基于高阶π演算的构件演化行为研究

何海洋 李 强 余 祥 韩翔宇
(电子工程学院 合肥 230037)

摘 要 用形式化的方法分析软件演化过程中构件的行为已成为目前构件开发中的研究热点。为了满足构件演化过程中行为建模形式化的需要,建立了一种基于高阶π演算的构件演化行为分析方法。对构件演化行为进行分类,将顺序图表示的演化请求转换成高阶π演算进程表达式,通过高阶π演算的语法语义和等价理论等数学基础对演化行为进行推演,并检测演化中是否存在死锁问题。最后通过实例对该方法进行分析说明,从而验证了该方法的可行性和有效性。

关键词 构件演化,演化行为,高阶π演算

中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.03.043

Research on Component Evolving Behavior Based on High-order π Calculus

HE Hai-yang LI Qiang YU Xiang HAN Xiang-yu
(Electronic Engineering Institute, Hefei 230037, China)

Abstract Using formal method to analyze the behaviors of component in the software evolution has become a research hotspot now. According to the requirement of the formal method supporting the component evolving behavior, a formal method for component evolving behavior, which is based on High-order π calculus, was proposed. It will classify the component evolving behaviors and transform the sequence diagram described evolution requirements into High-order π expressions. Based on the rule and equivalence of High-order π calculus, the evolving behaviors are computed and the evolving conflicts are detected. At last, this paper used an example to prove the feasibility and availability of this method.

Keywords Component evolution, Evolving behavior, High-order π calculus

1 引言

随着软件多样性的发展,软件需要不断演化以应对随时变更的用户需求和环境。特别是安全类软件,其面临的安全问题每时每刻都在发生变化,如果不具备演化能力,则无力应对新出现的安全问题,不能根据新情况及时做出调整,即失去了其安全保障的作用。例如国家信息安全漏洞共享平台(CNVD)公布的漏洞中新的漏洞出现速度较快,如果不能及时对新增漏洞进行处理,将针对性的方法加入系统中,则很有可能无法检测该漏洞的存在,更有甚者被恶意攻击与反利用,因此软件演化具有重要的现实意义。软件演化是指软件为了适应外界环境变化、用户需求变更、技术手段革新以及成本等诸多因素,不断进行自我调整和更新的一个过程[1]。构件技术是继面向对象技术后提高软件开发效率、降低软件成本和提高软件质量的有效技术手段。组成系统的构件之间具有耦合度低、相互之间的影响小等特点,使得运用构件技术对演化进行研究成为一种有效的方法[2]。对构件演化的研究主要从构件演化行为的形式化分析、验证等方面展开[3]。演化行为是指演化过程中的软件行为,是由一系列有序动作构成的集合[4],在构件演化中主要表现为消息的发送、接收行为,并行、选择、循环等结构行为以及对组成系统成员构件的增加、删除、修改和替换等行为操作上。高阶π演算[5]是并行计算理论领域中重要的形式化理论,是对系统行为建模和分析验证的有效形式化工具,具有良好的数学基础和严格的推理逻辑。其数学结构简单,但表达能力强,在形式化描述和分析并发、异步、非确定以及分布式系统行为和动态验证等方面具有天然的优势[6],高阶π演算的特性非常适合对构件演化行为进行形式化分析。

针对构件演化行为,文献[7]利用状态图表示交互构件的状态,利用π演算对演化中的构件关键行为进行建模分析。文献[8]建立了基于事件轨迹的构件行为建模方法,通过构件生命周期内发生的交互事件构成的集合来描述构件行为,但该方法没有演化行为推演分析的过程。文献[9]将构件行为分为静态行为和动态行为,对行为间的控制相关、一致相关、交互相关和互斥相关等关系进行检查,但缺少对死锁问题的讨论。文献[10]基于Petri网直观的图形变化描述演化行为

到稿日期:2016-03-23 返修日期:2016-03-03 本文受技术基础项目(72141022),电子工程学院科研基金资助项目(KY141623)资助。
何海洋(1990—),男,硕士生,主要研究方向为软件工程、信息安全,E-mail:hehyflying@163.com;李 强(1962—),男,教授,硕士生导师,主要研究方向为软件工程、信息安全;余 祥(1986—),男,讲师,主要研究方向为软件工程、信息安全;韩翔宇(1991—),男,硕士生,主要研究方向为软件工程、信息安全。

的变迁和演化情况，但缺少对演化行为死锁问题的验证。文献[11]提出了以构件实体、构件行为和构件行为属性构成的CBO方法(构件行为本体论方法)和CIAM模型(构件交互适应性模型)，检测并去除演化中不匹配的构件行为，从而保证演化的正确进行。文献[12,13]提出了一种基于行为描述语言的需求模型建模方法，分析了构件之间的行为和行为关系，设计了由行为转换成图形的转化算法，根据映射关系和转换算法将行为图形化，从而建立了需求演化变更的可视化方法，对构件演化中行为的划分和分析具有指导作用。

针对这些方法的不足，为了加强构件演化行为的形式化分析推演和验证，本文以构件组成的软件系统为研究对象，从演化过程中的构件演化行为入手，利用高阶 π 演算的形式化理论方法对构件演化行为进行分析推演，确保演化的正确性。

