Web服务编排描述语言WS-CDL的形式化模型框架*)

辜希武　卢正鼎
(华中科技大学计算机科学技术学院　武汉 430074)

摘　要　Web服务编排描述语言WS-CDL从全局的角度定义了一组Web服务之间的协作和交互必须遵守的规则。作为一个基于XML的描述性规范语言,WS-CDL缺乏形式化的模型和验证机制,难以保证协作和交互的正确性。本文针对WS-CDL规范提出了一个基于全局的形式化模型框架Abstract WS-CDL,包括语法、同构关系和操作语义,同时定义了一套从该模型框架到基于Pi-演算描述的局部模型的映射规则,最后通过案例分析给出了全局和局部2个层次的模型验证方法。

关键词　Web服务编排描述语言,Web服务,Web服务组合

A Formal Model Framework for Web Services Choreography Description Language

GU Xi-Wu　LU Zheng-Ding
(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　The Web Service Choreography Description Language(WS-CDL) defines a set of rules for the collaborations and interactions between a group of Web services from a global view. As a XML-based descriptive language, WS-CDL lacks a formal model and verification mechanism to guarantee the correctness of collaborations and interactions. This paper proposes a formal model framework Abstract WS-CDL for the WS-CDL specification that is based on global view, including syntax, structural congruence and operational semantics. The paper also defines a set of rules for mapping this global based formal model to local model depicted by Pi-calculus. At last the paper introduces the model verification methods at global level and local level through a case study.

Keywords　Web services choreography description language, Web service, Web service composition

1　引言

Web服务是一种分布在网络(通常是WWW)上、被基于XML的语言所描述、发布、查找和调用的自治的、异构的软件实体,它的出现带来了一种新型的分布式计算模式:面向服务的计算(Service Oriented Computing, SOC)。一个令人感兴趣的问题就是如何将已有的一些Web服务组合起来,以提供给用户新的、增值的服务。这些被组合的子Web服务会彼此并发地交互,以完成客户的请求,而彼此间的交互系通过通信和交换信息来完成。因此Web服务组合涉及到的问题包括被组合的服务间的并发、同步、通信。

基于工作流的Web服务组合方法可以分为2类[1,2]:基于编排(Choreography)和基于编制(Orchestration)。基于编排的组合方法从一个全局的角度定义每个参与者(子服务)之间的会话和协作,因而这种方法是面向全局的;基于编制的方法则是局部的,它从一个参与者的角度来定义该参与者如何与其它子服务交互。目前,基于编制的Web服务组合描述规范主要是Web服务商业流程执行语言[3](Business Process Execution Language for Web Services, BPEL4WS),基于编排的Web服务组合描述规范主要是Web服务编排描述语言[4](Web Services Choreography Description Language, WS-CDL)。这2种方法不是孤立的,而是位于2个不同层次上而被结合起来用于实现Web服务的组合。

为了保证Web服务组合的正确性,一个有效的方法是用一个形式化的模型来描述各个子服务间的交互行为,然后利用形式化的方法来验证模型,以检查子服务间的交互是否存在死锁、活锁,服务间的交互行为是否匹配等。WS-CDL和BPEL4WS都是基于XML的描述性语言,缺乏形式化的模型和验证机制,因此目前有很多研究者利用并发系统的形式化模型来建模、验证WS-CDL,以保证Web服务组合的正确性。Brogi, Canal, Pimentel等人[5]提出了一个基于CCS(Calculus of Communicating Systems)的Web服务编排接口[6](Web Service Choreography Interface, WSCI)的形式化模型。Foster, Uchitel, Magge等人[7]利用FSP(Finite State Process)来形式化描述WS-CDL,并且利用LTSA Labeled Transition System Analyzer)工具对FSP进程进行模型检查。W. L. Yeung, Ji Wang, Wei Dong[8]利用CSP(Communicating Sequential Processes)对用WS-CDL描述的Web服务进行形式化验证。Busi, Gorrieri, Guidi等人[9]定义了一个简单的形式化语言CL来描述WS-CDL,但是CL相对WS-CDL来说不完备,有些WS-CDL中行为在CL中不能描述,如WorkUnit。Gorrieri, Guidi, Lucchi[10]利用针对WS-CDL所定义的形式化语言CL分析WS-CDL中定义的三种交互行为的模式。

