协作策略驱动的构件组合行为提取*

刘奕明　赵文耘　彭鑫　唐姗

复旦大学计算机科学与工程系软件工程实验室，上海，200433
051021050@fudan.edu.cn

摘　要:

构件组合系统的行为通过构件之间的交互来体现.而构件交互遵循的协作策略往往决定组合系统所具有的性质,如安全性,活性和公平性等.因此,如何从构件交互中提取遵循特定构件交互策略的构件组合行为,从而使系统具有某种良好性质就成为了一个亟待解决的问题.本文提出了一种策略满足的构件组合行为提取方法.在这篇文章中,构件组合系统的设计和行为用接口自动机网络建模,它由一组通过共享动作同步的接口自动机组成.而构件协作策略用线性时序逻辑(LTL)描述.这个方法首先将用户指定的构件协作策略强加于组合系统的行为模型之上,并从结果模型中提取接受执行路径.然后,基于获取的行为构造协作环境,驱动组合系统中的构件以一种无死锁,协作策略满足的方式一同工作,进而使运行系统具有良好的属性,如安全性,公平性等.为了说明这个方法的正确性和有效性,我们给出了一个实例证明.

关键词:

基于构件软件开发，构件，行为提取，接口自动机

Coordination Policy Driven Component Composition Behavior Derivation

Liu Yiming　Zhao Wenyun　Peng Xin　Tang Shan

Computer Science and Engineering Department, Software Engineering Lab of Fudan University, Shanghai 200433

Abstract:

The behaviors of component-based composed system are represented by the interaction of its composed components. And the coordination-policy that the interaction of components satisfied often determines the quality and properties of the composed systems, e.g. Safety, Liveness and Fairness etc. Therefore, the coordination policy-based behavior derivation of composed components is a significant problem that needs to be solved, where the coordination policies specify the user's desired system properties. Aim to this problem, a policy-satisfying behavior derivation approach is proposed. In this paper, the design and behavior of composed system are modeled by Interface Automaton Network (IAN) which consists of a set of Interface Automata (IA) synchronized by shared actions, and user specific component coordination policies are specified by LTL. The approach firstly enforces user specific coordination policy upon the behavior model of the deadlock-free composed system, and then retrieves accepting execution paths from the resulting model. Finally, based on the retrieved behaviors the approach constructs a Coordination Environment (CE), driving components to work together in a deadlock-free and coordination policy-satisfying manner and to achieve better composed system properties. To explain the correctness and validity of this approach, we give a corresponding example certification.

Key Words:

Component-based software development (CBSD), Component, Behavior derivation, Interface Automata

---

*本文受国家 863 计划(课题编号 2006AA01Z189)，国家自然科学基金(课题编号 60473061、60473062)资助。刘奕明 博士研究生，主要研究方向为软件工程、领域工程、企业构件化开发技术。赵文耘 教授、博导，主要研究方向为软件工程、电子商务、企业应用集成。彭鑫 博士、讲师，主要研究方向为软件体系结构、软件再工程和软件产品线。唐姗 博士研究生，主要研究方向为软件工程、领域工程、企业构件化开发技术。

1 引言

基于构件的软件开发(Component based software development, CBSD)已经成为目前软件工程的研究热点.CBSD为我们提供了一种复用已有软件模块构造复杂软件系统的方法.利用这种方法,用户只需从构件库中提取所需软件构件并将它们组合在一起,就能构造出新的软件系统[3].此时,组合系统的整体行为通过构件之间的交互来完成[4].而构件交互遵循的协作策略(Coordination Policy, CP)往往决定了组合系统所具有的性质,如安全性(Safety),活性(Liveness)和公平性(Fairness)等.因此,如何从构件交互中提取遵循特定构件交互策略的构件组合行为,从而使系统具有某种良好性质成为了一个亟待解决的问题.然而,学术界对此问题多采用消极的模型验证方法,当验证系统不具有某种性质时则对系统组成构件进行修改,但是,这对于COTS(Commercial-Off-The-Shelf)构件来说是行不通的.而对于如何在不修改系统组成构件的前提下,主动地从组合系统中提取满足指定协作策略的构件组合行为的研究还显不足,本文试图解决这一问题.

本文提出了一种策略满足的构件组合行为提取方法.这个方法首先将用户指定的构件协作策略强加于组合系统的行为模型之上,并从结果模型中获取可接受的执行路径.然后,基于获取的行为,构造协作环境(Coordination Environment, CE).协作环境类似于连接器和适配器[6],但是,它能主动地驱动组合系统中的构件以一种无死锁,协作策略满足的方式一同工作,从而使运行系统具有良好的属性,如安全性,公平性等.

有效地复用已有构件需要某种形式化语言准确地描述构件的接口性质并支持设计阶段构件组合相关性质的检查.通常的形式化方法对接口性质的检查只局限于输入输出数据的静态检查上.但是,构件设计者可能会考虑范围更广泛的外部环境假设,比如,在面向对象设计中,要求在调用某个方法之前必需首先调用相关的初始化方法等等,这就存在一个对时序的假设,因此我们需要语义更为丰富的接口模型.本文使用了一种基于自动机的形式化语言——接口自动机(Interface Automata, IA)[2]及其网络来描述构件系统的行为模型.

线性时序逻辑(LTL)[5] 是一种重要的描述和验证系统特性的形式化工具.它采用线性,离散且与自然数同构的时间结构,以路径(或状态序列)作为命题论断对象.我们将使用它作为组合系统构件协作策略的描述语言.

