基于 Pi 演算的构件演化研究

龚洪泉,赵文耘,徐如志,钱乐秋
(复旦大学计算机科学与工程系,上海 200433)

摘 要: 确保构件系统的一致性是构件演化的根本目标.根据构件交互过程,借鉴 Pi 演算的类型系统和进程构造方法,提出构件交互的类型系统和基于交互的构件模型.为确保构件服务端口和交互通道的正确行为,给出服务的端口类型和通道类型.以此为基础,结合 Pi 演算中良类型的思想,提出一致性构件系统的概念.然后,结合构件演化的特点,给出能保持系统一致性的构件静态演化和动态演化规则.最后,给出非一致演化的恢复方法.

关键词: 构件系统维护; 构件交互; 构件演化; Pi 演算
中图分类号: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2004) 12A-238-05

A Research on Pi-Calculus Based Component Evolution

GONG Hong-quan, ZHAO Wen-yun, XU Ru-zhi, QIAN Le-qiu
(*Dept. of Computer Science & Engineering, Fudan University, Shanghai* 200433, *China*)

Abstract: Preservation of consistency is the ultimate criterion of component evolution. With component interaction processes and inspired by typing system and process construction methods of Pi-calculus, component interaction typing system and interaction-based component model are proposed. To ensure correct interaction behaviors of component ports and channels, service port type and channel type are introduced. At the same time, consistent component system is defined based on the notion of well-typedness in Pi-calculus. Then, based on the peculiarity of component evolution, static and dynamic evolution rules are proposed. At last, method for recovering from inconsistent state is introduced.

Key words: component system maintenance; component interaction; component evolution; Pi-calculus

1 引言

基于构件的软件工程(CBSE)可有效地提高软件开发质量和效率,降低开发及维护成本.构件演化是构件系统维护的关键技术和研究热点[1],确保构件系统的一致性是构件演化的根本目标.

针对构件系统的维护和构件演化,国内外的研究人员做了许多奠基性的研究工作.由北京大学主持研制的集成环境"青鸟 III系统"[2]对构件获取、程序理解和构件组装等进行了系统而深入的研究,提出了基于构件-构架复用的软件开发技术.随着对构件行为的关注,文献[3,4]提出了基于 π 演算的构件组装语言 PICCOLA,该语言能支持多种形式的构件组装.这些研究为构件系统的一致性和构件交互的正确性验证奠定了基础,但它们并没有对构件演化本身作深入的研究.

本文在研究构件交互具体特点的基础上,首先借鉴 π 演算[5]中的类型系统和进程构造方法,提出构件交互的类型系统和基于交互的构件模型.为了区分构件服务端口和交互通道在构件交互过程中扮演的不同角色,给出构件服务的端口类型和通道类型.同时,在考察构件交互行为正确性的基础上,提出一致性构件系统的概念.然后,结合构件演化的特点,给出构件静态演化和动态演化规则.最后,对非一致演化所带来的影响进行分析,并提出恢复系统一致性的方法.

2 构件演化与 π 演算

在一个由构件组装而成的软件系统中,随着系统维护和需求的变化,其组成构件也将发生相应的变化,即构件演化.一方面由于系统体系结构的变化,原有的构件可能被新的构件所代替[6],从而导致构件之间连接的变化,这是构件的外部演化.另一方面,构件可能有了新的升级版本,或者被其它更有竞争力的构件所替换[7],从而导致构件规约和实现的变化,这是构件的内部演化.

在构件演化过程中,必须分析新的构件是否与现有的系统环境相一致,是否会影响系统的整体行为.一种情况是构件的演化不影响系统的整体行为,构件演化后系统仍然能保持一致性(consistency),称这种演化为一致性演化.另一种情况是,由于新技术的采用和系统环境的变化,不得不对某些构件进行演化,这往往会影响系统的整体行为,导致系统产生不一致的情况,称这种演化为非一致演化.

收稿日期:2004-10-12;修回日期:2004-11-08
基金项目:国家 863 高科技发展计划(No. 2002AA114010);上海市科委攻关项目(No. 025115014)

π演算主要用来对系统元素的交互协作行为网络进行建模,其移动性(mobility)是指改变网络连接性的能力,即改变空间配置的能力.而构件演化会改变构件之间的连接,因此,π演算可用来对构件演化进行建模.