本文针对WS-CDL规范提出了一个形式化模型的框架

*)国家自然科学基金资助项目(项目编号:60403027)、湖北省自然科学基金(项目编号:2005ABA258)、软件工程重点实验室开放基金(项目编号:SKLSE05-07)。**辜希武**　博士研究生,研究方向为Web服务、语义Web和中间件;**卢正鼎**　教授、博士生导师,研究方向为分布式系统集成、Web数据管理、信息安全、数据库系统。

Abstract WS-CDL，该模型能描述 WS-CDL 规范中所定义的所有行为。该模型框架包括描述语法、同构关系和操作语义(即推演规则)。同时，本文定义了从该模型框架到基于 Pi-演算描述的 Orchestration 的映射规则。最后本文利用 Abstract WS-CDL 描述了一个具体案例，并通过案例给出了 Choreography 和 Orchestration 二个层次的模型验证方法。

本文包括以下内容：第 2 节总体介绍了 WS-CDL 规范；第 3 节给出了 WS-CDL 的形式化模型框架 Abstract WS-CDL；第 4 节定义了从该模型框架到基于 Pi-演算描述的 Orchestration 的映射规则；第 5 节通过一个具体案例给出了 Choreography 和 Orchestration 二个层次的模型验证方法；最后是全文总结和未来的工作。

2 WS-CDL 基本概念

WS-CDL 从全局的角度定义了一组 Web 服务之间相互交互的规则。WS-CDL 规范中的主要元素层次关系如图 1 所示。

Package 元素为一个 WS-CDL 文档的根元素。RoleType 元素定义了一个交互的参与者。ChannelType 元素定义了交互所用到的通道。Choreography 元素定义了一组角色间的交互规则，它可以看作一组行为(Activity)的容器，而这组 Activity 描述了角色间交互的动作。

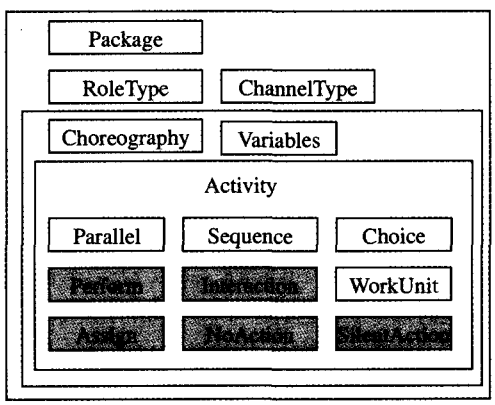

图 1 WS-CDL 规范主要元素层次关系

在 WS-CDL 规范里，行为分为三类：第一类为控制流行为，这类行为以结构化的方式把其包含的子行为组合起来。Parallel 描述了一组并行执行的子行为；Sequence 描述了一组顺序执行的子行为；Choice 描述了一组选择执行的子行为。第二类为 WorkUnit，描述了一组有条件的迭代执行的子行为，WorkUnit 元素的 guard 属性决定迭代是否执行，repeat 属性决定迭代终止的条件。第三类为基本交互行为(在图 1 中由灰色背景方框所示)。NoAction 描述一个角色不执行任何交互行为；SilentAction 描述一个角色执行一个内部行为，该行为不会对 Choreography 元素中包含的其它行为产生任何影响；Assign 描述了一个角色的变量间的赋值；Perform 描述了在当前正执行的 Choreography 的上下文环境中调用另外一个 Choreography，在调用过程中通过变量绑定机制传递变量值；Interaction 是 WS-CDL 规范里最为重要的基本交互行为，它描述了两个角色间三类信息的交换过程：request、response 和 request-response。

一个 request-response 类型的 Interaction 语法为(这里只列出主要的子元素)：

```
〈interaction name="NCName" channelVariable="QName"operation
    ="NCName" …〉
    〈participate relationshipType="QName"
            fromRoleTypeRef = " QName " toRoleTypeRef =
            "QName" …/〉
    〈exchange name="NCName" action="request" …〉
        〈send variable="XPath-expression" …/〉
        〈receive variable="XPath-expression" …/〉
    〈/exchange〉
    〈exchange name="NCName" action="respond" …〉
        〈send variable="XPath-expression" …/〉
        〈receive variable="XPath-expression" …/〉
    〈/exchange〉
〈/interaction〉
```