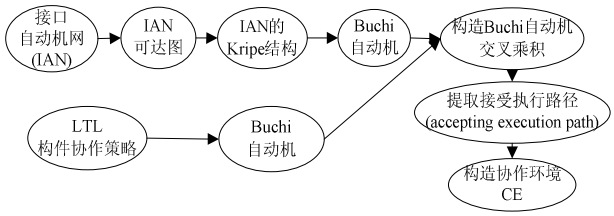

图1.基于协作策略的构件组合行为提取过程

如图1所示,我们的方法使用接口自动机网络(Interface Automaton Network, IAN)对组合系统的设计和行为进行建模,它包含一组分别代表系统组成构件行为的接口自动机,它们通过共享动作进行同步.用户指定构件协作策略P则用线性时序逻辑(LTL)公式描述.为了使组合系统和构件协作策略具有相同的语义基础,方法定义IAN行为模型的Kripke结构.接着,方法分别构造与IAN和协作策略P对应的Büchi自动机, B_{IAN} 和 B_P.设L(B_{IAN})和L(B_P)分别代表能被 B_{IAN} 和 B_P 接受的语言.这样,将构件协作策略强加于组合系统行为模型的过程,将转化为 B_{IAN} 和 B_P 交叉乘积(cross-product)的构造,以获得接受L(B_{IAN})∩L(B_P)语言的Büchi自动机.然后,通过考虑交叉积Büchi自动机中的接受执行路径,我们将能获得组合系统中满足协作策略的行为.最后,通过对提取行为中的输入和输出动作进行取反,为组合系统构造一个无死锁,协作策略满足的协作环境CE.

本文第2节简要介绍接口自动机.第3节引入一个贯穿全文的实例,然后给出作为组合系统行为描述的接口自动机网络的形式化定义及其兼容可达图的构造.第4节简单介绍如何使用LTL描述构件协作策略.第5节给出策略满足构件组合行为提取方法的形式化描述.第6节进行相关工作比较.第7节总结全文并对今后工作进行展望.

2 接口自动机

接口自动机(Interface Automata, IA)[2]是一种轻量级语言能用于描述构件接口的时序特征.它描述了使用构件时其对外界环境的输入假设和输出保证,即构件方法被调用的先后次序以及构件向外环境输出调用信息或结果的次序.

接口自动机采用与传统组合兼容性不同的思想,认为构件的设计总是基于一定的环境假设,当两个自动机组合时,它们的环境假设也应该同时组合在一起.那么只要存在某个环境使组合假设得以满足,则这二者可兼容.其语法形式与 I/O 自动机相似,都基于有限迁移系统.系统的每个状态上可能有输入,输出或内部动作(用符号"?", "!"和";"分别表示).在接口自动机中,明确地表示出在当前状态上只有哪些输入动作是可接受的,而其它的输入动作是不可接受的,即非法的输入.构件对环境的假设就是通过对系统状态上的某些输入动作做出限制来达到的.

定义1.1(接口自动机,IA).接口自动机是一个六元组 $P = (V_P, V_P^{Init}, A_P^I, A_P^O, A_P^H, \Gamma_P)$其中:

- V_P 是状态的有穷集合,每一个状态为$vi(0 \le i \le |V_P|)$;
- $V_P^{Init} \subseteq V_P$ 是初始状态集, V_P^{Init} 至多包含一个状态 v_P^{Init} ;如果 V_P^{Init} 为空集,则自动机P为空;
- A_P^I, A_P^O, A_P^H 分别是输入动作,输出动作和内部动作集合,且两两互不相交.记所有动作的集合为 $A_P = A_P^I \cup A_P^O \cup A_P^H$;
- $\Gamma_P \subseteq V_P \times A_P \times V_P$ 是迁移的集合.当 $a \in A_P^I, A_P^O$ 或 A_P^H 时,相应的迁移(v, a , v')分别称为输入,输出或内部迁移.对于某一状态 $v \in V_P$,当存在迁移(v, a , v') $\in \Gamma_P$, $v' \in V_P$,称a $\in A_P$ 在此状态上是可激活的.接口自动机不要求对任一状态$v \in V_P$,都有 $A_P^I = A_P^I(v)$.

定义 1.2(可组合).两个接口自动机 P 和 Q 若满足条件: $A_P^H \cap A_Q = \varnothing$, $A_P^I \cap A_Q^I = \varnothing$, $A_Q^H \cap A_P = \varnothing$且 $A_P^O \cap A_Q^O = \varnothing$,则称它们是可组合的.令 $share(P,Q) = A_P \cap A_Q = (A_P^I \cap A_Q^O) \cup (A_P^O \cap A_Q^I)$.

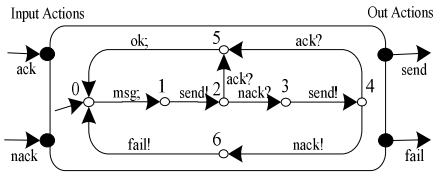

图2.接口自动机示例

图 2 中给出了用于消息传递的一个服务构件的接口自动机模型的直观形式.它提供一个外部调用方法 msg,当此方法被调用后,构件返回 ok 或 fail 表示消息发送成功与否.为了完成这个服务,它需要与一个通信信道交互,调用信道的 send 方法,并获得返回的 ack,然后返回通信成功 ok;若连续两次 send 都收到 nack,则返回通信失败 fail.从图中可以看出:输入动作 msg 仅在状态 0 可以接受,在其余的状态上则为非法,这就刻画了构件对环境的假设:一旦环境调用了 msg 方法,则在下一次调用 msg 之前,环境必须要等到该服务构件返回 ok 或者

fail.其余状态如此类推.图中的“!”表示输出动作, “?”表示输入动作,初始状态带有一个无源箭头.