2.1　构件交互的类型系统

构件是一个有明确接口规约和周境依赖的组装单元,它通过端口与其它构件进行交互.服务提供端口向其它构件提供服务,服务请求端口请求其它构件的服务.组成系统的构件在它们相互匹配的两个端口之间建立交互通道,以传送端口之间的交互信息(见图1).

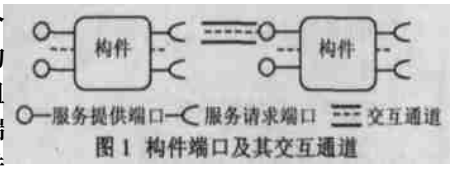

图1　构件端口及其交互通道

如果一个构件所提供的服务正好满足另一个构件所请求的服务,则它们可以互相交互.构件交互涵盖了服务匹配和交互通道的建立、服务调用和执行、服务应答和结果获取等多个阶段.设 R(Requestor)和 S(Server)分别代表请求服务和提供服务的两个构件,则它们之间有如下的交互过程:

(1)服务匹配:检测 S 所提供的服务 s 能否满足 R 所请求的服务 r;(2)交互通道的建立:R 要调用 S 的服务,必须先在端口 r 和 s 之间建立一条交互通道 c;(3)服务调用:R 通过交互通道发出调用 S 的服务的请求,同时建立应答通道;(4)服务执行:S 从交互通道接收到 R 的调用请求后,执行相应的服务;(5)服务应答:S 执行完所请求的服务后,通过应答通道发回相应的应答信息;(6)服务结果:R 从应答通道接收服务的执行结果.

π演算的类型系统最根本的用途在于防止运行时错误的出现[3~5,8],为了对构件演化的正确性进行分析和推理,本文引入如下的构件交互类型系统(图2),其中 $\Gamma=\{v_1:T_1,\cdots,v_n:T_n\}$ 称为类型环境,它为名字 v_i 赋予类型 T_i,且每个 v_i 只在 Γ 中出现一次.类型判断 $\Gamma\vdash E$ 指表达式 E 具有良类型,即表达式 E 中的所有变量在 Γ 中都有定义;$\Gamma\vdash x:T$ 指名字 x 的类型为 T;$\Gamma\vdash U\le T$ 指类型 U 是类型 T 的子类型.常用类型中的 T_{BSC} 表示 Integer,String 等基本类型.T_{SIG} 和 T_{PRD} 分别代表服务型构(signature)和谓词.T_{CTR},T_{INT} 和 T_{RLY} 都属于通道类型,它们根据通道所传送的数据类型对通道进行分类.端口类型指明构件交互过程中各个端口的角色.

$\Gamma\vdash E\quad\Gamma\vdash x:T\quad\Gamma\vdash U\le T$	类型判断
$T::=T_{BSC}\mid T_{SIG}(T\times\cdots\times T\times T_{LNK})\mid T_{PRD}(T)\mid T_{LNK}$	常用类型
$T_{LNK}::=T_C\quad T_P$	链接类型
$T_C::=T_{CTR}(T\times T\times T)\mid T_{INT}(T\times\cdots\times T\times T_{LNK})\mid T_{RLY}(T)$	通道类型
$T_P::=\pm(T_{REQ}\mid T_{SER}\mid T_{INV}\mid T_{EXE}\mid T_{REP}\mid T_{RES})$	端口类型

图2　构件交互类型系统

2.2　基于交互的构件模型

根据不同的系统环境,构件之间的交互将呈现不同的模式.结合π演算中的进程构造方法,构件交互可表示成 $R::=R_1+R_2\mid R_1\mid R_2\mid !R\mid(v,a)R\mid 0$.其中 R_1+R_2 是选择交互,表示根据系统环境,选择执行 R_1 或 R_2,但在某个时刻,只能执行其中一个.$R_1\mid R_2$ 是并行交互,即 R_1 和 R_2 并行执行.$!R$ 表示任意多份 R 的副本并行执行.$(v,a)R$ 表示变量 a 只在 R 中可见,即 a 是进程 R 的私有变量.0 表示非活动(inaction),是不执行任何活动的进程.另外,用 $R\{b/a\}$ 来表示变量 b 替代 R 中的变量 a.