其中：fromRoleTypeRef 属性和 toRoleTypeRef 属性定义了交互的双方，channelVariable 属性定义了交互的通道。exchange 子元素定义了一次单向交互，其 action 属性定义了该单向交互是 request 还是 respond，其 send 和 receive 子元素定义了交互双方用于保存发送和接收信息的变量。对于 request 和 response 类型的 Interaction，其 exchange 子元素只能有一个，action 类型相应地为 request 和 response。

3 WS-CDL 的形式化模型 Abstract WS-CDL

3.1 基本概念的形式化描述

在 WS-CDL 中的各类 Activities 描述了 Web 服务(在 WS-CDL 规范里称之为角色)之间的相互交互和协作的动作。在 Abstract WS-CDL 中用一个会话 *Session* 来表示一个 Activity。对于一个 *Session*，其参与者(角色)的集合记为 $S[R]$。

一个角色(Role)包含的信息有：角色拥有的变量(代表角色当前的状态)、角色能访问的通道、角色上能观察到的外部操作。因此，对于一个 *Session*，其中一个参与者 $r \in S[R]$ 可以表示成四元组，记为 $r = \langle N, V, O, C\rangle$。其中：$N$ 为角色 r 的名称，V 为角色 r 所拥有的变量的集合，C 为角色 r 能访问的通道的集合，O 为角色 r 的外部可观察到的操作的集合。我们分别将对角色 r 所拥有的变量集合 V、能访问的通道集合 C、操作集合 O 的引用记为 $r[V]$、$r[C]$、$r[O]$。

角色间的交互通过通道(Channel)进行。在 WS-CDL 规范里，通道主要的属性有：通道名、通道上能发生的基本交互行为的类型和通道的位置。通道名唯一地标示了一个通道；通道上能发生的基本交互行为的类型有：request、respond、request-respond；通道的位置由角色指定，即一个通道一定位于一个角色上。注意，通道的定义是面向信息的接受者的：如果角色 r_a 和 r_b 通过通道 ch 交互，r_a 为信息的发送者，r_b 为信息的接受者，则通道 ch 被定义为位于角色 r_b，记为 $ch@r_b$。同时，角色 r_a 和 r_b 都可以访问 ch，因此 $ch \in r_a[C]$，$ch \in r_b[C]$。

WS-CDL 规范里的 WorkUnit 和 Choice 行为中都有条件判断，我们把这些条件都抽象为命题。注意，WS-CDL 规范是面向全局的，因此一个命题的真假会涉及到多个角色的变量。对于命题 p，我们将和它相关的角色的集合记为 $p[R]$。

最后，WS-CDL 规范中 Choreography 元素可以表示为一个三元组，记为

$$CHO = \langle \overline{S}, \overline{R}, \overline{V}\rangle$$

其中：$\overline{S}$ 为 Choreography 元素所包含的 *Session* 的集合，$\overline{R}$ 为 Choreography 元素所包含的角色的集合，$\overline{V}$ 为 Choreography 元素所包含的变量的集合。显然，$\overline{R} = \bigcup_{S_i \in \overline{S}} S_i[R]$，$\overline{V} = \bigcup_{R_i \in \overline{R}} R_i[V]$。

3.2 Abstract WS-CDL 的语法

为了方便形式化描述，首先引入下列记号：

(1)$r[V:v]$、$r[C:c]$、$r[O:o]$分别表示对角色所拥有的一个变量v、通道c、操作o的一个引用。

(2)$r[V:x]=e$表示角色r的变量集合V中的变量x被赋值为e，而其他变量不变。

(3)$r(o)$表示角色r的一个操作o被执行。

对一个 Session S，其形式化描述的语法如下：

$S::=S_{atom}\mid[p_1]S_1+[p_2]S_2\mid S_1\parallel S_2\mid S_1.S_2\mid[P_g][P_{rep}]^*S\mid NULL$

$S_{atom}::=S_{no}\mid S_{silent}\mid S_{assign}\mid S_{req}\mid S_{resp}\mid S_{req\text{-}resp}\mid S_{perform}$