3 构件组合系统设计模型和接口自动机网络

3.1 实例

一个构件系统可以看作是由一组构件经组合而构成的网络.构件系统的行为规约可以由系统组成构件行为规约的乘积来描述.因此,我们使用接口自动机网络(Interface Automaton Network, IAN)对构件组合系统的设计和行为进行建模.IAN由一组接口自动机组成,分别代表各个软件构件的抽象行为.在IAN中,组合的接口自动机通过共享动作进行同步.下面我们将给出其形式化定义,其中部分概念参考自文献[1,2].

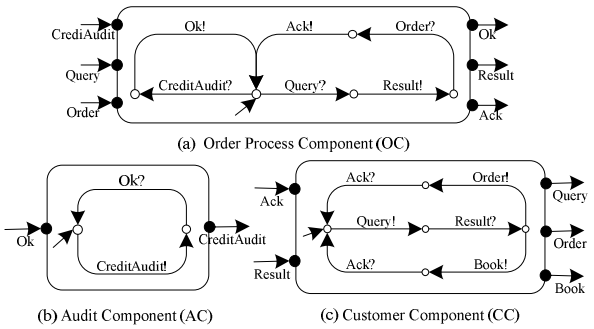

图3. 在线购物系统的组合构件

考虑图3中给出的被简化了的在线购物系统(Online Shopping System, OSS).它包含三个业务构件,它们是审计构件(Audit Component, AC),顾客构件(Customer Component, CC)和订单处理构件(Order Process Component, OPC).审计构件,由银行系统提供,它使得在线商店可以对用户的信用进行审计(*CreditAudit*),判断客户是否有足够的信用额度支付订单.顾客构件,使得在线客户能对产品列表进行查询(*Query*),跟着对喜好商品进行订购(*Order*)或预订(*Book*).订单处理构件是一个平台构件,它能让在线客户和商店在网上进行贸易.根据定义1.2,这三个构件是可组合的.但是,在这个例子中,订单处理构件并不支持商品预订(*Book*)处理.所以,它不能与顾客构件中的*Book*操作进行同步,而这将会给组合系统带来死锁可能性.

为了保证OSS的支付安全性(*payment security*,它指在线客户具有足够的Credit支付商品订单),如果在线客户执行了订购操作(*Order*)后,此时,如果在线商店没有使用Audit构件对客户的Credit进行审计(*CreditAudit*),那么OSS将不允许在线客户继续进行商品的定购,反之亦然.所以,设计者期望OSS组合系统中的Audit, Customer和Order Process构件应基于交替协作策略(*alternating coordination policy*)进行交互,从而使OSS具有良好的支付安全性.如何用LTL公式描述构件协作策略将在下面的章节中讨论.

3.2 接口自动机网络

接口自动机网络由一组接口自动机组成,它们分别代表软件构件的行为抽象[1].共享动作指一个接口自动机的输入动作可能与另一个接口自动机的输出动作一致,即动作名相同但是在不同的IA中分属输入动作集或输出动作集.记为$shared(Ci,Cj)= A_{Ci} \cap A_{Cj} = (A^O_{Ci} \cap A^I_{Cj}) \cup (A^I_{Ci} \cap A^O_{Cj})$.组合之后这些共享动作即作为IAN的内部动作,它实际上包含一对输入和输出动作组.构件之间的交互就是通过共享动作来表达.另外,每个共享动作的同步是在两个构件之间进行的.

定义3.1(接口自动机网络,IAN).接口自动机网络是二元组$N=(K,S)$, 其中:

- $K = \{C_1, C_2,\ldots,C_n\}$表示可组合的接口自动机集合, 其中$C_i=(V_{Ci}, V^{Init}_{Ci}, A^I_{Ci}, A^O_{Ci}, A^H_{Ci}, \Gamma_{Ci})(1\le i\le n)$;
- $S = \{shared(C_i,C_j)\mid 1 \le i, j \le n, i\ne j\}$表示所有的共享动作集.

定义3.2(IAN的状态和动作集). $N=(K, S)$是一个IAN,其中$K = \{C_1, C_2,\ldots,C_n\}$:

- N的状态集为$V_N=(V_{C1}\times V_{C2}\times\ldots\times V_{Cn})$,其中每个组合状态形为$\vec{v}=(v_1, v_2,\ldots, v_n)$ $(v_i\in V_{Ci}, 1\le i\le n)$;
- N的初始状态集为$V^{Init}_N \subset V_N$, V^{Init}_N中最多包含一个初始状态$v^{Init}_N=(v^{Init}_{C1}, v^{Init}_{C2},\ldots, v^{Init}_{Cn})$;
- N的动作集为$A_N = A^I_N \cup A^O_N \cup A^H_N$,其中,输入动作集为$A^I_N=(\bigcup_{1\le i\le n} A^I_{Ci})\setminus S$,输出动作集为$A^O_N=(\bigcup_{1\le i\le n} A^O_{Ci})\setminus S$,内部动作集为$A^H_N=(\bigcup_{1\le i\le n} A^H_{Ci})\setminus S$.

