的分析方法,从而可以检测系统开发早期存在的错误,确保模型的可靠性和正确性.

关键词: 多 Agent 系统;软件体系结构;面向对象 Petri 网;π 演算;演化

中图分类号: TP18;TP301　　**文献标志码:** A

Modeling and Analyzing Multi-Agent Systems based on π-net

YU Zhen-hua[1,2], CAI Yuan-li[1], XU Hai-ping[3]

(1.School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China;2.School of Telecommunication Engineering,Air Force Engineering University,Xi'an 710071,China;3.Department of Computer and Information Science, University of Massachusetts Dartmouth, North Dartmouth 02747, USA)

Abstract: A general formal method (π-net) is presented, which integrates two complementary formalisms, namely Object-Oriented Petri nets and π-calculus. In π-net, OPN is employed to visually model the system architecture and system behaviors, and π-calculus is employed to describe the system evolution. Based on the π-net, a visual multi-agent systems architecture model (MASAM) is presented from the point of view of software architecture. MASAM divides multi-agent systems (MAS) into computing agents, connecting agents and configuration module, and MAS evolution is described in details; the consistency of MAS architecture is analyzed. Consequently the defects in early design stage can be detected, and the correctness and reliability of MASAM can be ensured.

Key words: multi-agent systems; software architecture; object-oriented petri nets; π-calculus; evolution

1 引言

多 Agent 系统(Multi-agent systems, MAS)是分布式人工智能研究的一个重要分支,已成为计算机及自动化领域的一项关键性主流技术和一种重要的计算范型[1,2],广泛应用于开发大规模、复杂和分布式的工业和商业软件系统,如过程控制、空中交通管制、电子商务等领域.

MAS 是一个自适应的柔性动态系统,可以对系统进行动态配置,对于一些需要动态维护的关键任务系统(例如电信交换系统、电子商务系统、指挥自动化系统等)具有重要意义[3].因为一旦这些系统由于需求或环境变化,为了进行更新或维护而停止运行,将会引起高额的费用和巨大的风险.同时由于系统规模的增大,系统的复杂度以指数速率增长,这都给动态 MAS 的开发带来了一定的难度,不能保证系统的可靠性、可重用性和可扩展性.因此,需要用一种形式化建模方法对 MAS 进行描述、分析和验证,减少在系统开发早期引入的错误,降低开发成本,提高软件质量[4].有研究表明,缺乏形式化规格说明是阻碍 MAS 得到大规模应用的一个主要原因[5].

收稿日期:2006-01-20

资助项目:国家 863 计划(2003AA721070)

作者简介:于振华(1977 -),男,博士生,研究领域为多 Agent 系统形式化建模与分析及应用.

近年来,已提出一些形式化建模方法[5~10]对 MAS 进行建模和开发,但目前国内外对动态 MAS 建模研究较少,文献[5~7]仅局限于刻画静态 MAS,不支持动态、开放系统的开发;以 π 演算为基础的文献[8,10]具有一定的动态体系结构建模能力,Agent 规约和演化也可以对 Agent 演化进行建模,然而它们只是提出了一个描述 MAS 的形式化框架,并没有对模型及 MAS 的演化进行分析和验证,这是确保正确、可靠的 MAS 开发的关键.

本文从软件体系结构的角度,以 π 网为语义基础,建立一种直观的多 Agent 系统体系结构模型(Multi-agent Systems Architecture Model, MASAM).MASAM 可以对 MAS 的静态和动态体系结构进行建模和分析,并分析系统演化过程中 Agent 的演化策略及 MAS 体系结构的一致性.

2 π 网

我们根据软件组件的思想,建立了一种新的面向对象 Petri 网(Object-Oriented Petri nets, OPN)[11],可以提高模型的模块性和柔性,比较适合复杂系统的建模与分析.然而,OPN 的结构是静态的,不适合描述结构动态变化的系统.

π 演算[12]是在 CCS(Calculus of Communicating System)基础上提出的一种并发计算模型,其基本建模单元为名字和进程,进程通过传递名字进行交互,系统由进程的集合构成.通过接受名字,进程可以动态地与其它进程进行交互,并具有了建立新通道的能力,因此 π 演算可以用来描述结构不断变化的并发系统.

OPN 和 π 演算所描述的系统结构及其语义是不相同的,如果单独使用的话,不可能充分利用二者的优点,因此我们把二者结合起来,建立一种集成的形式化方法—π 网.π 网利用 OPN 形象地描述系统的初始化模型和动态行为,当系统发生演化时,利用 π 演算对系统建模并分析动态体系结构的一致性,确保系统的正常交互.利用两种互为补充的形式化方法,可以保留各自的优势,并互相弥补不足,这也是目前形式化方法的发展趋势[13].