π演算中的活动 $\alpha::=T_{PORT}\overline{x}\langle y\rangle\mid T_{PORT}x(y)\mid\tau$ 用来描述系统的行为.T_{PORT} 代表活动的端口类型.输出活动 $\overline{x}\langle y\rangle$ 表示沿着通道(端口)x 发送名字 y,输入活动 $x(y)$ 表示从 x 中接收名字 y,不可观察活动 τ 是进程的内部活动,在进程外部不可见.构件交互过程可用活动表示成:

$$\pi::=T_{REQ}\overline{r_C}\langle r_I\rangle\mid T_{SER}s_C(s_I)\mid T_{INV}\overline{r_I}\langle a_1,\cdots,a_s,r_R\rangle\mid T_{EXE}s_I(x_1,\cdots,x_t,s_R)\mid T_{REP}\overline{s_R}\langle b\rangle\mid T_{RES}r_R(y)$$

其中符号的含义为 T_{REQ} 请求服务,T_{SER} 提供服务,T_{INV} 调用服务,T_{EXE} 执行服务,T_{REP} 服务应答,T_{RES} 服务结果.

在系统交互中,请求服务的构件称为服务请求构件,它由一组服务请求端口组成.提供服务的构件称为服务提供构件,它由一组服务提供端口组成.由它们组成的系统称为组装好的系统.另外,一个构件通常既是服务请求构件,又是服务提供构件,即既可以请求服务又可以提供服务.基于交互的构件模型如下,其中 R_i 代表服务请求构件,S 代表服务提供构件(图3):

$$R_i(r_1,\cdots,r_m)\overset{def}{=}T_{REQ}\overline{r_C^1}\langle r_I^1\rangle.!(T_{INV}\overline{r_I^1}\langle a^1,r_I^1\rangle.T_{RES}r_R^1(y^1).0)\mid\cdots\mid T_{REQ}\overline{r_C^m}\langle r_I^m\rangle.!(T_{INV}\overline{r_I^m}\langle a^m,r_R^m\rangle.T_{RES}r_R^m(y^m).0)$$

$$S(s_1,\cdots,s_n)\overset{def}{=}!(T_{SER}s_C^1(s_I^1).!(T_{EXE}s_I^1(y_I,s_R^I).T_{REP}\overline{s_R^1}.0)+\cdots+T_{SER}s_C^n(s_I^n).!(T_{EXE}s_I^n(y^n,s_R^n).T_{REP}\overline{s_R^n}\langle b^n\rangle.0))$$

$$composedSystem\overset{def}{=}S(s_1,\cdots,s_n)\mid R_1(r_{1_1},\cdots,r_{m_1})\mid\cdots\mid R_j(r_{1_j},\cdots,r_{m_j})$$

$$Component\overset{def}{=}(T_{REQ}\overline{r_C^1}\langle r_I^1\rangle.0)\mid\cdots\mid T_{REQ}\overline{r_I^m}\langle r_I^m\rangle.0\quad!(S(s_1,\cdots,s_n)+!(T_{INV}\overline{r_I^1}\langle a^1,r_R^1\rangle.T_{RES}r_R^1(y^1).0+\cdots+T_{INV}\overline{r_I^m}\langle a^m,r_R^m\rangle.T_{RES}r_R^m(y^m).0))$$

图3　构件交互模型

2.3　端口和通道

构件通过端口来请求或提供服务,当两个构件发生交互时,将在相应的端口之间建立联系通道.服务端口由契约端口 p_{CTR}、交互端口 p_{INT} 以及应答端口 p_{REP} 组成.其中契约端口包含服务型构、前置条件和后置条件;交互端口用于服务调用和执行;应答端口用于传送服务应答信息.

由构件交互模型知,不同的端口有不同的功能特性和方向特性,即具有不同的端口类型.另外,构件交互时,通道将连接相应的端口以传送构件间的交互信息,不同的通道传送实体的能力不同,因此,不同的端口具有不同的通道类型.端口 p 的端口类型用 $T_p(p)$ 或 $p:_pt$ 表示,通道类型用 $T_c(p)$ 或 $p:_ct$ 表示.

通道类型对构件之间的交互进行约束.$p_{CTR}:_cT_{CTR}(T_{SIG}(T_1,\cdots,T_n,T_{RLY}(T)),T_{PRD}(PRE),T_{PRD}(POST))$ 表示沿着契约端口传送交互通道时,交互通道必须满足一定的契约类型.