定义3.3(IAN的状态迁移). $N=(K,S)$是一个IAN,其中$K=\{C_1,C_2,\ldots,C_n\}$. 给定组合状态$\vec{v}=(v_1,v_2,\ldots,v_n)$和$\vec{v}'=(v'_1,v'_2,\ldots,v'_n)$,当满足以下任一条件时,系统可通过动作$a$从状态$\vec{v}$到达状态$\vec{v}'$,用$\vec{v}\xrightarrow{a}\vec{v}'$表示:

- 对于一个动作$a\notin shared(C_i,C_j)(1\le i,j\le n, i\ne j)$,在$C_k(1\le k\le n)$中存在一个迁移$(v_k, a, v'_k)\in\Gamma_{Ci}$,而且对于任何$i(i\ne k, 1\le i\le n)$,有$v_i=v'_i$;
- 对于一个动作$a\in shared(C_i,C_j)(1\le i,j\le n, i\ne j)$,在$C_i$中存在一个迁移$(v_i, a!, v'_i)\in\Gamma_{Ci}$ ($a!$表示a是输出动作);同时,在C_j中存在一个迁移$(v_j, a?, v'_j)\in\Gamma_{Cj}$ ($a?$表示a是输入动作),且对于任何$k(k\ne i, j, 1\le k\le n)$,有$v_k=v'_k$.

与IO自动机不同,在IA中输入动作并非在每个状态上都是可以激活的,所以IAN中可能会出现:在某个组合状态上一个IA对另一个IA有输出动作,但是后者并没有相应的输入动作接受.实际上,这反映了两个具有共享接口的构件对外界环境假设相互之间存在矛盾的部分.这种类型的组合状态被称之为非法状态(*illegal state*).在提取IAN构件协作策略满足行为时,需要去除IAN组合状态集合中可达的非法状态.而可达的合法状态表示两个构件接口一致的输入,输出交互部分.IAN的非法状态集定义如下.

定义3.4(IAN非法状态集).$N=(K,S)$是一个IAN,其中$K=\{C_1, C_2,\ldots,C_n\}$, 其非法状态集合为:$illegal(N)=\{(v_1,v_2,\ldots,v_n)\in V_N \mid \exists(v_i,v_j)(i\ne j, 1\le i,j\le n), \exists a\in shared(C_i,C_j), (a\in A^O_{Ci}(v_i)\wedge a\notin A^I_{Cj}(v_j))\vee(a\in A^O_{Cj}(v_j)\wedge a\notin A^I_{Ci}(v_i))\}$.

只要某一个组合状态上所有可能的共享动作中有一个出错,即认为此组合状态为非法状态.从非法状态出发,根据文献[2]中给出的逆向可达算法,可以构造出只包含兼容组合状态的IAN,记为Comp(N).Comp(N)的状态集中不再包含非法状态和那些从初始状态出发不可达的状态.

定义3.5(兼容IAN的行为). $Comp(N)=(K,S)$是一个兼容的IAN,状态迁移序列$\vec{v}_0\xrightarrow{a_0}\vec{v}_1\xrightarrow{a_1}\ldots\xrightarrow{a_{n-1}}\vec{v}_n\xrightarrow{a_n}\vec{v}_{n+1}$是Comp(N)的一个行为当且仅当$\vec{v}_0=v^{Init}_{Comp(N)}$,且对每一个$i(0\le i\le n)$有$(\vec{v}_i, a_i, \vec{v}_{i+1})\in\Gamma_{Comp(N)}$.

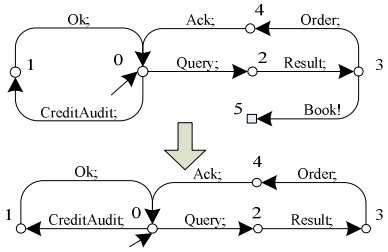

图4. IAN的无死锁行为图

图4给出了OSS组合系统IAN行为模型的原始组合行为图和无死锁/兼容组合行为图.在原始组合行为图中,状态5是一个非法状态,它将导致OSS的死锁.因为,在执行完*Book*!输出动作后,Customer构件将期望与Order Process 构件在共享动作*Ack*动作上进行同步.但是,实际上Order Process构件并不准备在Customer构件执行*Book*!动作后,与它的*Ack*?动作进行同步.这是因为Order Process构件只期望在Customer构件执行*Order*!输出动作后,与它的后继*Ack*?动作进行同步.换句话说,Order Process构件并不支持预订(Book)业务.所以,为了从Audit, Customer和Order Process构件组合中衍生OSS的无死锁组合行为,状态5和它的向后迁移(backward transition)都应被删除.

定义3.6(兼容IAN的可达图). *Comp*(*N*)=(*K*,*S*)是一个兼容的IAN,其对应的可达图$G=(V;T;LA)$可以构造如下:

- V是有穷的节点集,*Comp*(*N*)中每一个状态$\bar{v}_i$对应于V中的一个节点,起始节点v_0就是$\bar{v}_{comp(N)}^{Init}$;
- T是有穷的边集,*Comp*(*N*)中每一个迁移$(v_i, a_i, v_{i+1}) \in \Gamma_{comp(N)}$对应可达图G中一条边$t_i=(v_i, l_i, v_{i+1})\in T$, $l_i \in LA$是边上的标记,对应于迁移中的动作a,其中LA是有穷标记集.

显然,Comp(N)中的任一行为对应于可达图G中的一条路径.不妨设$\rho=t_0\hat{}t_1\hat{}\ldots\hat{}\ t_n$是G中的任一条路径,其相应边上的标记序列为$l_0\hat{}l_1\hat{}\ldots\hat{}l_n$,由可达图的构造可知该标记序列对应于一个动作序列$a_0\hat{}a_1\hat{}\ldots\hat{}a_n$.