$S_{no}::=r(0)$ $S_{silent}::=r(\tau)$ $S_{assign}::=assign(r[V:x],e)$

$S_{req}::=request(r_1,r_2,ch@r_2,r_1[V:x],r_2[V:y])$

$S_{resp}::=respond(r_1,r_2,ch@r_1,r_1[V:x],r_2[V:y])$

$S_{req\text{-}resp}::=req\text{-}resp(r_1,r_2,ch@r_2,r_1[V:x],r_2[V:y],r_1[V:s],r_2[V:t])$

$S_{perform}::=CHO(y_1,y_2,\cdots,y_n)$

$CHO(x_1,x_2,\cdots,x_n)\overset{def}{=}S$

其中：

(1)S_{atom}为原子不可分的会话。

(2)$[p_1]S_1+[p_2]S_2$表示会话的选择执行。当命题p_1为真时，会话S_1被执行；当命题p_2为真时，会话S_2被执行。

(3)$S_1\parallel S_2$代表2个并行执行的会话。

(4)$S_1.S_2$代表2个顺序执行的会话。

(5)$[P_g][P_{rep}]^*S$表示迭代执行的会话。如果命题P_g为真，会话S被执行；会话S执行完毕后如果命题P_{rep}为真，会话S被迭代执行直到命题p_{rep}为假。

(6)$NULL$代表一个空会话。

(7)S_{no}为角色r上的一个空行为，即不执行任何动作。

(8)S_{silent}为角色r上的一个哑行为，即对当前会话不产生任何影响的行为。

(9)S_{assign}为角色r上的赋值行为，角色r上的变量集合V中变量x被赋值为e。

(10)S_{req}为角色r_1和r_2间的请求(Request)交互行为。r_1通过通道$ch@r_2$向r_2发送请求$r_1[V:x]$，r_2收到该请求保存在$r_2[V:y]$。

(11)S_{resp}为角色r_1和r_2间的应答(Reponse)交互行为。r_2通过通道$ch@r_1$向r_1发送应答$r_2[V:y]$，r_1收到该请求保存在$r_1[V:x]$。

(12)$S_{req\text{-}resp}$为角色r_1和r_2间的请求-应答(Request- Reponse)交互行为。r_1通过通道$ch@r_2$向r_2发送请求$r_1[V:x]$，r_2收到该请求保存在$r_2[V:y]$。r_2通过通道$ch@r_2$向r_1发送应答$r_2[V:t]$，r_1收到该请求保存在$r_1[V:s]$。

(13)$CHO(x_1,x_2,\cdots,x_n)\overset{def}{=}S$为一个 Choreography 的定义，$x_1,x_2,\cdots,x_n$为在$S$中出现的参数名，代表被调用 Choreography 中包含的变量。

(14)$S_{perform}$为对一个已定义的 Choreography 的调用，$y_1,y_2,\cdots,y_n$为调用时的实际参数。

3.3 Abstract WS-CDL 的同构关系

同构关系(Structural Congruence)$\equiv$定义了两个会话在行为上的不可区分性。如果两个会话S_1和S_2满足同构关系$S_1\equiv S_2$，则会话S_1和S_2在行为上完全相同。

会话应该满足下列同构关系：

(1)$[p_1]S_1+[p_2]S_2\equiv[p_2]S_2+[p_1]S_1$

(2)$([p_1]S_1+[p_2]S_2)+[p_3]S_3\equiv[p_1]S_1+([p_2]S_2+[p_3]S_3)$

(3)$[p_1]S+[p_2]S\equiv[p_1\vee p_2]S$

(4)$S_1\parallel S_2\equiv S_2\parallel S_1$ $(S_1\parallel S_2)\parallel S_3\equiv S_1\parallel(S_2\parallel S_3)$

(5)$S\parallel S_{no}\equiv S$ $S\parallel S_{silent}\equiv S$ $S\parallel NULL\equiv S$

(6)$S_{no}.S\equiv S$ $S_{silent}.S\equiv S$ $NULL.S\equiv S$

(7)$CHO(y_1,y_2,\cdots,y_n)\equiv S\{y_1/x_1,y_2/x_2,\cdots,y_2/x_n\}$ if $CHO(x_1,x_2,\cdots,x_n)\overset{def}{=}S$