2　演化请求处理

演化请求是对演化目的、意图以及具体过程的详细描述，可以是用户根据软件演化需求及软件使用过程中存在的问题等实际需求提出，也可以是软件设计开发人员因设计、功能等调整或软件维护等需要而提出。目前，关于演化请求的处理是基于自然语言的描述，容易由于描述模糊而出现二义性等问题，不利于对演化行为进行分析。顺序图是一种用于描述对象间交互的图，清晰地描述了系统中各角色间消息传递的顺序关系，跨越了多个对象，展示了系统的控制流程，把系统动态交互的情况描述得非常清楚。为了使演化请求的描述更加简单明了、清晰明白，结合构件演化过程中交互频繁的特点，本文利用 UML 中的顺序图对演化请求进行建模，将演化请求转化成与之对应的顺序图，从而利用顺序图满足工程化标准的特点，使演化请求简单易懂，便于处理。

例如，用户要求漏洞管理平台的漏洞库中增加一个新的漏洞基本信息构件，通过自然语言描述该演化请求的详细过程：用户向演化控制构件发送请求创建一个漏洞 A 的基本信息构件的消息，演化控制构件接收到该消息后，首先检查系统中是否已经存在该漏洞的基本信息构件，返回给用户一个创建基本信息构件的响应消息，并创建对应的新构件。利用 UML 顺序图对该演化请求进行描述，如图 1 所示。

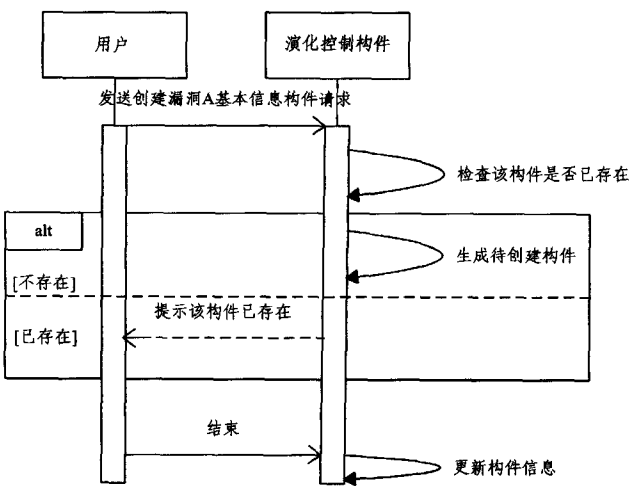

图 1　演化请求过程

从图 1 中可以看出，顺序图主要由参与交互的对象和对象之间消息的传递组成，是对特定场景中参与对象间的消息传递以及消息发生顺序的描述。图中构件之间的交互消息包括“发送创建漏洞 A 的基本信息构件请求”、“提示该构件已存在”、“结束”，而“检查该构件是否存在”、“生成待创建构件”和“更新构件信息”则看作演化控制构件的内部行为。图中的 alt 组合片段表示的是选择分支，类似的常用的组合片段还有并行 par、循环 loop 等控制逻辑，这些组合片段在描述演化请求的复杂结构时起着重要作用。

通过顺序图对演化请求进行描述后，不管是用户还是设计开发人员都能够清楚地明白演化意图，明确演化目的以及演化过程中构件之间的交互行为。

3　基于高阶 π 演算的演化行为分析

构件演化行为是演化过程中的基础，演化目的的实现是通过演化行为的有序执行完成的。通过顺序图对演化请求进行描述是为了将演化目的和演化过程中的交互行为进行直观的图形化表达。在此基础上，为了更加精确地分析演化过程中构件的交互行为，基于高阶 π 演算对构件演化行为进行分类，并结合顺序图将其中的消息和交互用高阶 π 演算表达式进行描述，从而得到行为的形式化表达，进而利用高阶 π 演算的演化规则和相关理论对行为进行推演分析。

3.1　演化行为分类和形式化表达

根据构件演化行为特征，将构件演化行为分成原子行为、结构行为和操作行为 3 类，其中原子行为是不可再分的最小单元，包含发送、接收消息等基本操作；结构行为具有一定的结构特征，代表并行、循环、选择和条件等基本结构；操作行为包括增加、删除、修改、替换构件等行为，由原子行为和基本的结构行为组成构件演化过程中的操作行为。

3.1.1　原子行为

原子行为是指演化过程中发送、接收消息的一类行为，在顺序图中表现为对象之间相互进行的发送和接收消息的过程。原子行为是构件演化行为中最基本的要素，通过利用高阶 π 演算，建立对原子行为的形式化表达式，实现利用高阶 π 演算描述原子行为，为演化行为分析奠定基础。

(1)不带参数的消息发送和接收行为的高阶 π 演算表达式。

设 ComProcSend 和 ComProcRecv 分别表示发送和接收消息的构件对象，msg 表示输入输出信道，由 ComProcSend 向 ComProcRecv 发送消息，msg 为交互的消息名，则发送消息行为的高阶 π 演算描述为：

$$send \stackrel{\text{def}}{=} \overline{msg}$$

接收消息的行为用高阶 π 演算描述为：

$$recv \stackrel{\text{def}}{=} msg$$

(2)带参数消息的发送和接收行为的高阶 π 演算表达式。

设 ComProcSend 和 ComProcRecv 分别表示发送和接收消息的构件进程，由 ComProcSend 通过消息名 msg 向 ComProcRecv 发送消息，$x_1, x_2, \cdots, x_n$ 为发送消息的具体内容，$y_1, y_2, \cdots, y_n$ 为接收的消息参数列表，则发送带参数消息行