3 基于 π 网的多 Agent 系统体系结构模型

软件体系结构揭示了软件系统高抽象层次上的特性,是软件系统设计的重要内容,直接关系到软件系统的质量.我们把软件体系结构的概念引入 MAS 中,并以 π 网为语义基础,建立 MAS 的体系结构模型 MASAM.

定义 1 MASAM 是一个三元组,$MASAM = (AComp, AConn, Conf)$,其中 $AComp = (AComp_1, AComp_2, \cdots, AComp_o)$表示计算 Agent 的集合,$AConn = (AConn_1, AConn_2, \cdots, AConn_p)$表示连接 Agent 的集合,$Conf$ 表示 MAS 的配置.

3.1 计算 Agent($AComp$)

计算 Agent 是一个数据单元或一个计算单元,由接口和内部实现组成,负责与用户和环境进行交互.计算 Agent 是一个三元组,$AComp_o = (ID, AS, EA)$,其中,ID 是计算 Agent 的标识符集合;AS 定义了计算 Agent 的接口和内部实现;EA 利用 π 演算描述了计算 Agent 的演化,在第四节中我们将研究 Agent 的演化与分析.

计算 Agent 中的元 AS 以 BDI 模型为基础,是一个五元组 $AS = (O_o, P_r, P_k, P_g, P_p)$,其中 P_r, P_k, P_g, P_p 为抽象库所,用椭圆表示,分别表示推理模块、知识库模块、目标模块和规划模块.推理模块表示 Agent 可以对获取的外界信息进行推理,并把推理结果存储在知识库 P_k 中.知识库模块对应于 BDI 模型中的信念(Bliefs),主要描述了环境和其它 Agent 的信息,在具体的实现中可以表示成一些简单的变量、数据结构或数据库;目标模块对应于 BDI 模型中的愿望(Desires),主要描述了 Agent 的动机和最终目标,可以表示为一个变量、数据结构或表达式;规划模块对应于 BDI 模型中的意图(Intentions),主要描述了 Agent 获取目标时的行为[14].O_o 中的元 IT 和 OT 描述了计算 Agent 的输入和输出接口,$AComp_o.Interface = \{(t_1, t_2) \mid t_1 \in IT, t_2 \in OT\}$,接口描述计算 Agent 需要的服务和提供给外界的服务,计算 Agent 的实现部分由 O_o 的其它元进行描述.

计算 Agent 的模板如图 1 所示，其中私有应用提供一些私有方法，如注册信息等；内部实现和接口可以根据系统需求进行定制，推理、知识库、目标和规划库所用带阴影的椭圆表示，可以根据需要进行求精. Agent 之间通过通讯语言进行通信，遵循言语行为理论并用复杂的协议进行协商[7]，如 ACL 和 KQML 等.

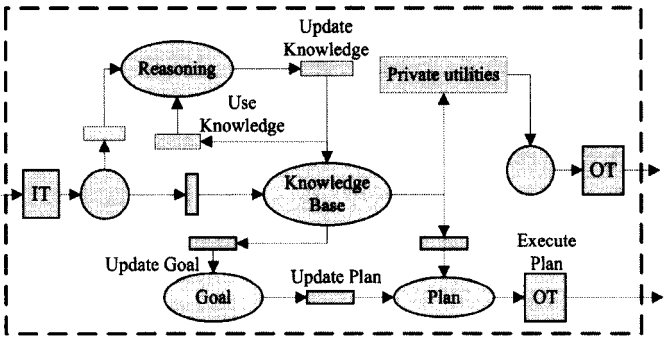

图 1　计算 Agent 模板

3.2　连接 Agent(*AConn*)

连接 Agent 定义了 Agent 之间交互的规则并且给出了一些实现的机制. 连接 Agent 是一个六元组，$AConn_p = (ILP, Channel, KBP, Role, EC, C)$，其中 *ILP*(Intelligent Link Place)为系统中的智能连接库所，用椭圆表示，负责建立 Agent 之间的消息传递通道，然后 Agent 通过 KQML 或 ACL 等进行交互；*Channel* 为 OPN 中的元，表示 Agent 间的消息传递通道；*KBP*(Knowledge-base Place)为知识库所，存储了连接 Agent 感知外部环境的信息和各个计算 Agent 提供的服务信息(如名字、地址和接口信息等)；*Role* 是连接 Agent 中的角色，为与连接 Agent 相交互的、在连接 Agent 中处于相同地位的所有计算 Agent 集合，$Role = \{C_{ID1}, \cdots, C_{IDi}\}$，$C_{IDi}$ 为系统中计算 Agent 的标志符；*EC* 利用 π 演算描述了连接 Agent 的演化，主要描述连接 Agent 建立 Agent 之间交互通道的动态过程；*C*(*ILP*)和 *C*(*KBP*)为与库所 *ILP* 和 *KBP* 相关联的颜色集.