其中：(1)、(2)指会话的选择执行操作符+满足交换律和结合律；(3)指$[p_1]S$和$[p_2]S$的选择执行所表现出的行为和$[p_1\vee p_2]S$完全一样；(4)指会话的并行执行操作符$\parallel$满足交换律和结合律；(5)指S_{no}、S_{silent}、$NULL$分别和会话S并行执行时表现出的行为和会话S完全一样；(6)指S_{no}、S_{silent}、$NULL$分别和会话S顺序执时表现出的行为和会话S完全一样；(7)中y_i/x_i表示实际参数y_i替换被调用 Choreography 所定义的变量x_i。

3.4 Abstract WS-CDL 的操作语义

Abstract WS-CDL 的形式化操作语义通过会话的推演规则(Reduction Rule)来定义，即通过会话的一步步形如$S\xrightarrow{p,\alpha}S'$的变迁来描述。$S\xrightarrow{p,\alpha}S'$是指当命题$p$为真同时会话$S$执行一个基本会话$\alpha$后，其行为表现为会话$S'$(即会话$S$变迁为会话$S'$)。基本会话$\alpha$定义为

$\alpha::=S_{no}\mid S_{silent}\mid S_{assign}\mid S_{req}\mid S_{resp}\mid S_{req\text{-}resp}\mid S_{perform}\mid NULL$

推演规则定义如下：

$$\frac{S\equiv S'\ S'\xrightarrow{p,\alpha}T'\ T'\equiv T}{S\xrightarrow{p,\alpha}T}\ (\text{STRUCT})$$

$$\frac{}{\alpha\xrightarrow{true,\alpha}NULL}\ (\text{ATOM})\qquad\frac{}{a.S\xrightarrow{true,\alpha}S}\ (\text{SEQ1})$$

$$\frac{S\xrightarrow{p,\alpha}S'}{S.T\xrightarrow{p,\alpha}S'.T}\ (\text{SEQ2})\qquad\frac{S\xrightarrow{p,\alpha}S'}{S\mid T\xrightarrow{p,\alpha}S'\mid T}\ (\text{PAR})$$

$$\frac{S_1\xrightarrow{p_1,\alpha}S_1'}{[p_1]S_1+[p_2]S_2\xrightarrow{p_1,\alpha}S_1'}\ (\text{CHOICE})$$

$$\frac{S\xrightarrow{\neg P_g,NULL}S}{[p_g][p_{rep}]^*S\xrightarrow{\neg p_g,NULL}NULL}\ (\text{NONBLOCK-GUARD})$$

$$\frac{S\xrightarrow{p_g\wedge\neg P_{rep},\alpha}S'}{[p_g][p_{rep}]^*S\xrightarrow{p_g\neg p_{rep},\alpha}S'}\ (\text{NOREPEAT})$$

$$\frac{S\xrightarrow{p_g\wedge P_{rep},\alpha}S'}{[p_g][p_{rep}]^*S\xrightarrow{p_g\wedge P_{rep},\alpha}S'.[P_{rep}]^*S}\ (\text{REPEAT})$$

每个规则中的分子部分为前提(若分子部分为空则表示前提永远为真)，分母部分为结论。其中：

(1)规则 STRUCT 说明了同构在会话推演中的应用。

(2)规则 ATOM 指基本会话的原子不可分性。

(3)规则 SEQ1 指当基本会话α被执行后，顺序执行的会话$\alpha.S$的行为表现为会话S。

(4)规则 SEQ2 指如果会话S经过基本会话α的执行后其行为表现为S'，则顺序执行的会话$S.T$的行为经过基本会话α的执行后表现为$S'.T$。

(5)规则 PAR 指如果会话S经过基本会话α的执行后其

行为表现为 S'，则并行执行的会话 $S \parallel T$ 的行为经过基本会话 α 的执行后表现为 $S' \parallel T$。

(6)规则 CHOICE 指如果命题 p_1 为真，同时会话 S_1 经过基本会话 α 的执行后其行为表现为 S_1'，则选择执行的会话 $[p_1]S_1+[p_2]S_2$ 的行为表现为 S_1'。