- 对可达图G中标记为内部动作a_i;的边t,引入中间节点v'将t分割成t′和t″,并用a_i!和a_i?分别进行标记,其余的保持不变.

实际上,兼容IAN的可达图是IAN的无死锁行为图,它展开了组合系统构件之间的基于共享动作的同步操作,显式地给出组合系统中所有可能的构件交互动作序列.图5展示了实例OSS兼容IAN的对应可达图.下面讨论的基于构件协作策略的构件组合行为提取过程,就是在兼容IAN的可达图上进行的.

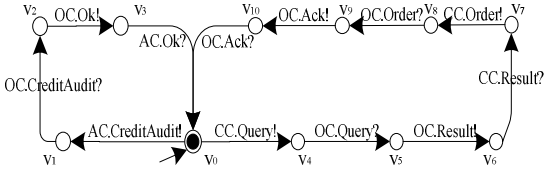

图5. IAN的可达图

4 基于 LTL 的构件协作策略说明

兼容IAN应遵循的构件协作策略,主要与IAN中组合构件的交互行为相关.IAN中的行为如果不遵循指定协作策略将被认为是组合系统的行为错误.由于,IAN中的行为以IAN中构件的动作执行序列给出.所以,在声明构件协作策略时,我们必须区分由构件C_i执行的动作a和另一构件$C_j(i\neq j)$执行的动作a.因此,IAN构件协作策略应根据构件$C_1[f_1],C_2[f_2],..,C_n[f_n]$的输入/输出动作描述,其中,$f_i$是重标记函数,对于所有$a\in A_i$,有$f_i(a) = C_i.a$,这里 A_i 是构件C_i的动作集合

参考一般模型检验的方法[5],我们使用线性时序逻辑(LTL)作为构件协作策略的描述语言.我们定义 $AP = \{\gamma: \gamma=l \vee \gamma=\neg l,$ 其中, $l\in C_i[f_i], i=1,...,n\}$为一组原子命题,并在此基础上定义与构件协作策略相关的 LTL 公式.有关 LTL 语法和语义可参考[5].

5 协作策略驱动的行为提取

5.1 IAN和协作策略的语义统一

LTL公式的语义可以用Kripke结构定义.所以,为了统一IAN和协作策略的形式化基础,我们将定义与IAN对应的Kripke结构,记为 K_{IAN},其定义如下.

定义5.1(IAN的Kripke结构). 设G=($V;T;LA$)为兼容IAN的可达图,那么其Kripke结构可定义为 $K_{IAN}=(V, T, \{v_0\}, LV)$,其中$V=V, T=T, LV=2^{LA}$,且$LV(v_i) = \{a_i: LA((v_{i-1}, v_i)) = a_i, (v_{i-1}, v_i)\in T\}$.对于每一个$v\in V$, $LV(v)$代表在状态v上为true的原子命题集.

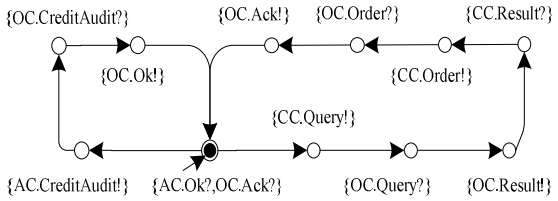

图6. IAN的Kripke结构

图6展示了实例IAN的Kripke结构.带无源箭头的节点表示初始状态.

在第4节,我们介绍了如何用LTL描述构件协作策略,指定兼容IAN必需满足的构件交互行为.接着这一节定义了兼容IAN的Kripke结构,使得兼容IAN和使用LTL公式描述的构件协作策略具有了相同的语义基础.基于Kripke结构,我们可以构造与IAN和构件协作策略P对应的Büchi自动机.设它们分别为B_{IAN}和B_P,那么$L(B_{IAN})$和$L(B_P)$就是能被B_{IAN}和B_P接受的语言.这样,将构件协作策略P强加于组合系统行为模型IAN的过程,将等价于它们对应Büchi自动机的交叉乘积(cross-product)构造,记为$B_{intersection}^{IAN,P}$,它接受$L(B_{IAN})\cap L(B_P)$.如果$B_{intersection}^{IAN,P}$为空,意味着兼容IAN的所有可能行为都违反构件协作策略P.在这种情况下,从IAN中提取满足协作策略的行为是不可能的.否则,我们可以通过提取$B_{intersection}^{IAN,P}$中的接受执行路径获得兼容IAN中满足协作策略的行为.

定义5.2(Büchi自动机,BA). Büchi自动机是一个5元组$<S,A,\Delta,q_0,F>$,其中, S是有限的状态集合,A是有限的字母集合,$\Delta\subseteq S\times A\times S$是一个由字母标识的迁移关系,$q_0\in S$是初始状态,$F\subseteq S$是可接受状态集合.

Büchi自动机的一个有限执行路径是满足对所有$0\le i\le n$存在$a_i\in A$,使得$(q_i,a_i,q_{i+1})\in\Delta$的状态序列$\sigma = q_0\ q_1...q_n$. BA的一个无限执行路径是满足对所有$0\le i$存在$a_i\in A$,使得$(q_i,a_i,q_{i+1})\in\Delta$的状态序列$\sigma = q_0q_1....$字母表$A$上的一个无限长字符串$\omega = a_0\ a_1\ a_2...$对BA来说是可接受的,当且仅当存在一个无限执行路径$\sigma = q_0\ q_1...$,满足q_0是初始状态,$(q_i,a_i,q_{i+1})\in\Delta$,且$inf(\sigma)\cap F\neq\varnothing$,其中的$inf(\sigma)$表示在路径$\sigma$中出现无限多次的状态的集合.