在连接 Agent 中，角色有静态和动态两种类型. 静态角色在 MAS 建立时就已确定，但随着 MAS 的演化，系统中的 Agent 可能会随着环境变化动态地增加或删除，角色因而也就发生动态的变化. 连接 Agent 的元 *EC* 负责建立 Agent 之间的交互通道，描述如下.

当新 Agent 加入时，首先在连接 Agent 中注册，然后通过 *ILP* 建立和其它 Agent 的交互通道. Agent 创建进程为：

$$NewAComp(id, s) = Create(id, s)$$

表示创建了一个标志符为 *id*，服务为 *s* 的 Agent. Agent 注册进程可以描述为：

$$RegInfo(id, s) = (id, s)(\overline{register}\langle id, s\rangle)$$

上式表示 Agent 通过通道 *register* 传递其标志符 *id* 和服务 *s* 到连接 Agent 中. 在连接 Agent 中相应的注册进程为：

$$CRegInfo(x, y) = (vx, y)(register(x, y))$$

表示连接 Agent 通过通道 *register* 获取新 Agent 的注册信息 *x* 和 *y*，同时更新其知识库 *KBP*.

当新 Agent 请求一个服务的时候，由于事先不知道哪一个 Agent 能提供所需的服务，需要通过连接 Agent 查找提供相应服务的 Agent. 如果存在相应的服务 Agent，连接 Agent 负责建立这两个 Agent 之间的交互通道；如果不存在相应的服务 Agent，请求 Agent 可以向连接 Agent 定购这个服务，一旦有提供这类服务的 Agent 注册，连接 Agent 就向请求 Agent 发送消息，通知可以提出相应的服务请求[10]. 新 Agent 请求服务的进程为：

$$RequestService(i, r, l) = \bar{i}\langle a\rangle.r(z).([z = \text{nil}]\,\overline{subscribe}\langle a\rangle + \bar{z}\langle l\rangle)$$

表示新 Agent 通过通道 *i* 发送请求 *a* 到连接 Agent 查询相应的服务 Agent，然后通过通道 *r* 等待连接 Agent 的答复. 一旦新 Agent 收到了服务 Agent 的标志符 *z*，就通过 *z* 发送请求服务的地址 *l* 到服务 Agent.

在连接 Agent 中相应的服务查询进程为：

$$QueryService(i, r, p) = i(y).(\overline{kb}\langle y\rangle \mid Belief(y)).(\bar{r}\langle \text{nil}\rangle.subscribe(y) + \bar{r}\langle p\rangle)$$

表示连接 Agent 通过通道 i 接收请求 Agent 的请求 y，然后通过知识库判断是否存在相应的服务 a，如果存在，通过通道 r 发送该 Agent 的标志符 p，否则，定购该服务.

在服务 Agent 中相应的服务提供进程为：

$$PovdService(p,s) = p(x).\bar{x}\langle s\rangle$$

表示服务 Agent 通过通道 p 接收服务请求的地址 x，并通过 x 发送相应的服务 s 到请求 Agent.

根据上述的 π 演算进程可以建立请求 Agent 和服务 Agent 的交互通道.

在实际的实现过程中，连接 Agent 采用图形界面，主要显示系统中的各个 Agent 及其功能和状态等，可以用来监控 MAS 的体系结构，在系统的设计过程中，是一个重要的调试工具.

在 MASAM 中，连接 Agent 和计算 Agent 从微观层次上描述了 Agent 的结构、方法和接口信息.

3.3 MAS 配置(*Conf*)

Conf 为体系结构的配置，主要描述了由计算 Agent 和连接 Agent 构成的 MAS 拓扑，可以促进软件生命周期中不同用户对 MAS 的理解.