为的高阶 π 演算描述为：

$send(x_1,x_2,\cdots,x_n)\ \underline{\underline{\text{def}}}\ \overline{msg}\langle x_1,x_2,\cdots,x_n\rangle$

接收带参数消息的行为用高阶 π 演算描述为：

$recv(y_1,y_2,\cdots,y_n)\ \underline{\underline{\text{def}}}\ msg(y_1,y_2,\cdots,y_n)$

两构件间消息的传递如图 2 所示。

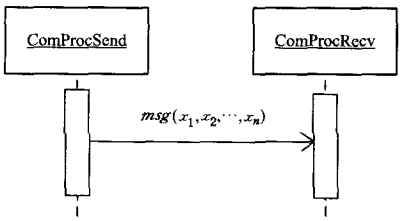

图 2 两构件间消息传递顺序图

用 $P1$，$P2$ 分别表示 ComProcSend 和 ComProcRecv 的进程，则根据上述规则，通过高阶 π 演算形式化描述发送消息和接收消息：

ComProcSend $\underline{\underline{\text{def}}}\ \overline{msg}\langle x_1,x_2,\cdots,x_n\rangle P1$

ComProcRecv $\underline{\underline{\text{def}}}\ msg(y_1,y_2,\cdots,y_n)P2$

3.1.2 结构行为

演化过程中，由于演化的需求是多样的，不仅存在简单的发送和接收消息此等行为，往往还有更加复杂的带有并行、顺序、选择、循环等结构特征的演化行为，这类演化行为称为结构行为。例如，通常的演化过程需要构件之间按照固定的步骤通过消息的发送和接收进行交互，从而完成演化流程，而有的演化则要求有不同的分支或者流程结构，通过不同的选择和条件的判断，执行不同的演化流程或重复某项行为。为了描述和分析这些复杂的演化逻辑，需要定义结构行为来描述不同的逻辑流程。结构行为拥有明显的结构特征，主要包括并行结构行为、选择结构行为和循环结构行为等。在顺序图中通过顺序图中的组合片段能够表示出这些结构行为，同样利用高阶 π 演算可以对演化中的结构行为进行形式化表达，实现对更加复杂的演化过程的形式化描述和建模。

(1)并行结构行为的高阶 π 演算表达式。

并行结构行为表示在演化过程中，满足约束的各个行为同时处于执行的状态，它们可以独立与其它进程进行交互，也可以相互之间进行通信。在顺序图中，可以通过 par 组合片段表示并发消息的传送，如图 3 所示，发送消息的构件 ComProcSend 向接收消息的构件 ComProcRecv 并行发送了两个分别为 $msg1(x_1,x_2,\cdots,x_n)$ 和 $msg2(x_1,x_2,\cdots,x_n)$ 的消息。通过利用高阶 π 演算中的并行算子可以对并行结构行为进行形式化建模。

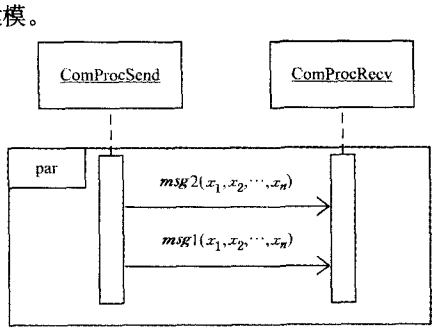

图 3 并行结构行为顺序图

并行结构行为的高阶 π 演算表示为：

$par\ \underline{\underline{\text{def}}}(\overline{msg1}\langle x_1,\cdots,x_n\rangle|\overline{msg2}\langle x_1,\cdots,x_n\rangle)P1$

(2)选择结构行为的高阶 π 演算表达式。

选择结构行为表示在演化过程中，当执行到某一部分时，需要根据条件约束有选择地执行特定行为，在高阶 π 演算中则对应为选择不同的后续进程，通过利用高阶 π 演算中的选择算子结合匹配算子可以对并行结构行为进行形式化建模。选择结构行为分为确定选择结构行为(alternative)和非确定选择结构行为(switch)，其中确定选择结构行为描述的是选择条件不是为真就是为假的情况，非确定选择结构行为描述的是选择条件为多种时的处理行为。

1) 确定选择结构行为的高阶 π 演算表达式。

当选择的条件只有为真和为假两种情况时，直接通过 alt 确定在为真条件和为假条件下分别选择执行的行为。设在顺序图中通过 alt 组合片段对选择结构行为进行描述，确定选择时的情况如图 4 所示。

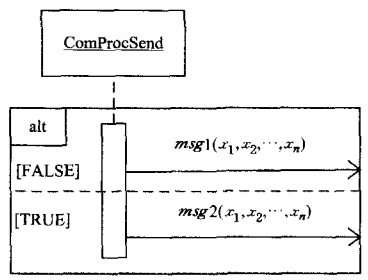

图 4 确定选择结构行为顺序图

$C(x)$为行为接收的布尔条件，当满足 FALSE 时，发送消息 $msg1(x_1,x_2,\cdots,x_n)$，执行 $P1$；当满足 TRUE 条件时，发送消息 $msg2(x_1,x_2,\cdots,x_n)$，执行 $P2$。则双向选择结构行为的高阶 π 演算表示为：

$alt\ \underline{\underline{\text{def}}}\ C(x).([x=\text{true}]P1+[x=\text{false}]P2)$

2)非确定选择结构行为的高阶 π 演算表达式。

当选择的条件是多个时，需要用到 switch 选择结构对输入的多种条件进行判断，并选择性地执行正确的行为。设在顺序图中通过 alt 组合片段对选择结构行为进行描述，多个条件选择时的情况如图 5 所示。