根据3.1节实例中给出的对Customer构件,Audit构件和Order Process构件之间交互应遵循的交替协作策略(*alternating coordination policy*)描述,我们给出它的LTL公式表示:

$$P=\mathbf{F}((CC.Order!\wedge\mathbf{X}(\neg CC.Order!\mathbf{U}AC.CreditAudit!))\vee(AC.CreditAudit!\wedge\mathbf{X}(\neg AC.CreditAudit!\mathbf{U}CC.Order!)))$$

在这个公式中,我们使用了三个LTL时序操作符:**F**,**X**和**U**.它们分别是"最终"或"将来","下一时刻"和"直到"时序操作符.

该策略是为了保证OSS具有支付安全性.它要求在线客户执行了*Order*!动作后,如果在线商店没有使用Audit构件对客户的Credit进行审计(*CreditAudit*!),那么OSS将不允许在线客户继续进行商品的订购,反之亦然.在图7中,我们将P转换成对应的Büchi自动机B_P,在这里p_0和p_3分别是初始状态和接受状态.

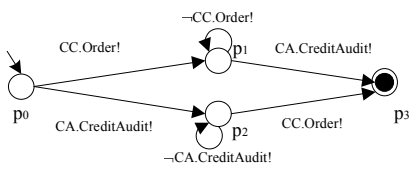

图7. 协作策略P的Büchi自动机

根据文献[5]的定义,若给定一个Kripke结构M可以构造与

其对应的一个Büchi自动机 B_M .所以,根据文献[5]中给出的Büchi自动机生成算法,我们可以构造与兼容IAN对应的Büchi自动机 B_{IAN} 和与协作策略P对应的Büchi自动机 B_P (图7, 图8分别给出与实例IAN和交替协作策略P对应的Büchi自动机),其中的双环圆黑节点代表可接受状态.

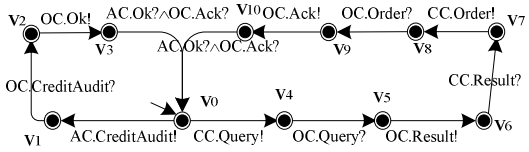

图8. IAN的Büchi自动机

5.2 策略满足的行为提取

给定两个Büchi自动机, B_{IAN} = $(V,\Delta,\{v_0\},V)$ 和 B_P = $(P,\Gamma,\{p_0\},F)$,下面给出如何从兼容IAN中提取满足构件协作策略P的行为的过程,:

1) 构造接受 $L(B_{IAN})\cap L(B_P)$ 的自动机:这个自动机定义为 $B^{IAN,P}_{intersection}$ = $(V\times P,\Delta',\{<v_0,p_0>\},V\times F)$,其中,对于组合状态 $<v_i,p_j>,<v_m,p_n>\in V\times P$, $(<v_i,p_j>,a,<v_m,p_n>)\in\Delta'$ 当且仅当 $(v_i,a,v_m)\in\Delta$ 和 $(p_j,a,p_n)\in\Gamma$ $(v_i,v_m\in V, p_j,p_n\in P)$;

2) 如果 $B^{IAN,P}_{intersection}$ 非空则返回它作为包含兼容IAN中满足协作策略P的执行路径的Büchi自动机;

3) 从 $B^{IAN,P}_{intersection}$ 中提取满足给定协作策略的行为序列.这个过程只需考虑 $B^{IAN,P}_{intersection}$ 中的接受执行路径(图9中由粗箭头连成的执行路径);

下面我们给出 $B^{IAN,P}_{intersection}$ 接受执行路径的定义:

定义5.3($B^{IAN,P}_{intersection}$ 的接受执行路径). $B^{IAN,P}_{intersection}$ = $(V\times P,\ \Delta',\ \{<v_0,p_0>\},\ V\times F)$是接受 $L(B_{IAN})\cap L(B_P)$ 的自动机. $B^{IAN,P}_{intersection}$ 的接受执行路径是形如 $\gamma = s_0, s_1, .., s_n$ 的状态序列:其中对于 $\forall i=0,..,n: s_i\in V\times P$ 是 $B^{IAN,P}_{intersection}$ 的组合状态;对于 $0\le i\le n-1$, $(s_i, s_{i+1})\in\Delta'$ 并且 $(s_n, s_0)\in\Delta'$ 或 $(s_n, s_0)\notin\Delta'$;$\exists k=0,..,n: s_k\in V\times F$.

注意,根据协作策略的不同, $B^{IAN,P}_{intersection}$ 的接受执行路径可能是一个环(如,使用always时序操作符声明的协作策略).在这种情况下,为了提取满足指定协作策略的行为,我们将不考虑不包含接受状态的环执行路径.