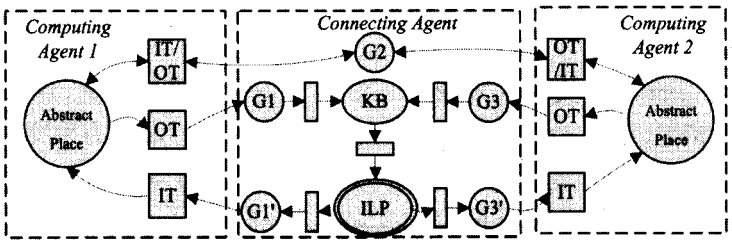

图 2 MAS 体系结构配置

借助于 π 网对动态 MAS 的描述能力，我们能够即时地描述计算 Agent 和连接 Agent 不断更改的配置，从而能清晰地理解系统的演化过程.基于 π 网的 MAS 体系结构模型配置如图 2 所示，主要从宏观层次上描述 MAS，侧重于系统的整体行为和 Agent 之间的交互，也描述了 MAS 模型的静态语义.图 2 只是一个简单抽象的 MASAM，Agent 用 *IT*、*OT* 和抽象库所表示，抽象库所可以进一步求精表示 Agent 的具体实现，计算 Agent 1 和 2 通过通道 *G*1、*G*1′、*G*3 和 *G*3′ 与连接 Agent 交互，计算 Agent 1 通过通道 *G*2 与计算 Agent 2 交互.

MASAM 的动态语义可以通过变迁的使能和发射规则来描述，变迁的发射，使 Token 从一个库所分配到另外一个库所，表明了资源或消息的传递，形象地刻画了 Agent 之间的交互过程.

4 多 Agent 系统动态演化及分析

利用形式化方法建模的一个重要目的就是可以利用数学分析方法对模型进行分析和验证，确保系统的安全性和无死锁性等关键属性的正确，使开发人员能根据系统需求开发出稳定、可靠的产品.对静态 MAS 模型的分析，可以利用 Petri 网的相关分析方法和工具对模型的死锁、有界性和可达性等进行分析；本文的重点是分析动态 MAS，主要研究 Agent 的演化策略和演化过程中系统的一致性，保证演化后的系统能正常交互.

4.1 MAS 体系结构一致性分析

在系统运行时，由于新 Agent 的加入或 Agent 的更新，导致 MAS 体系结构发生演化，演化必须保证系统体系结构的一致性(Consistency)[16].一致性是指 MAS 中各个 Agent 必须成功的进行交互，而不会彼此冲突.一致性对于动态 MAS 尤为重要，因为一旦经过演化后的 Agent 和连接 Agent 彼此发生冲突，那么由此演化出来的系统就一定存在问题，不能正常工作.

Agent 由接口和内部实现组成.内部实现描述接口规定的功能，接口代表了 Agent 提供或需要的服务，表示从一个特定的逻辑交互角度观察时 Agent 能够执行的行为，Agent 之间通过接口进行交互，因而可以在接口层次上判断 Agent 交互的一致性[15].根据软件组件接口利用进程描述的思想[16]，我们把 Agent 接口用 π 演算的进程描述，首先定义进程之间的一致性关系，进而得出 Agent 交互的一致性定理.

如果两个进程 P 和 Q 是一致的，则 P 和 Q 至少能借助一个共享的对偶名字进行交互，如 $P=\bar{x}\langle y\rangle.P'$，$Q=x(z).Q'$，这样的进程之间存在同步关系，因而得出以下的定义.

定义 2(进程半一致性[16]) 设同步进程集合上的二元关系 R，对于 PRQ，且对所有的替代 $\sigma\notin fn(P)\cup fn(Q)$，都满足 $P\sigma RQ\sigma$，而且：

1)如果 $P\xrightarrow{\tau}P'$，则 $P'RQ$；

2)如果 $\neg(P\sim 0)\wedge\neg(P\equiv 0)$，$P\xrightarrow{x(z)}P'$，$Q\xrightarrow{\bar{x}y}Q'$，则 $P'\{y/z\}RQ'$；

3)如果 $\neg(P\sim 0)\wedge\neg(P\equiv 0)$，$P\xrightarrow{x(n)}P'$，$Q\xrightarrow{\bar{x}(n)}Q'$，则 $P'RQ'$.

则称 R 为进程间半一致性关系.如果 R 和 R^{-1} 都是半一致性关系，则称其为进程间一致性关系，记为 $\backsim$.