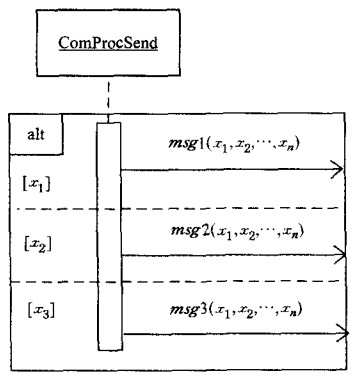

图 5 非确定选择结构行为顺序图

用 $C(x)$表示行为接收的行为约束条件，当 $x=x_i$ 满足时才能够执行行为进程 Pi。则用高阶 π 演算表示 switch 选择的表达式为：

switch $\underline{\text{def}}\, C(x).([x=x_1]P1+[x=x_2]P2+\cdots+[x=x_n]Pn)$

(3)循环结构行为的高阶 π 演算表达式。

循环结构行为表示在演化过程中，当满足一定的条件时，需要多次重复执行的行为，通过利用高阶 π 演算中的重复算子结合匹配算子、空进程可以对循环结构行为进行形式化建模。在顺序图中通过 loop 组合片段对选择结构行为进行描述，循环时的情况如图 6 所示。

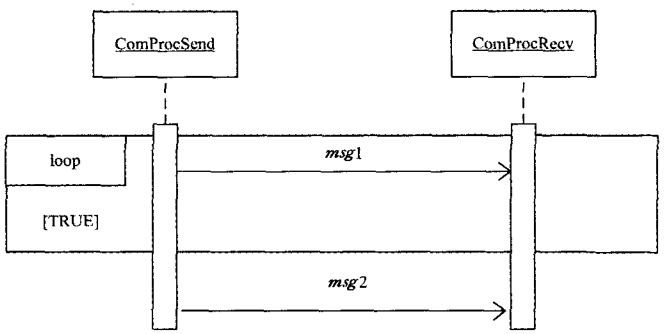

图 6　循环结构行为顺序图

设输入的条件表示为 $C(x)$，循环执行的是 $x=$true，执行发送消息 $msg1$ 的进程 $loop$，跳出循环后执行发送消息 $msg2$ 的进程 $P2$，则循环结构的高阶 π 演算表达式为：

$loop\ \underline{\text{def}}\, C(x).([x=\text{true}]!loop+[x=\text{false}]loop.0.P2)$

其中，loop.0 代表循环进程演化为空进程，表示了循环的结束。

3.1.3　操作行为

基于构件的演化过程具有其特有的行为特征。由于基于构件的软件系统是通过各个功能构件连接、组装而成，在演化过程中，整个构件系统通过改变构件及构件之间的连接关系即可达到演化的目的，而这种改变可以视为构件的增加、删除、修改和替换这些演化行为作用下的结果。通过增加新功能构件、删除不再需要的构件、修改现有构件使其满足新的环境和需求以及对无法修改的构件直接用其他构件替换等行为就能够实现并满足构件演化的需要。在原子行为和结构行为的基础上，定义了演化过程中的操作行为：增加、删除、修改、替换构件等行为，由于替换构件的行为可以通过增加和删除构件实现，因此这里只对增加、删除和修改构件行为进行描述。

(1)增加构件行为

增加构件行为是将一个根据需要定制的含有新功能的构件加入到原有的构件系统中，演化成为新的构件系统。增加构件的行为模型的简单示意图如图 7 所示。其中，原有构件系统中有两个构件，现在需要将构件 R 加入系统中，ε 和 μ 分别是构件 P 和 Q 以及 Q 和 R 之间的连接通道。

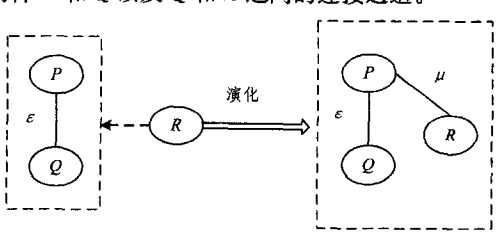

图 7　增加构件行为

用高阶 π 演算将进程 P 和进程 Q 形式化描述：

$P\ \underline{\text{def}}\, \bar{\varepsilon}\langle Create\rangle.\varepsilon(msg).[msg=CreateAck]\varepsilon(val).P$

$Q\ \underline{\text{def}}\, \varepsilon(msg1)[msg1=Create]\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(R,\mu)\rangle.Q$

则原有构件组成表示为 $P|Q$，现在由于演化需要，需要增加一个构件 R，从而演化成 $P|Q|R$，增加构件行为演化的具体过程为：

$P|Q\ \underline{\text{def}}\, \bar{\varepsilon}\langle Create\rangle.\varepsilon(msg).[msg=CreateAck]$

$\varepsilon(val).P|\varepsilon(msg1)[msg1=Create]$

$\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(R,\mu)\rangle.Q$

$\rightarrow\varepsilon(msg).[msg=CreateAck]\varepsilon(val).P$

$|\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(R,\mu)\rangle.Q$

$\rightarrow\varepsilon(val).P|\bar{\varepsilon}\langle New(R,\mu)\rangle.Q$

$\rightarrow P[New(R,\mu)/val]|Q$

$\rightarrow P|R|Q$

其演化含义是若进程 P 想要创建进程 R，则向 Q 发送创建请求消息 Create，当进程 Q 收到创建进程的消息时，返回一个确认创建的消息 CreateAck 并将代表新构件的进程 R 及其与外界交互的通道作为消息发送给进程 P，进程 P 通过获取新增的进程 R 及其通道 μ，建立连接，形成演化后的系统。