$p_{INT}:_c T_{INT}(T_1,\cdots,T_n,T_{RLY}(T))$ 表示交互端口可以传送数值和应答信息.而 $p_{REP}:_c T_{RLY}(T)$ 表示应答端口只能传送数值.契约端口的型构 $T_{SIG}(T_1,\cdots,T_n,T_{RLY}(T))$ 反映了调用构件服务时传送参数 $T_1,\cdots,T_n$,同时建立应答通道 $T_{RLY}(T)$.前置条件和后置条件由谓词类型 T_{PRD} 表示.构件交互过程中的各个端口的通道类型如下:

$r_C:T_{CTR}(T_{SIG}(T_1,\cdots,T_n,T_{RLY}(T)),T_{PRD}(PRE),T_{PRD}(POST))$

$s_C:T_{CTR}(T_{SIG}(T_1',\cdots,T_n',T_{RLY}(T')),T_{PRD}(PRE'),T_{PRD}(POST'))$

$r_1:T_{INT}(T_1,\cdots,T_n,T_{RLY}(T))$ $s_I:T_{INT}(T_1',\cdots,T_n',T_{RLY}(T'))$

$r_R:T_{RLY}(T)$ $s_R:T_{RLY}(T')$

其中,交互端口和应答端口仅对交互的两个构件可见,即构件交互时建立的交互通道是构件间的私有通道.

3 一致性构件系统

构件交互首先要判定所提供的服务能否满足所请求的服务,即服务匹配.契约端口的通道类型由型构、前置和后置条件组成.对请求服务 $r_C:_c T_{CRT}(T_{SIG},PRE,POST)$ 和提供服务 $s_C:_c T_{CRT}(T_{SIG}',PRE',POST')$,如果 $T_{SIG}=T_{SIG}'\wedge PRE\rightarrow PRE'\wedge POST'\rightarrow POST$,则称提供服务 s_C 匹配请求服务 r_C.其隐含条件是,服务匹配时,要求 T_{REQ} 与 T_{SER} 互补,并且具有相反的端口方向,记为 $T(r_C)\Leftrightarrow T(s_C)$.

服务匹配后将在构件之间建立交互通道,当沿着一个通道传送信息时,被传送的实体的类型必须是该通道所允许的类型,即类型满足(type satisfaction)的概念.

定义 1(交互类型满足契约类型) 对交互类型 $T_I=T_{INT}(T_1,\cdots,T_n,T_{RLY}(T))$ 和契约类型 $T_C=T_{CTR}(T_{SIG},\mathrm{PRE},\mathrm{POST})$,如果服务 s 的交互端口 s_I 满足约束 $T_{SIG}=T_{SIG}(T_1,\cdots,T_n,T_{RLY}(T))$,且如果前置条件 PRE 成立,当 s_I 的执行终止时,有后置条件 POST 成立,则称交互类型 T_I 满足契约类型 T_C,记为 $T_I\vdash T_C$.

同时,根据端口通道类型的描述知,构件之间的成功交互还要求,应答类型 T_R 满足交互类型 T_I,数值类型 T 满足应答类型 T_R,即 $T_R\vdash T_I$ 和 $T\vdash T_R$.

定义 2(正确的构件演化) 能保持 $T_I\vdash T_C$,$T_R\vdash T_I$ 和 $T\vdash T_R$ 的演化称为正确的构件演化.

定义 3(活动失败) 沿着通道传送的数据违反通道的类型约束时,活动失败.

由构件交互类型系统知,类型判断 $\Gamma\vdash E$ 指表达式 E 中的所有变量在类型环境 Γ 中都有定义,即 E 具有良类型(well-typedness).良类型的根本作用在于,有了建立在构件交互类型系统之上的构件服务规约,它能确保构件演化的正确性.良类型具有如下的一些属性,由于证明比较简单,本文将不给出具体的证明:

引理 1(良类型程序中的判定不会失败) 如果 $\Gamma\vdash R$,则对 R 的执行不会失败.

引理 2(变量替代能保持良类型) 如果 $\Gamma\vdash R$,且 $\Gamma\vdash x:T,v:T$,则 $\Gamma\vdash R\{v/x\}$.

引理 3(演化能保持良类型) 如果 $\Gamma\vdash R_1$,且 $R_1\leq R_2$,则 R_1 演化到 R_2 后仍具有良类型,即 $\Gamma\vdash R_2$.