进程的一致性确保在进程交互中不存在不匹配的情况，强调进程的正常交互，表明进程能够执行内部演化而成功结束.如果存在进程 P'，使得 $P\Rightarrow P'$，且 $P'\xrightarrow{\tau}$ 或 $P'\equiv 0$($P'\xrightarrow{\tau}$ 表示 $\exists P''$，$P'\xrightarrow{\tau}P''$)，则进程 P 可正常运行；如果 $P\Rightarrow P'$，$\neg(P'\xrightarrow{\tau})$ 且 $\neg(P'\equiv 0)$ 就表示进程死锁，进程不能再执行内部演化，其行为也并未完全执行.

定理 1 设 P 和 Q 分别为 $Agent_1$ 和 $Agent_2$ 的接口，且 P 和 Q 为一致性进程，即 $P\backsim Q$，且，则 $Agent_1$ 和 $Agent_2$ 满足一致性，即 $Agent_1\mid Agent_2$ 能正常交互.

证明 Agent 之间满足一致性，即 Agent 能正常交互，不存在死锁.由于 Agent 接口描述 Agent 所提供的动作，因此证明 Agent 满足一致性只需证明如果 $P\backsim Q$，那么 $(\upsilon fn(P)\cup fn(Q))(P\mid Q)$ 可正常交互，即存在一个进程 S(Success)，使得 $(\upsilon fn(P)\cup fn(Q))(P\mid Q)\Rightarrow S$，其中 $S\xrightarrow{\tau}$ 或 $S\equiv 0$.

令 $fn(P)\cup fn(Q)=N$，则 $(\upsilon fn(P)\cup fn(Q))(P\mid Q)=(\upsilon N)(P\mid Q)\Rightarrow S$，根据文献[6]，我们利用数学归纳法证明经过 n 次内部演化后进程仍能正常运行.

1)当 $n=0$ 时，证明 $(\upsilon N)(P\mid Q)\xrightarrow{\tau}$ 或 $(\upsilon N)(P\mid Q)\equiv 0$ 成立.由于 $P\backsim Q$，根据定义 2，有 $\exists\alpha$，$P\xrightarrow{\alpha}$ 和 $Q\xrightarrow{\bar{\alpha}}$，$\alpha$ 和 $\bar{\alpha}$ 为借助一个共享的对偶名字进行交互的动作，为进程的内部演化，因此 $(\upsilon N)(P\mid Q)\xrightarrow{\tau}$ 成立.

2)假设 $n=k$，$k>0$ 时，$\forall P'$，Q'，$P'\Theta Q'$，$(\upsilon N)(P'\mid Q')(\xrightarrow{\tau})^kS$，其中 $S\xrightarrow{\tau}$ 或 $S\equiv 0$ 成立.

3)当 $n=k+1$ 时，要证明 $(\upsilon N)(P\mid Q)\xrightarrow{\tau}(\upsilon N)(P'\mid Q')(\xrightarrow{\tau})^kS$ 成立.根据定义 2，对于第一次内部演化，可以得到：

- $P\xrightarrow{\tau}P'$，由于 $P\backsim Q$，则 $P'\Theta Q$.
- $Q\xrightarrow{\tau}Q'$，由于 $P\backsim Q$，则 $P\Theta Q'$.
- $P\xrightarrow{x(z)}P'$，$Q\xrightarrow{\bar{x}y}Q'$，由于 $P\backsim Q$，则 $P'\{y/z\}\Theta Q'$.
- $P\xrightarrow{x(n)}P'$，$Q\xrightarrow{\bar{x}(n)}Q'$，由于 $P\backsim Q$，则 $P'\Theta Q'$.
- $Q\xrightarrow{x(z)}Q'$，$P\xrightarrow{\bar{x}(y)}P'$，由于 $P\backsim Q$，则 $P'\Theta Q'\{y/z\}$.
- $Q\xrightarrow{x(n)}Q'$，$P\xrightarrow{\bar{x}(n)}P'$，由于 $P\backsim Q$，则 $P'\Theta Q'$.

因此若 $(\upsilon N)(P\mid Q)\xrightarrow{\tau}(\upsilon N)(P'\mid Q')$，则 $P'\Theta Q'$，根据假设得 $S\xrightarrow{\tau}$ 或 $S\equiv 0$ 成立，因此 $(\upsilon fn(P)\cup fn(Q))(P\mid Q)$ 可成功运行，$Agent_1$ 和 $Agent_2$ 满足一致性，得证.

定理 2 如果 MAS 中 Agent 完全满足一致性，则该系统可以正常交互.

由定理 1 可以导出，证明比较简单，这里从略.