从最终的演化结果可以看出，构件系统由最初的 P 和 Q 组成演化成了由 P，Q，R 三者组成。系统中增加了新的代表构件的进程 R，实现了向系统增加新功能构件的演化行为操作。

(2)删除构件行为

删除构件行为是将一个功能淘汰的构件或不适应新应用场景的构件从构件系统中删除，从而保证系统的简洁高效。删除构件的行为模型的示意图如图 8 所示。原系统由 P，Q，R 3 个构件相连，参与删除演化行为的构件是构件 P，Q。通过高阶 π 演算将其映射为对应的进程 P 和 Q。

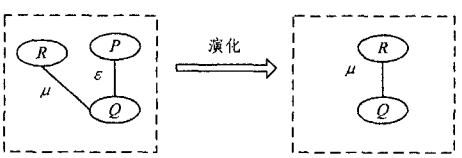

图 8　删除构件行为

用高阶 π 演算对进程 P，Q 进行形式化描述：

$P\ \underline{\text{def}}\, \bar{\varepsilon}\langle Delete\rangle.\varepsilon(msg).[msg=DeleteAck].0$

$Q\ \underline{\text{def}}\, \varepsilon(msg1)[msg1=Delete]\bar{\varepsilon}\langle DeleteAck\rangle.Q$

则原有构件组成表示为 $P|Q$，现在由于演化需要，需删除构件 P，从而演化成 Q，删除行为演化的具体过程：

$P|Q\ \underline{\text{def}}\, \bar{\varepsilon}\langle Delete\rangle.\varepsilon(msg).[msg=DeleteAck].0$

$|\varepsilon(msg1)[msg1=Delete]\bar{\varepsilon}\langle DeleteAck\rangle.Q$

$\rightarrow\varepsilon(msg).[msg=DeleteAck].0$

$|\bar{\varepsilon}\langle DeleteAck\rangle.Q$

$\rightarrow 0|Q$

$\rightarrow Q$

其演化含义：进程 P 通过通道 ε 向进程 Q 发送删除请求消息 Delete，进程 Q 在接收进程 P 的删除请求后，发送响应

删除请求的消息 DeleteAck，进程 P 接收到确定删除的消息后，演化成空进程(用 0 进程表示)，不再具有任何功能。

当删除构件的演化行为执行完毕后，构件系统只剩下进程 Q，表示只有构件 Q 存在，而构件 P 则已经被删除，从而达到了删除冗余构件的目的。

(3)修改构件行为

修改构件行为是指在构件演化过程中根据需要修改构件之间的连接关系，将一个功能构件或子构件加入到另一个功能构件或复合构件中，从而达到调整构件组织结构的演化需求，而不是修改构件的内部内容。修改构件的行为模型的示意图如图 9 所示。原系统中有 P,Q,R 3 个构件，通过高阶 π 演算将其映射为对应的进程 P,Q,R。由于演化需要，进程 Q 与 R 不再连接，而进程 P 需要与进程 R 连接。

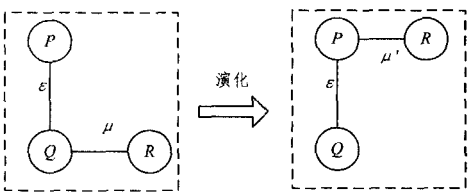

图 9 修改构件行为

用高阶 π 演算对进程 P,Q,R 进行形式化描述：

$P \overset{\text{def}}{=} \bar{\varepsilon}\langle Modify\rangle.\varepsilon(msg).[msg=ModifyAck]\varepsilon(val).P$

$Q \overset{\text{def}}{=} \varepsilon(msg1)[msg1=Modify]\bar{\varepsilon}\langle ModifyAck\rangle.\bar{\varepsilon}\langle\mu'\rangle.Q$

$R \overset{\text{def}}{=} R$

则修改行为演化的具体过程：

$P|Q|R \overset{\text{def}}{=} \bar{\varepsilon}\langle Modify\rangle.\varepsilon(msg).[msg=ModifyAck]\varepsilon(val).P$

$|\varepsilon(msg1)[msg1=Modify]\bar{\varepsilon}\langle ModifyAck\rangle.\bar{\varepsilon}\langle\mu'\rangle.Q|R$

$\rightarrow\varepsilon(msg).[msg=ModifyAck]\varepsilon(val).P$

$|\bar{\varepsilon}\langle ModifyAck\rangle.\bar{\varepsilon}\langle\mu'\rangle.Q|R$

$\rightarrow\varepsilon(val).P|\bar{\varepsilon}\langle\mu'\rangle.Q|R$

$\rightarrow P[\mu'/val]|Q|R$

$\rightarrow P|Q|R$

演化的具体过程：进程 P 通过通道 ε 向进程 Q 发送修改请求消息 Modify，当进程 Q 收到修改消息时，返回一个确认创建的消息 ModifyAck，并将其与构件 R 之间的连接通道 μ 作为消息发送给进程 P，进程 P 通过获取进程 R 与外界交互的通道 μ 与进程 R 建立连接，从而达到演化的目的，形成演化后的系统。