结合类型满足的概念,定义构件交互活动的良类型如下:

定义 4(构件交互的良类型)

(1)如果 $T_c(r_I)\vdash T_c(r_C)$,则 $\Gamma\vdash T_{REQ}\overline{r_C}\langle r_I\rangle$,否则 $T_{REQ}\overline{r_C}\langle r_I\rangle$失败.

(2)如果 $T_c(s_I)\vdash T_c(s_C)$,则 $\Gamma\vdash T_{SER}s_C(s_I)$,否则 $T_{SER}s_C(s_I)$失败.

(3)如果 a 是数值类型,且 $T_c(r_R)\vdash T_c(r_I)$,则 $\Gamma\vdash T_{INV}\overline{r_I}\langle a,r_R\rangle$,否则 $T_{INV}\overline{r_I}\langle a,r_R\rangle$失败.

(4)如果 y 是数值类型,且 $T_c(s_R)\vdash T_c(s_I)$,则 $\Gamma\vdash T_{EXE}s_I(y,s_R)$,否则 $T_{EXE}s_I(y,s_R)$失败.

π 演算中,0 表示不执行任何活动的非活动进程,当它是选择交互 R_1+R_2 的左元素或右元素时,称为退化 0,否则称为非退化 0.如 $0+x(y).0$ 中的第一个 0 是退化的,第二个则是非退化的.当用空位[.]替换进程表达式中的非退化 0 时,得到一个周境 C(Context).

周境与进程的区别就在于周境用空位[.]替换了进程中的非退化 0,用进程 R 替换周境 C 中的空位[.]记为 $C[R]$.例如,$C_0=\overline{z}\langle w\rangle.[.]+x(y).0$ 是一个周境,当用进程 $z(b)$ 替换空位时,得到 $C_0[z(b)]=\overline{z}\langle w\rangle.z(b)+x(y).0$.

从构件的交互模型中可以看出,每个构件本质上是一个进程表达式.由良类型的含义知,只有构件交互的进程表达式具有良类型时,系统才具有正确的行为,即系统是一致的(consistent).当组成系统的所有构件都具有良类型时,称该系统为一致性构件系统.在构件演化过程中,给定任意周境 C,当 $\Gamma\vdash C[R]$ 蕴涵 $\Gamma\vdash C[R']$ 时,构件 R 可以保持一致性地演化到构件 R'.

4 构件演化

为适应系统维护所带来的变化,组成系统的构件必须相应地演化.有的构件演化能保持系统的一致性,这可以通过静态分析构件的契约和运行时环境来判定.有时由于系统环境的变化或新技术的采用,构件演化会破坏系统的一致性,这时必须分析构件演化对其它构件的影响,以便对其它构件进行适当的调整,从而将系统恢复到一致的状态.

4.1 静态演化

构件通过端口来请求或提供服务,构件演化就意味着构件端口的演化.构件之间的交互行为由端口的契约类型和构件之间的交互通道类型决定.如果构件演化只影响端口的契约类型,则不需要重新建立构件之间的交互通道.这可以通过静态分析构件的契约来判定系统的一致性,称这种构件演化为静态演化.

由构件交互模型知,构件由一组服务端口组成,为了叙述简便,下面先讨论端口演化,再讨论构件演化.

命题 1(服务请求端口一致性演化) 如果 $T_p(r_C)=T_p(r_C')\wedge T_c(r_C)\leq T_c(r_C')$,则服务请求端口 $r_C:_p T_{REQ}$ 能保持一

致性地演化到 $r'_C:_p T_{REQ}$.

证明　由假设条件知,契约端口 r_C 和 r'_C 的端口类型相同,而 r_C 的通道类型是 r'_C 的通道类型的子类型,即 r 是 r 的精化.因此 r 有更强的前置条件和更弱的后置条件.由引理 3,如果 $\Gamma \vdash \overline{r_C}\langle r_I\rangle$,并且 $T_c(r_C) \leq T_c(r'_C)$,则 $\Gamma \vdash \overline{r'_C}\langle r_I\rangle$.另外由于静态演化不需要重新建立交互通道,即交互通道之间满足 $T_c(r'_I) \leq T_c(r_I)$,从而有 $\Gamma \vdash \overline{r'_C}\langle r'_I\rangle$,因此良类型被保持,即 r_C 可以保持一致性地演化到 r'_C.