例 1 考虑一个电子商务系统中多 Agent 进行价格协商的例子.如果买方 Agent 同意卖方 Agent 提出的价格，则决定购买，否则重新对价格进行协商.假设系统开始时只存在一个卖方 Agent 和一个买方 Agent，

我们把卖方 Agent 抽象为知识库所和接口变迁.初始的 MAS 模型如图 3 所示,买卖 Agent 与连接 Agent 之间通过 $G1$、$G1'$、$G3$、$G3'$ 四个通道交互,表示 Agent 的注册和服务查询过程,买卖 Agent 之间通过 $G2$、$G2'$ 进行交互.

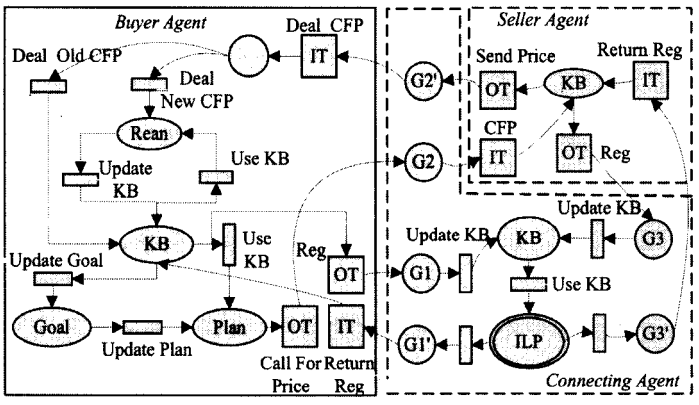

图 3 基于 π 网的初始时 MAS 模型

在系统运行中,买方 Agent 2 加入系统,此时系统中包含 2 个买方 Agent.在 π 网中连接 Agent 为买方 Agent 2 与卖方 Agent 之间建立 $G4$ 和 $G4'$ 两个交互通道.此处我们关注的只是买方 Agent 2 到达后系统的一致性问题,因此我们把新加入 Agent 的接口和与新 Agent 有交互行为的 Agent 接口用进程描述出来,然后根据接口判断新 Agent 的加入是否会影响系统的一致性.买方 Agent 2 与卖方 Agent 的接口定义如下:

买方 Agent 2($BAgent2$):

$$CFP(cfp) = \overline{g4}\langle cfp\rangle.CFP(cfp);$$

$$Dealprice(x) = g4'(x).Dealprice(x);$$

卖方 Agent($SAgent$):

$$DealCFP(y) = g4(y).DealCFP(y);$$

$$Sendprice(price) = \overline{g4'}(price).Sendprice(price);$$

根据定义 2 可得 $CFP(cfp) \backsim DealCFP(y)$, $Dealprice(x) \backsim Sendprice(price)$,而买方 Agent 1 与卖方 Agent 的交互通道与买方 Agent 2 的不相同,不会发生冲突,因此根据定理 1,由这些接口描述的 Agent $BAgent1 \mid BAgent2 \mid SAgent$ 能正常运行,进而根据定理 2,由这些 Agent 组成的系统能保持正常交互.最后可以利用 Petri 网和 π 演算的支持工具分析模型的其它性质,由于已有成熟的工具,本文不详细讨论.

4.2 MAS 动态演化分析

MAS 的动态演化包括计算 Agent 或连接 Agent 的创建或删除、Agent 的更新及负载平衡、重配置等几种情况.下面我们分别进行研究.

对于计算 Agent 或连接 Agent 的创建,首先在当前的连接 Agent 中注册,然后建立与其它 Agent 的交互通道,并判断与其它 Agent 是否一致;如果某个 Agent 达到其目标,需要从系统中删除,则要把它的信息从当前的连接 Agent 中删除,与其它 Agent 的交互通道也要删除.进而根据 4.1 节的一致性分析判断新的系统能否正常运行.

Agent 的加入或删除可能会导致体系结构的动态配置.例如,为了提高系统的稳定性,系统中加入一些备份计算 Agent,这样当主 Agent 发生错误时,可以启用备份 Agent,同时与其它 Agent 的交互也切换到备份 Agent 上.

例 2 如图 3 中的卖方 Agent($SAgent$),在系统运行时卖方加入一个备份 Agent($BakSAgent$)通过通道 bak 备份数据.当卖方 Agent 发生异常需要临时关闭时,需要通知买方 Agent($BAgent$)把链接通道切换到备份 Agent,通道切换过程如下:

$$\overline{bak}\langle g2, g2'\rangle.SAgent \mid bak(x,y).BakSAgent \mid BAgent \xrightarrow{\tau} SAgent \mid BakSAgent\{g2/x, g2'/y\} \mid BAgent$$

表示卖方 Agent 在停止服务前,通过通道 bak 将原来属于自己和买方 Agent 之间的通道传递给备份 Agent,

从而系统的拓扑结构发生改变.