当修改构件的演化行为执行完后，构件系统外部结构形式不变，但是其内在构件之间的连接关系已被修改。这种连接关系的修改为设计者优化软件系统结构、调整构件布局提供方便。

3.2 演化行为死锁问题分析

在演化推演过程中，可能会由于演化条件不满足使得演化不能继续执行和演化条件恒为真造成死循环等死锁情况，从而影响演化的正常进行，为了避免死锁的发生，对死锁问题进行分析。将死锁问题分为两类：1)循环条件不正确造成的死循环死锁问题；2)输入条件不成立形成的空等待死锁问题。

3.2.1 循环死锁问题

循环死锁问题是指在演化流程结构中含有循环等结构行为时，容易产生由于条件不正确导致某一部分演化行为持续执行的情况，这种死循环问题长期占用执行资源，极易造成计算机资源的浪费。假设一个演化流程中的部分交互示意图如图 10 所示。

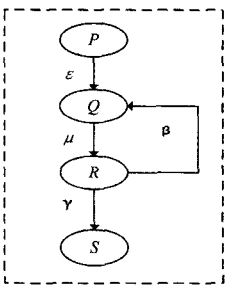

图 10 包含循环死锁的演化示意图

用高阶 π 演算将图中各构件进程及行为形式化表示为：

$P \overset{\text{def}}{=} \tau_P.\bar{\varepsilon}\langle msg\rangle.P$

$Q \overset{\text{def}}{=} (\varepsilon(msg)|\beta(msg1)).\bar{\mu}\langle msg2\rangle.Q$

$R \overset{\text{def}}{=} \mu(msg2).[x=\text{true}]\bar{\beta}\langle msg1\rangle.\bar{\gamma}\langle msg3\rangle.R$

$S \overset{\text{def}}{=} \gamma(msg3).S.0$

则演化的具体过程为：

$P|Q|R|S \overset{\text{def}}{=} \tau_P.\bar{\varepsilon}\langle msg\rangle.P|(\varepsilon(msg)|\beta(msg1)).\bar{\mu}\langle msg2\rangle.Q$

$|\mu(msg2).[x=\text{true}]\bar{\beta}\langle msg1\rangle\rangle.\bar{\gamma}\langle msg3\rangle.R|\gamma(msg3).S.0$

$\rightarrow P|\beta(msg1).\bar{\mu}\langle msg2\rangle.Q|\mu(msg2).[x=\text{true}]\bar{\beta}\langle msg1\rangle.\bar{\gamma}\langle msg3\rangle.R|\gamma(msg3).S.0$

当演化执行到这一步时，如果条件 x 为真恒成立，则循环不能结束，一直处于死循环状态，从而出现死锁问题。由于循环结构在演化过程中经常使用，如果不进行死锁分析，则演化很有可能出现问题。

3.2.2 等待死锁问题

等待死锁问题是指当某项条件不成立时，导致演化行为一直无法执行，处于等待条件成立的状态。等待死锁问题使系统在很长一段时间内无法响应操作，造成极差的使用效果。假设一个演化流程中的部分交互示意图如图 11 所示。

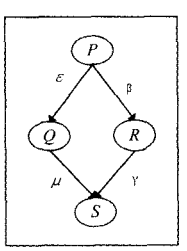

图 11 包含等待死锁的演化示意图

用高阶 π 演算将图中各构件进程及行为形式化表示为：

$P \overset{\text{def}}{=} \tau_P.(\bar{\varepsilon}\langle msg\rangle + \bar{\beta}\langle msg1\rangle).P$

$Q \overset{\text{def}}{=} \varepsilon(msg).\bar{\mu}\langle msg2\rangle.Q$

$R \overset{\text{def}}{=} \beta(msg1).\bar{\gamma}\langle msg3\rangle.R$

$S \overset{\text{def}}{=} \mu(msg2).\ \gamma(msg3).S.0$

则演化的具体过程为：

$P|Q|R|S \overset{\text{def}}{=} \tau_P.(\bar{\varepsilon}\langle msg\rangle + \bar{\beta}\langle msg1\rangle).P$

$|\varepsilon(msg).\bar{\mu}\langle msg2\rangle.Q$

$|\beta(msg1).\bar{\gamma}<msg3>.R|\mu(msg2).\ \gamma(msg3).S.0$

$\rightarrow P|\bar{\mu}\langle msg2\rangle.Q|\beta(msg1).\bar{\gamma}\langle msg3\rangle.R$

$|\mu(msg2).\ \gamma(msg3).S.0$

$\rightarrow P|\beta(msg1).\bar{\gamma}[msg3].R|\gamma(msg3).S.0$

在该过程中，进程 P 分别从 ε 和 μ 通道向进程 Q 和进程 R 发送消息 msg 和 $msg1$，发送消息 msg 的条件成立，而发送 $msg1$ 的条件一直不满足，导致的结果是进程 P 向 Q 发送消息，Q 在接收到消息后向进程 S 发送消息，而进程 R 一直未接收到消息，无法触发演化行为，导致进程 S 执行的条件不满足，一直处于等待状态。

通过利用高阶 π 演算对死锁问题进行形式化建模和分析，可以得出以下结论：进程表达式在演化推演过程中不能经过推演得到最简结构，且仍包含未执行行为的进程，则可能存在死锁问题。