(7)规则 NONBLOCK-GUARD 规则指当 p_g 命题为假时，如果 $[p_g][p_{rep}]^* S$ 工作在非阻塞模式下，则会话 $[P_g][P_{rep}]^* S$ 被直接跳过。

(8)规则 NOREPEAT 指如果 p_g 为真，p_{rep} 为假，同时会话 S 经过基本会话 α 的执行后其行为表现为 S'，则会话 $[p_g][p_{rep}]^* S$ 经过基本会话 α 的执行后其行为表现为 S'。

(9)规则 REPEAT 指如果 p_g 和 p_{rep} 都为真，同时会话 S 经过基本会话 α 的执行后其行为表现为 S'，则会话 $[p_g][p_{rep}]^* S$ 经过基本会话 α 的执行后其行为表现为 $S'.[p_{rep}]^* S$。其中 $[p_{rep}]^* S$ 表示如果 p_{rep} 为真则迭代执行 S。

4 从 Chereography 到 Orchestration 的映射

从软件工程的角度来说，Chereography 和 Orchestration 提供了一种自顶向下的方法来设计组合的 Web 服务，即首先利用 WS-CDL 规范定义一个基于全局的 Web 服务之间的交互规则，然后将全局规则映射到每个参与者，就得到每个参与者与其他合作伙伴的局部的交互规则(比如用 BPEL4WS 描述的一个 Process)，每个参与者的局部交互规则就是一个 Orchestration。J. Mendling 和 M. Hafner[11] 给出了从 WS-CDL 到 BPEL4WS 的映射规则，并且实现了一个转换原型系统，但是这种转换直接在 WS-CDL 和 BPEL4WS 之间进行，缺乏形式化的验证机制。

为了形式化地描述从 Chereography 到 Orchestration 的映射，还必须给出关于 Orchestration 的形式化模型。我们采用进程代数 Pi-演算[12] 来描述局部的 Orchestration，即把每个参与协作者用 Pi-演算建模为一个 Pi-演算进程。由于篇幅所限，这里仅给出 Pi-演算的基本语法。一个 Pi 演算的进程 P 可以定义如下：

$$P::=0 \mid \bar{a}x.P \mid a(x).P \mid \sum_{i\in I}P_i \mid [x=y]P \mid [x\neq y]P \mid (\nu x)P \mid !P \mid A(y_1,\cdots,y_n) \mid P_1 \mid P_2$$

其中：

(1)输出前缀 $\bar{a}x$ 表示名字 x 沿着通道 a 输出；输入前缀 $a(x)$ 表示从通道 a 接受一个名字，该名字将替换名字 x。前缀 τ 表示一个进程外部不可见的内部动作(或称为哑动作)。

(2)空进程 0 表示不执行任何动作。

(3)$\alpha.P$ 表示先执行前缀动作 α，然后执行进程 P。符号 . 用来表示顺序执行。

(4)$\sum_{i\in I}P_i$ 表示选择执行，I 为有限索引集合。例如 P_1+P_2 表示选择执行 P_1 或 P_2。

(5)匹配表达式 $[x=y]P$ 表示一个进程，当名字 x 和 y 为同一个名字时，其行为表现为进程 P。匹配表达式 $[x\neq y]P$ 表示一个进程，当名字 x 和 y 不为同一个名字时，其行为表现为进程 P。

(6)约束 $(\nu x)P$ 表示名字 x 被局限在进程 P 内部，P 外部不可见。

(7)$!P$ 表示 P 被重复复制无穷多次。

(8)$A(y_1,\cdots,y_n)$ 为带实参 $y_1,\cdots,y_n$ 的进程调用。

(9)$P_1 \mid P_2$ 表示 2 个进程的并行执行。

映射的基本思想为：对于一个由 WS-CDL 规范定义的 Chereography 元素 CHO，有 CHO $=\langle \bar{S}, \bar{R}, \bar{V}\rangle$。映射从 CHO 元素所定义的最顶层的会话 S_{root} ($S_{root}\in\bar{S}$) 开始，按照下面所定义的映射规则递归进行。对于 CHO 元素所包含的每个角色 $r\in\bar{R}$，执行 Chereography 到 Orchestration 的映射，得到描述角色 r 局部交互行为的 Pi-演算进程。在定义从 Chereography 到 Orchestration 的映射规则前，首先定义下列记号：