由于系统体系结构发生变化或 Agent 升级,需要对原 Agent 进行更新.Agent 的更新方式分为两种情况,①由于 Agent 内部出现错误或算法效率低,需要对 Agent 内部进行更新,即 Agent 的内部演化;②由于 Agent 在与环境交互过程中获得一些新的知识,对外界提供了一些新的功能,即 Agent 内部实现和接口都发生了演化.对于第一种情况,可以利用 π 演算的弱等价关系判断新旧 Agent 能否进行替换.弱等价关系可以判定两个具有不同内部结构而表现出不同内部行为的系统从外部看来是否等价,可确定替换规则如下:

规则 1　设 $Agent_0$ 具有接口 P,$Agent_1$ 具有接口 Q,若 $P \approx Q$,则接口 Q 可以替换 P,那么 $Agent_1$ 可替换 $Agent_0$.

对于第二种情况,由于 Agent 提供新的功能,因此 Agent 的原有接口发生变化或提供新的接口,在进行 Agent 更新时,可能新旧 Agent 的行为并不完全等价[16],我们主要从接口层次上判别新旧 Agent 能否替换,下面给出 Agent 替换规则.

规则 2　设 $Agent_0$ 具有接口 P,$Agent_1$ 具有接口 Q,对于进程集合上的二元关系 $\$$,当 $P \$ Q$ 时,满足以下条件:

1) $fn(P) \subseteq fn(Q)$;

2) 如果 $P \xrightarrow{x(z)} P'$,那么 $\exists\, Q'$,$Q \xrightarrow{x(z)\cdots x_i(z_i)} Q'$ 且 $P' \$ Q'$($\xrightarrow{x(z)\cdots x_i(z_i)}$ 表示 Q 可响应若干个输入行为);

3) 如果 $P \xrightarrow{\bar{x}\langle y\rangle} P'$,那么 $\exists\, Q'$,$Q \xrightarrow{\bar{x}\langle y\rangle\cdots\bar{x}_i\langle y_i\rangle} Q'$ 且 $P' \$ Q'$($\xrightarrow{\bar{x}\langle y\rangle\cdots\bar{x}_i\langle y_i\rangle}$ 表示 Q 可执行若干个输出行为);则称二元关系 S 为替换关系.如果 $P \$ Q$,则进程 Q 可以替换 P,记为 $P \prec Q$,那么 $Agent_1$ 可以替换 $Agent_0$.

条件 1)说明了 P 的自由名集合为 Q 自由名集合的子集,条件 2)和 3)分别说明了 Q 除了响应和执行 P 的动作,还可以执行其它的动作.因此如果 $Agent_1$ 除了保持对原有接口的更新,同时还提供一个新的接口,那么 $Agent_1$ 也可替换 $Agent_0$.

在分布式应用中,为了提高系统性能而配备新的服务器,需要把客户端的请求平均分配到不同的服务器上,对负载进行均衡,借此缩短系统的响应时间.在 MASAM 中,由于通过连接 Agent 建立计算 Agent 之间的交互通道,可以通过连接 Agent 对服务计算 Agent 的负载进行调整.

例 3　在例 1 的电子商务系统中,假设系统中有两个卖方 Agent,随着时间的变化,系统中买方 Agent 的数量增加,可以通过连接 Agent 调整买方 Agent 的请求使两个卖方 Agent 的负载保持平衡,调整规则如下:

$$\begin{aligned}&\text{if}(SAgent_1.cn \leqslant SAgent_2.cn)\\&\text{then } QueryService(i_1, r_1, p_1)\\&\text{else } QueryService(i_2, r_2, p_2)\end{aligned}$$

这个规则表示连接 Agent 在建立买方和卖方的交互通道时,首先判断两个卖方 Agent 中买方的数量,如果第一个卖方 Agent 顾客的数量少于第二个,则如果有新的买方加入时,通过连接 Agent 的服务查询进程发送第一个卖方 Agent 的标志符给买方,从而建立它们之间的交互通道.反之,建立买方与第二个卖方 Agent 的交互通道.

在 MAS 体系结构演化后,尤其对于 Agent 的创建和更新,由于系统中加入了新的 Agent 而且 Agent 提供了一些新的接口,这些新的功能可能会影响系统的一致性[17],从而发生冲突.因此必须分析 Agent 演化对系统中其它 Agent 的影响,利用 MASAM 可以从体系结构配置中找出可能受影响的其它 Agent 并进行相应的调整,然后根据定理 1 和 2 判断系统能否正常交互.