4　实例分析

漏洞管理系统是安全领域的重要软件，由于当前漏洞增长的速度越来越快，漏洞形成的原理、出现的形式和攻击的手段越来越多样化，因此系统中针对漏洞的信息描述、扫描和攻击措施需要不断进行优化更新，这要求系统必须具有很强的演化性来应对这种变化。同时为了保证系统不因演化而产生不稳定及出现错误，必须对漏洞管理系统中的演化行为进行分析验证。因而系统中对漏洞构件的实时演化对于研究构件演化行为这一类问题具有很强的代表性。

漏洞管理系统将所有的漏洞信息进行构件化处理，建立了漏洞构件库，对当前出现的漏洞信息均建立基本的信息描述、扫描以及攻击等构件与之对应，并能够及时根据需要增加新的漏洞构件、删除无利用价值的漏洞构件以及对漏洞利用、攻击构件进行替换，加强利用力度，方便安全检测人员进行漏洞的查询、扫描和攻击。

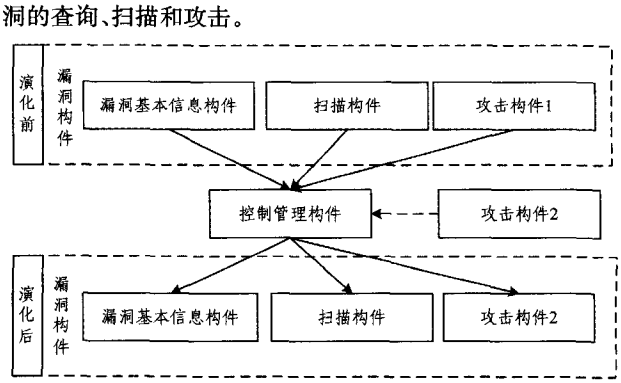

图 12　替换漏洞攻击构件过程

该系统中，每个漏洞构件由漏洞基本信息构件、漏洞扫描构件和漏洞攻击构件组成。随着漏洞的不断更新，需要对漏洞管理系统中的漏洞信息进行演化。现在，由于发现了 CVE 编号为“CVE-2016-0701”漏洞的新利用价值和攻击手段，需要对漏洞构件中的原有攻击构件进行替换演化行为操作。演化过程如图 12 所示。

为了利用高阶 π 演算对其演化行为进行分析，将替换漏洞攻击构件的演化过程简化表示，如图 13 所示。替换构件行为的实现原理是通过删除构件行为和增加构件行为联合完成，构件系统先通过删除构件行为，将待替换的原构件从构件系统中删除，在删除原构件之后，利用增加构件行为将新构件加入到构件系统中，从而完成构件的替换行为。该实例是用新的攻击构件 2 来替换原攻击构件 1。

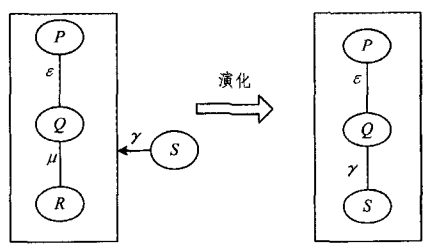

图 13　替换构件示意图

用高阶 π 演算对进程 P,Q,R 进行形式化描述：

$R \overset{\text{def}}{=} \bar{\mu}\langle Delete\rangle.\mu(msg).[msg=DeleteAck].0$

$Q \overset{\text{def}}{=} \varepsilon(msg1)[msg1=Delete]\bar{\varepsilon}\langle DeleteAck\rangle$

$.\bar{\varepsilon}\langle Create\rangle.\varepsilon(msg).\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q$

$P \overset{\text{def}}{=} \varepsilon(msg2)[msg2=Create]\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

则替换构件行为演化的具体过程为：

$R|Q|P \overset{\text{def}}{=} \bar{\mu}\langle Delete\rangle.\mu(msg).[msg=DeleteAck].0$

$|\varepsilon(msg1)[msg1=Delete]\bar{\varepsilon}\langle DeleteAck\rangle$

$\bar{\varepsilon}\langle Create\rangle.\varepsilon(msg).\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q$

$|\varepsilon(msg2)[msg2=Create]\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

$\rightarrow\mu(msg).[msg=DeleteAck].0$

$|\varepsilon(msg1)[msg1=Delete]\bar{\varepsilon}\langle DeleteAck\rangle$

$.\bar{\varepsilon}\langle Create\rangle.\varepsilon(msg).\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q$

$|\varepsilon(msg2)[msg2=Create]\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

$\rightarrow\mu(msg).[msg=DeleteAck].0$

$|\bar{\varepsilon}\langle DeleteAck\rangle.\bar{\varepsilon}\langle Create\rangle.\varepsilon\langle msg\rangle$

$.\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q$

$|\varepsilon(msg2)[msg2=Create]\bar{\varepsilon}<CreateAck>.\bar{\varepsilon}<New(S,\gamma)>.P$

$\rightarrow 0|\bar{\varepsilon}\langle Create\rangle.\varepsilon\langle msg\rangle.\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q|\varepsilon(msg2)[msg2=Create]\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

$\rightarrow\varepsilon\langle msg\rangle.\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q$

$|\varepsilon(msg2)[msg2=Create]\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

$\rightarrow\varepsilon\langle msg\rangle.\langle msg=CreateAck\rangle\varepsilon\langle val\rangle.Q$

$|\bar{\varepsilon}\langle CreateAck\rangle.\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

$\rightarrow\varepsilon\langle val\rangle.Q|\bar{\varepsilon}\langle New(S,\gamma)\rangle.P$