对于端口演化,它的演化环境是端口所在的构件.而对于构件演化,它的演化环境就必须包括与之交互的其它构件,即构件组装成的系统.因此,讨论服务请求构件的演化时,必须涉及到提供服务的构件.

命题 2(服务请求构件一致性演化)　设构件交互 $R|S$ 中 S 提供的服务 s 与 R 请求的服务 r 相连接,构件 R 请求的服务 r 是 r 的演化,如果 $T_c(s_C) \leq T_c(r'_C)$,则构件交互 $R|S$ 中的服务请求构件 R 可以保持一致性地演化到 R',即 $\Gamma \vdash R|S \rightarrow \Gamma \vdash R'|S$.

证明　由服务 s 与服务 r 相连接知,$T_c(s_C) \leq T_c(s_C)$,即 s 提供的服务满足 r 请求的服务.又因为 $T_c(s_C) \leq T_c(r'_C)$,所以 s 提供的服务也满足 r 请求的服务.这意味着 R 演化到 R' 不会影响交互的行为.因此该演化可以保持良类型,是一致性演化.

服务提供端口和服务提供构件的演化与服务请求端口和服务请求构件的演化类似:

命题 3(服务提供端口一致性演化)　如果 $T_p(s_C) = T_p(s'_C) \wedge T_c(s'_C) \leq T_c(s_C)$,则服务提供端口 $s_C:_c T_{SER}$ 可以保持一致性地演化到 $s'_C:_c T_{SER}$.

证明　类似于命题 1 的证明.

命题 4(服务提供构件一致性演化)　设构件交互 $R|S$ 中 S 提供的服务 s 与 R 请求的服务 r 相连接,构件 S 提供的服务 s 是 s 的演化,如果 $T_c(s'_C) \leq T_c(r_C)$,则构件交互 $R|S$ 中的服务提供构件 S 能保持一致性地演化到 S',即 $\Gamma \vdash R|S \rightarrow \Gamma \vdash R|S'$.

证明　类似于命题 2 的证明.

4.2　动态演化

在系统运行时执行的构件演化称为动态演化.动态演化会改变构件交互的类型环境,从而导致构件间交互通道的重新建立.每个端口在交互时都有一个运行时环境(run-time environment) $T_{RTE}e(p,t_p)$ 来记录交互发生时端口的环境信息,其含义在类型环境 Γ 中动态地声明,记为 $\Gamma[T_{RTE}e(p:t_p)]$.动态地改变端口的运行时环境,就可实现交互通道的重新建立,从而达到动态演化的目的.例如运行时环境 $T_{RTE}e(r_C: T_{CTR}(T'_{SIG}\text{PRE}', \text{POST}'))$ 的执行将改变类型环境 中已有的契约绑定 $r_C: T_{CTR}(T_{SIG}, \text{PRE}, \text{POST})$.

在构件形式化模型的基础上引入运行时环境,服务请求构件和服务提供构件的规约变成:

$R_i \stackrel{def}{=} !(T_{RTE}e(r:t_r).\overline{R_i}\langle r\rangle)\ S \stackrel{def}{=} !(T_{RTE}e(s:t_p).S(s))$

服务请求构件 R 动态地演化到 R 的规约为

$$R \stackrel{def}{=} !(T_{RTE}e(r_C:t_{r_C}).\overline{R}\langle r_C\rangle)$$

$$R' \stackrel{def}{=} (T_{RTE}e(r_C:t_{r_C}).\overline{R}\langle r_C\rangle + T_{REQ}\overline{r_C}\langle r_I\rangle.(T_{RTE}e(r_C:t_{r_C}).\overline{R'}\langle r_C\rangle + !(T_{INV}\overline{r_I}\langle a_1,\cdots,a_s,r_R\rangle).(T_{RTE}e(r_C:t_{r_C}).\overline{R'}\langle r_C\rangle + T_{RES}r_R(y).0)))$$

它表示通过执行 $T_{RTE}e(r_C:t_{r_C}).\overline{R}\langle r_C\rangle$ 将导致交互通道的重新建立(见 $T_{REQ}\overline{r_C}\langle r_I\rangle$),或在交互通道不变的情况下导致重新请求服务(见 $T_{INV}\overline{r_I}\langle a_1,\cdots,a_s,r_R\rangle$),或导致重新接收服务结果(见 $T_{RES}r_R(y)$).