5　结束语

对于大规模、复杂的软件系统开发,MAS 被视作一种理想的技术.本文以 π 网为语义基础,提出了一种适合于描述静态和动态 MAS 的体系结构模型 MASAM.MASAM 从微观和宏观层次上研究 MAS,在微观层次上,侧重于 Agent 的实现细节;在宏观层次上,主要关注系统的总体设计和 Agent 之间的动态交互.为了确保演化后系统能正常交互,利用 π 演算的相关分析方法分析了 Agent 的演化策略及系统的一致性.

参考文献:

[1] Zambonelli F, Omicini A. Challenges and research directions in agent-oriented software engineering [J]. Autonomous Agents and Multi-Agent Sytems, 2004, 9(3): 253 - 283.

[2] Luck M, Mcburney P, Preist C. A manifesto for agent technology: Towards next generation computing [J]. Autonomous Agents and Multi-Agent Sytems, 2004, 9(3): 203 - 252.

[3] Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages [J]. IEEE Trans on Software Engineering, 2000, 26(1): 70 - 93.

[4] He X, Yu H, Shi T, et al. Formally analyzing software architectural specifications using SAM [J]. Journal of Systems and Software, 2004, 71:11 - 29.

[5] Brazier F M T, Dunin-Keplicz B M, Jennings N R, et al. DESIRE: modelling multi-agent systems in a compositional formal framework [J]. International Journal of Cooperative Information Systems, 1997, 6(1): 67 - 94.

[6] Luck M, d'Inverno M. A formal framework for agency and autonomy [C]//Proc First Int'l Conf Multi-Agent Systems (ICMAS-95), 1995, 254 - 260.

[7] Xu H, Shatz S M. A framework for model-based design of agent-oriented software [J]. IEEE Transactions on Software Engineering, 2003, 29(1): 15 - 30.

[8] Dumond Y, Roche C. Formal specification of a multi-agent system architecture for manufacture: The contribution of the π-calculus [J]. Journal of Materials Processing Technology, 2000, 107: 209 - 215.

[9] 詹剑锋, 程虎. 基于软件体系结构的 Agent 规约和演化 [J]. 计算机研究与发展, 2002, 39(12): 1543 - 1549.
Zhan Jianfeng, Cheng Hu. Specification and evolution of agent from a perspective of software architecture [J]. Journal of Computer Research and Development, 2002, 39(12): 1543 - 1549.

[10] 焦文品, 史忠植. 构造 MAS 的动态体系结构的模型 [J].计算机学报, 2000, 23(7): 732 - 737.
Jiao Wenpin, Shi Zhongzhi. Modeling dynamc architectures for multi-agent systems [J]. Chinese Journal of Computers, 2000, 23 (7): 732 - 737.

[11] 于振华, 蔡远利. 基于面向对象 Petri 网的软件体系结构描述语言 [J]. 西安交通大学学报, 2004, 38(12): 1236 - 1240.
Yu Zhenhua, Cai Yuanli. Software architecture description language based on object-oriented Petri nets [J]. Journal of Xi'an Jiaotong University, 2004, 38(12): 1236 - 1240.

[12] Milner R, Parrow J, Walker D. A calculus of mobile processes [J]. Journal of Information and Computation, 1992, 100(1): 1 - 77.

[13] Clarke E, Wing J. Formal methods: State of the art and future [J]. ACM Computing Surveys, 1996, 28 (4): 626 - 643.

[14] Kavi K M, Aborizka M, Kung D. A framework for designing, modeling and analyzing agent based software systems [C]//Proc 5th International Conf Algorithms and Architectures for Parallel Processing, 2002, 196 - 200.

[15] Goudarzi K. Consistency preserving dynamic reconfiguration of distributed systems [D]. Ph D thesis, Imperial College London, 1998.

[16] Canal C, Pimentel E, Troya J M. Compatibility and inheritance in software architectures [J]. Science of Computer Programming, 2001, 41: 105 - 138.

[17] 马晓星,余萍,陶先平,等. 一种面向服务的动态协同架构及其支撑平台 [J]. 计算机学报, 2005, 28(4): 467 - 477.
Ma Xiaoxing, Yu Ping, Tao Xianping, et al. A service-oriented dynamic coordination architecture and its supporting system [J]. Chinese Journal of Computers, 2005, 28(4): 467 - 477.