$\rightarrow Q[New(S,\gamma)/val]|P$

$\rightarrow Q|S|P$

该替换构件演化过程主要分为两项内容：1)删除进程 R；2)创建进程 S。

1)删除进程 R 的过程：进程 R 向进程 Q 发送删除自身的演化消息，进程 Q 收到删除消息后返回给进程 R 删除响应，进程 R 接收到响应消息后，演化成空进程 0，从而不再具有任何功能，被删除。

2)创建进程 S 的过程：进程 P 发送创建进程 S 的请求，接收到 Q 返回的响应消息后，将新建进程 S 与其内部通道作为消息发送给进程 Q，通过换名操作，完成进程 S 的创建。系统由最初的进程 R,Q,P 并行组合，演化为最终结果 S,Q,P 组成的系统，即由 $R|Q|P$ 演化成为 $Q|S|P$。

根据演化行为死锁问题发生的条件对此演化过程进行分析发现，在替换漏洞攻击构件的演化行为推演过程中，没有循环结构出现，因而不存在循环死锁问题；演化最终的结果形成了最简结构，且演化行为均执行结束，没有处于等待条件而未执行的演化行为不存在等待死锁问题。所以该替换漏洞攻击构件演化行为中不存在死锁问题，可以执行该演化行为操作。

结束语 本文研究了构件演化过程中的演化行为问题，提出了基于高阶 π 演算的构件演化行为分析方法。先利用顺序图对演化请求进行处理，刻画了演化过程中的行为交互，使演化请求直观明了，易于理解。利用高阶 π 演算与构件演化行为的相似性，将构件演化行为分为原子行为、结构性为和操作行为，建立演化行为的高阶 π 演算进程表达式。在此基础上，通过高阶 π 演算演化规则对演化行为进行推演验证，分析了演化中的死锁问题，描述了死锁出现的条件，为避免演化出现死锁和演化顺利执行奠定了基础。

下一步的研究工作包括：进一步研究构件演化行为，深入分析演化行为的特征，结合支持高阶 π 演算的工具对演化中的死锁问题进行自动化分析和判断。除此之外，在本文基础上，研究建立基于高阶 π 演算的构件演化模型，形成系统性、结构化的构件演化体系，为构件演化的自动化处理做准备。

参 考 文 献

[1] RAJLICH V. Software Evolution and Maintenance[C]// on Future of Software Engineering: ACM, 2014: 133-144.

[2] ZHANG H, ZHANG L, XU Q, et al. A formalized architecture-centric evolution process for component—based software system [C]// IEEE Intelligent Control and Automation. 2014: 3461-3466.

[3] XU J, XIAO G, ZHANG Y M, et al. Research on Component Behavior Fragment Extraction and Composition Based on Logical Reasoning[J]. Computer Science, 2012, 39(5): 120-123. (in Chinese)
徐俊，肖刚，张元鸣，等. 基于逻辑推理的构件行为片段提取与重组研究[J]. 计算机科学，2012，39(5)：120-123.

[4] SHEN L M, YANG H F, MA C, et al. Software Behavior Detection Method Based on Hierarchical Relation Analysis[J]. Journal of Chinese Computer Systems, 2014, 35(9): 1945-1955. (in Chinese)
申利民，杨慧锋，马川，等. 基于层次关系分析的一种软件行为检测方法[J]. 小型微型计算机系统，2014，35(9)：1949-1955.

[5] PIERARD A, SUMII E. A higher-order distributed calculus with name creation[C]// IEEE Symposium on Logic in Computer Science. 2012: 531-540.

[6] JEFFREY A, RATHKE J. Contextuale quivalence for higher-order picalculus revisited[J]. LMCS, 2005, 1: 1-4.

[7] WANG D, CHANG J S, ZHAO W B. Verification Model for Trustworthiness of Interaction between Software Components with Picaculus[J]. Journal of Frontiers of Computer Science and Technology, 2012, 6(5): 419-429. (in Chinese)
王丹，常建生，赵文兵. 使用 Pi 演算的构件交互可信性验证模型[J]. 计算机科学与探索，2012，6(5)：419-429.

[8] CHEN Z, ZHANG J, MIAO H, et al. A Formal Modeling For Component Behaviors Based on Event Track[Z]. 2010, 113-116.

[9] WEI W, TONG L. Component Behavior Modeling and Relativity Analysis[C]// International Conference on Intelligent Computing and Cognitive Informatics. 2010: 218-222.

[10] ZHENG F, HU H, JIAN L. Applying a Component Behavior Model to MVC Pattern [C] // International Conference for Young Computer Scientists, ICYCS 2008. Hunan, China, 2008: 1106-1111.

[11] HUACHENG Q I, RONG M, ZHANG G. A Behavior-Driven Model of Component Interaction Adaptation[J]. Computer Engineering & Applications, 2009, 45(21): 156-159.

[12] MA L, WU H G, WU G Q. Visualization of software behavior-oriented requirements model[J]. Application Research of Computers, 2015, 32(8): 2406-2409. (in Chinese)
马丽，吴怀广，毋国庆. 面向软件行为的需求模型可视化研究[J]. 计算机应用研究，2015，32(8)：2406-2409.

[13] MA L, WU G Q, HUANG B, et al. Visualization Method of BDL Model to UML State Diagram[J]. Computer Science, 2015, 42(7): 38-43. (in Chinese)
马丽，毋国庆，黄勃，等. BDL 模型到 UML 状态图的可视化方法研究[J]. 计算机科学，2015，42(7)：38-43.