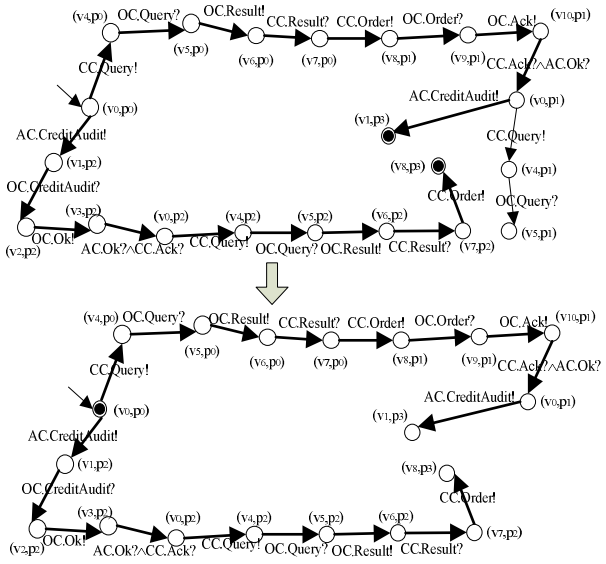

图9. IAN的协作策略满足行为图(局部)

图9展示了如何从 $B^{IAN,P}_{intersection}$ 中提取实例IAN中满足协作策略P的行为.在图9的上部,我们给出了 $B^{IAN,P}_{intersection}$ 的执行路径局部.根据 $B^{IAN,P}_{intersection}$ 的构造过程可知其中的组合状态 $<v_1,p_3>$ 和 $<v_8,p_3>$ 是接受状态,它们代表满足协作策略P的IAN状态.所以图中所有包含这两个组合状态之一的路径都是接受执行路径.为了使IAN的运行遵循给定协作策略P, 我们只抽取 $B^{IAN,P}_{intersection}$ 中的接受执行路径来构造IAN的行为图.另外,在这个例子中由于协作策略定义的关系,其中的接受状态 p_3 也是终止状态(*stop state*),所以,当IAN到达接受状态时IAN的运行应当返回它的初始状态(这里是 $<v_0, p_0>$).

6 协作环境的构造和实现

协作环境的构造是为了使IAN中构件能根据提取的无死锁和策略满足的构件组合行为路径进行交互.所以,CE的行为图可以简单的通过对提取的策略满足行为中的输入和输出动作取反来获得.也就是说,假设兼容IAN的提取行为中有一个 $Ci.a$ 动作,相应地,在协作环境中就会有一个 $Ci.\overline{a}$ 动作与其对应, $\overline{a}$ 是 a 的同步动作,例如, $\overline{a}! = a?$, $\overline{a?} = a!$.然后,CE将在IAN组合构件中扮演协调器(*Coordinator*)的角色,根据协作策略接受一方交互构件的动作,然后将其转发至另一方交互构件.图10,给出了基于协调器的体系结构模型(Coordinator Based Architecture, CBA).

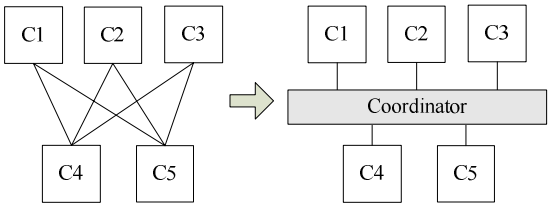

图10.基于协调器的组合系统体系结构

定义6.1(基于协调器的体系结构,CBA). CBA ≡ $(C_1[f_1]|C_2[f_2]|...|C_n[f_n]|Coordinator)\backslash\bigcup_{i=1}^{n} Act_i[f_i]$,其中,对于 $1\le i\le n$, f_i 是重标记函数, C_i 是构件行为图,对所有 $a\in Act_i$,有 $f_i(a) = C_i.a$,这里 Act_i 是构件 C_i 的动作集合,*Coordinator*是协作环境的行为图.

从定义6.1可知组合系统的交互构件将不再直接地进行同步,而是通过CE来进行.另外,CE并不是简单地接受每个交互构件的每个动作,而是根据行为图中的接受执行路径选择性地接受交互构件的动作.所以,在组合系统遵循无死锁,策略满足行为运行的过程中, 协作环境将起主导作用.

通过遍历CE的行为图,并利用存储在行为图状态和迁移中的信息,我们可以自动地生成实现协作环境的代码.在我们的实现中,通过以C++作为编程语言使用Microsoft COM和ATL技术,我们成功地在我们的在线购物系统中构造了一个协作环境CE,并将其实现为一个协调器构件Coordinator,协调在线购物系统的运行,使系统应用具有良好的支付安全性.

下图11展示了协调器构件Coordinator的IDL,COM库及其COM类定义的局部.可以看出,协调器构件通过定义Coordinator类并使用封装机制(*wrapping mechanism*) 对Audit和Customer构件对Order Process构件的请求操作进行封装,实现了Order Process构件的COM接口OrderProcess. 其中,orderPrcessObj是Order Process构件的内部COM服务器,Coordinator将引用它实现其COM接口orderProcess.另外,下图还展示了协调构件Coordinator实现orderProcess接口方法Order的无死锁,策略满足的实现代码.

```
import orderprocess.idl; … library Coordinator_Lib {
…
coclass Coordinator {
    [default] interface OrderProcess;
}}
…
class Coordinator : public OrderProcess {
    // stores the current state of the coordinator
    private static int sLbl;
    // stores the current state of the coordination policy  automaton
    private static int pState;
    // stores the number of clients
    private static int clientsCounter = 0;
    // channel's number of a client
    private int chId;
    // COM smart pointer; is a reference to the Order  Process Component server object
    private static OrderProcess* orderprocessObj;
    …
    // the constructor
    Coordinator () {
          sLbl = 1;
          pState = 0;
          clientsCounter++;
          chId = clientsCounter;
          orderprocessObj = new OrderProcess();
          …     }
// implemented method
…
HRESULT order (/* params list of order */) {
  if(sLbl = = 0) {
      if(chId = = 1) && (pState = = 0)) {
           return orderprocessObj-> order (/* params list of order */);
           pState = 1; sLbl = 0; //it goes on the state preceding the next
           //request of a method from a client
      }
      else if(chId = = 1) && (pState = = 2)) {
           return orderprocessObj-> order (/* params list of order */);
           pState = 0; sLbl = 0; //since it has found an accepting stop node,
           //it returns to its initial state}
  }
  else if(sLbl = = 5) {
       if(chId = = 2) && (pState = = 1)) {
                return orderprocessObj-> order (/* params list of order */);
                pState = 0; sLbl = 0; //since it has found an accepting stop node,
                //it returns to its initial state}
          else if(chId = = 2) && (pState = = 0)) {
          return orderprocessObj-> order (/* params list of order */);
          pState = 2; sLbl = 0; //it goes on the state preceding the next
          //request of a method from a client}
    }
   return E_HANDLE;
   }}
  …  }
```