根据命题 1 和 3,结合端口运行时环境,可得到服务请求端口和提供端口的动态演化迁移规则:

规则 1(服务请求端口动态演化迁移规则)　通过执行运行时环境 $T_{RTE}e(r'_C:t'_C)$ 实现服务请求端口 $r_C:t_C$ 到 $r'_C:t'_C$ 的动态演化:

$$[REPL_{REQ\text{-}PORT}]\frac{\Gamma \vdash r_C:_c t_c \quad \Gamma \vdash r_C:_p T_{REQ}}{\Gamma[T_{RTE}e(r'_C:t'_C)] \vdash r'_C:_p t'_C}\langle t_C \leq t'_C$$

规则 2(服务提供端口动态演化迁移规则)　通过执行运行时环境 $T_{RTE}e(s'_C:t'_C)$ 实现服务提供端口 $s_C:t_C$ 到 $s'_C:t'_C$ 的动态演化:

$$[REPL_{PRO\text{-}PORT}]\frac{\Gamma \vdash s_C:_c t_c \quad \Gamma \vdash s_C:_p T_{SER}}{\Gamma[T_{RTE}e(s'_C:t'_C)] \vdash s'_C:_p t'_C}\langle t'_C \leq t_C$$

由一致性演化的含义知,基于这两个端口动态演化规则进行的演化都是一致性演化.

命题 5(端口动态演化能保持一致性)　基于规则 $[REPL_{REQ\text{-}PORT}]$ 和 $[REPL_{PRO\text{-}PORT}]$ 的端口动态演化是一致性演化.即对 $\Gamma \vdash S$,根据这两个规则的任一规则从 S 演化到 S',有 $\Gamma \vdash S'$.

证明:类似于命题 1 和 3 的证明.

在命题 2 和 4 的基础上,可以类似地得到关于构件的动态演化规则.

4.3　非一致演化

随着新技术的采用和系统环境的变化,构件演化会影响系统的整体行为,使系统产生不一致的情况.这时,必须分析构件演化对系统中其它构件的影响,以便对其它构件进行相应的调整,从而确保系统的一致性.

构件通过它们之间的相互连接发生交互,要分析非一致演化对系统的影响,就必须根据相互连接的构件形成的网络来分析构件之间的依赖关系.由构件的交互模型知,构件的服务提供端口可能依赖于它自身的服务请求端口,而服务请求端口又依赖于其它构件的服务提供端口.这种依赖关系具有传递性,因此当一个构件演化时,可以通过计算端口之间的依赖闭包来找出可能受影响的所有构件.然后,对受影响的各个构件进行相应的调整,即可将系统恢复到一致状态.

π 演算中的流图[9](*flow graph*)描述相互连接的进程的空间结构,它可以用来表示由构件之间的相互连接形成的网络.同时,考虑构件自身端口之间的内部依赖关系,可以得到构件的依赖图.

定义 5(流图)　流图中的节点由端口 $r_C, r_I, r_R, , s_C, s_I, s_R$ 组成,有向边由连接 (r_C, s_C),(r_I, s_I),(r_R, s_R) 组成.有向边

(r,s) 的方向由 r 指向 s,表示 r 依赖于 s.

定义 6(依赖图) 依赖图是用构件内部依赖 (s,r) 对流图进行扩展而得到的图.

由依赖图的定义知,依赖图中含有两种类型的边:(r,s) 代表构件之间的依赖;(s,r) 代表构件内部的依赖,描述了构件的服务提供端口对自身服务请求端口的依赖.

如果构件演化后产生的依赖图仍然具有良类型,则称这个依赖图是一致依赖图:

定义 7(一致依赖图) 若依赖图中所有边 (r,s) 都满足 $T_c(s) \leq T_c(r) \wedge T_p(r) \Leftrightarrow T_p(s)$,则依赖图是一致的.

如果构件演化产生了不一致的情况,则依赖图中的某条边不能保持良类型.通过计算该边的依赖闭包,可以得到受影响的所有边组成的影响图.

定义 8(边的影响图) 如果依赖图中某条边 (e_i,e_j) 不能保持良类型,则受该边影响的所有边组成的图是 $\{(e_i,e_j) \mid i \geq 2, j \geq 1, e_i$ 依赖于 $e_j\}$.