图11. 协调器构件Coordinator的实现(局部)

7 相关工作

CBSD的重点是解决构件获取和构件组合这两个问题.构件组合的目的是解决系统功能的实现.此时,组合系统的整体行为通过构件之间的交互来完成.但是,系统组合行为里面可能存在违反用户指定构件协作策略的执行部分.因此,我们或许可以对系统组成构件进行修改,但是,这对于 COTS 构件来说是行不通的.

文献[7][8]主要解决了构件组合中行为的兼容问题,即构件间交互顺序的匹配问题.但是它们并没有考虑组合后的行为是否满足构件协作策略,以及如何从组合后的行为中抽取策略满足的行为.文献[10]提出了一个基于场景的构件行为抽取的方法,但是该方法没有给出如何判断组合后行为是否满足用户指定系统协作属性的方法.文献[12]提出了一种自动构件获取与适配的方法,但该方法也只是考虑单个构件的获取与适配,并没有考虑构件组合后的行为是否满足系统协作属性.文献[9]提出了基于形式化规约匹配的方法,但是这些方法只适用于离散式构件的获取与组合.文献[11]也采用有限状态机来描述构件的行为,并给出了相应的分类、检索和适配的方法,但是该方法也只是考虑单个构件的行为.

本文给出的方法通过对环境的使用,根据用户指定构件协作策略从组合系统中抽取用户期望的组合行为,弥补了上述研究工作的不足.

8 总结和展望

本文研究了协作策略驱动的构件组合行为提取. 通过将构件协作策略强加于组合系统的行为模型上, 从结果模型中获取可接受执行路径;然后, 根据提取的组合行为构造协作环境, 驱动组合系统以一种无死锁, 协作策略满足的方式运行;从而, 使系统运行具有良好的属性, 如安全性, 公平性或活性等. 我们使用接口自动机对构件行为进行建模. 用接口自动机网络来刻画组合系统的行为. 接着, 我们给出了方法的形式化描述, 并用实例对文中所述的方法进行了说明.

协作策略驱动的构件行为抽取方法首先可以提高CBSD中构件的复用率.另外,利用该方法可以使通过构件库中构件组合而成的系统自动地遵循给定协作策略运行,从而使开发系统获得良好的属性.这个过程无疑提高了系统开发效率和质量.

在以服务为基本组成元素而组合起来的软件系统中(如Web Services),服务与构件有许多共同特征,如模块性、可组合性和可复用性等.而服务也可以用构件技术来实现.因此,本文所述方法同样适用于服务.

本文只研究了如何从组合系统中提取满足协作策略的行为.在提取行为时,可能会有多种选择策略,我们下一步的工作目标是如何根据系统运行场景选择协作策略,获得最符合用户需求的系统组合行为和系统属性.

References:

[1] 胡军,于笑丰,张岩,王林章,李宣东,郑国梁.基于场景规约的构件式系统设计分析与验证.计算机学报,2006,29(4):513.525.

[2] de Alfaro, L. and T. A. Henzinger, Interface Automata, in: *Proceedings of 9th Annual ACM Symposium on Foundations of Software Engineering (FSE 2001), ACM* Press, New York (2001), 109-120.

[3] Szyperski, C. Component software: beyond object-oriented programming. ACM Press and Addison Wesley, New York, USA (1998).

[4] Heineman, G. T. and W. T. Councill, "Componet-Based Software Engineering: Putting the Pieces Together", Addison-Wesleg, Boston, 2001.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. Model Checking. The MIT Press, 2001.

[6] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 3, pp. 213-249, July 1997.

[7] Bracciali A, Brogi A, Canal C. A formal approach to component adaptation. Joural of Systems and Software, 2005,74(1):45-54.

[8] Yellin DM, Strom RE. Protocol specifications and component adaptors. ACM Trans. on Programming Languages and Systems,1997,19(2):292.333.

[9] Kakeshita.T, Murata.M. Specification-based component retrieval by means of examples. Proceedings of International Symposium on Database Applications in Non-Traditional Environments(DANTE '99), 1999:411~420.

[10] 张岩, 胡军,于笑丰,王林章,李宣东,郑国梁.场景驱动的构件行为抽取.软件学报, 2007,18(1):50-61.

[11] Redondo, R.P.D.; Arias, J.J.P.; Vilas, A.F.; Martinez, B.B. Approximate Retrieval of incomplete and formal specifications applied to vertical reuse. Proceedings of International Conference on Software Maintenance (ICSM'02), 3-6 Oct. 2002:618- 627.

[12] Hai-Feng Guo, Miao Liu, Jiaxiong Pi. Precise specification matching for automated component retrieval and adaptation. IEEE international conference on Information Reuse and Integration, 27-29 Oct. 2003:77-84.