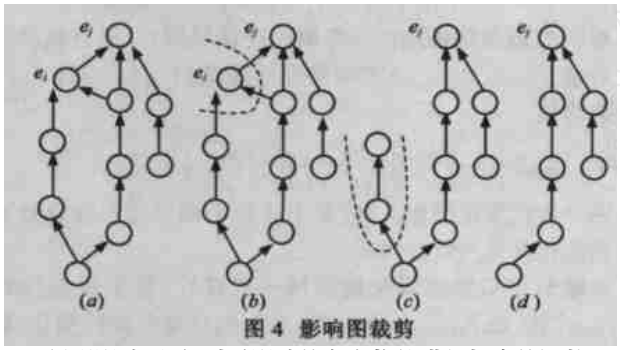

图 4 影响图裁剪

得到影响图后,对受影响的各个构件进行相应的调整,即可将系统恢复到一致状态.图 4(a)是边 (e_i,e_j) 的影响图示意结构.为了减少对影响图中节点的调整数量,我们提出如下的影响图启发式裁剪过程:

(1)对节点 e_j 进行调整,如果边 (e_i,e_j) 恢复良类型状态,则转(2),否则转(4).

(2)删除节点 e_i 以及与 e_i 连接的所有边,见图 4(b),转(3).

(3)删除不存在指向 e_j 的有向通路的所有节点以及与它们连接的边,见图 4(c),对 e_i 的裁剪结果见图 4(d).

(4)对 e_j 的调整不能使边 (e_i,e_j) 恢复良类型时,则必须对 e_i 进行调整,其过程是(1)～(4)的反复递归过程.

5 结论

本文根据构件交互的特点,借鉴 π 演算中的类型系统和进程构造的方法,提出构件交互的类型系统和基于交互的构件模型.为确保构件端口和交互通道的正确行为,给出构件服务的端口类型和通道类型.同时,结合 π 演算中良类型的思想,给出一致性构件系统的定义.以此为基础,对构件静态演化和动态演化进行了讨论,这两种演化能保证系统的一致性.最后,分析了不能保证系统一致性的非一致演化的影响,并提出恢复系统一致性的方法.该研究为基于构件的软件系统的维护和正确性演化奠定了基础.

参考文献:

[1] Szyperski C. Component Software. Beyond Object-Oriented Programming[M]. second edition. Harlow: Addison-Wesley, 2002.

[2] 杨芙清,梅宏,李克勤,袁望洪,吴穹. 支持构件复用的青鸟 III 型系统概述[J]. 计算机科学,1999,26(5):50 - 55.

[3] Lumpe M, Achermamm F, Nierstrasz O. A formal language for composition[A]. Leavens GT, Sitaraman M. Foundations of Component-based Systems[C]. Cambridge: Cambridge University Press. 2000. 69 - 90.

[4] Lumpe M. A -calculus based approach for software composition [D]. Bern: Universität Bern, Institut für Informatic und angewandte Mathematic, 1999.

[5] Sangiorgi D, Walker D. The -Calculus. A theofy of Mobile Processes [M]. Cambridge: Cambridge University Press, 2001.

[6] Szyperski C. Component technology what, where, and how[A]. Proc of the 25th International Conference on Software Engineering[C]. Washington: IEEE Computer Society Press, 2003. 684 - 693.

[7] Vigder M, Dean J. Building maintainable COTS-based systems[A]. Proc of the International Conference on Software Maintenance[C]. Washington: IEEE Computer Society Press, 1998. 132 - 138.

[8] Pahl C. A Pi-calculus based framework for the composition and replacement of components[A]. Giannakopoulou D, Leavens GT, Sitaraman M. Proc. of OOPSLA Workshop on specification and verification of component-based systems[C]. Iowa: Iowa State University, 2001. 97 - 106.

[9] Milner R. Communicating and Mobile Systems: the -Calculus[M]. Cambridge: Cambridge University Press, 1999.

作者简介:

龚洪泉 男,1976 年生于四川安岳,博士生,主要研究领域为软件复用,软件体系结构. E-mail: 021021080@fudan.edu.cn.

赵文耘 男,1964 年生于江苏常熟,教授,博士生导师,主要研究方向为软件复用,软件体系结构.