(1)函数 $|\rangle_r$ 为一个会话 S 对角色 r 所对应的 Pi-演算进程的映射，即对于一个会话 $S\in\bar{S}$ 和一个 $r\in\bar{R}$，$|S\rangle_r$ 返回角色 r 所对应的 Pi-演算进程。

(2)对于一个命题 p 和一个 $r\in\bar{R}$，p 对 r 的映射记为 p_r。一个全局命题 p 可以映射分解为每个相关角色 r_i ($r_i\in p[R]$，$p[R]$为和命题 p 相关的角色的集合)上的子命题 P_{r_i}。例如命题 $r_1[V:x]=10\wedge r_2[V:y]=20$ 只和角色 r_1 和 r_2 相关，该命题对角色 r_1 映射得到的子命题为 $r_1[V:x]=10$。对于一个和命题无关的角色 r，我们定义命题 p 对该角色的映射为真，即 $p_r=true$ if $r\notin p[R]$。在将全局命题 p 映射分解到各个角色上时，我们把分解的子命题抽象为名字比较(为了和 Pi-演算语法一致)。即对 $\forall r_i\in p[R]$，子命题 p_{r_i} 被形式化为 $x_{p_{r_i}}=v_{p_{r_i}}$。

映射规则如下：

1. $|S_{req}\rangle_r = |request(r1,r2,ch@r2,r1[V:x],r2[V:y])\rangle_r$

$$=\begin{cases}\overline{ch}x & \text{if } r=r1\\ ch(y) & \text{if } r=r2\\ 0 & \text{otherwise}\end{cases}$$

2. $|S_{resp}\rangle_r = |respond(r1,r2,ch@r1,r1[V:x],r2[V:y])\rangle_r$

$$=\begin{cases}ch(x) & \text{if } r=r1\\ \overline{ch}(y) & \text{if } r=r2\\ 0 & \text{otherwise}\end{cases}$$

3. $|S_{req-resp}\rangle_r = |req-resp(r1,r2,ch@r2,r1[V:x],r2[V:y],r1[V:s],r2[V:t])\rangle_r$

$$=\begin{cases}\overline{ch}x.ch(s) & \text{if } r=r1\\ ch(y).\overline{cht} & \text{if } r=r2\\ 0 & \text{otherwise}\end{cases}$$

4. $|S_{assign}\rangle_r = |assign(r_1[V:x],e)\rangle_r$

$$=\begin{cases}(\nu s)(\bar{s}e.s(x)) & \text{if } r=r1\\ 0 & \text{otherwise}\end{cases}$$

5. $|S_{no}\rangle_r = 0$ $for\ \forall r\in\bar{R}$

$$|S_{silent}\rangle_r=|r_1(t)\rangle_r=\begin{cases}\tau & \text{if } r=r_1\\ 0 & \text{otherwise}\end{cases}$$

6. $|S_{perform}\rangle_r=|CHO(y_1,y_2,\cdots,y_n)\rangle_r=|S\rangle_r(y_1,y_2,\cdots,y_n)$ if $CHO(x_1,x_2,\cdots,x_n)\overset{def}{=}S$

7. $|S_1\parallel S_2\rangle_r = |S_1\rangle_r \mid |S_2\rangle_r$

8. $|S_1.S_2\rangle_r=|S_1\rangle_r.|S_2\rangle_r$

9. $|[p1]S_1+[p2]S_2\rangle_r$

$$=\begin{cases}[x_{p_{1_r}}=v_{p_{1_r}}]|S_1\rangle_r+[x_{p_{2_r}}=v_{p_{2_r}}]|S_2\rangle_r & \text{if } r\in p_1[R]\wedge r\in p_2[R]\\ [x_{p_{1_r}}=v_{p_{1_r}}]|S_1\rangle_r+|S_2\rangle_r & \text{if } r\in p_1[R]\wedge r\notin p_2[R]\\ |S_1\rangle_r+[x_{p_{2_r}}=v_{p_{2_r}}]|S_2\rangle_r & \text{if } r\notin p_1[R]\wedge r\in p_2[R]\\ |S_1\rangle_r+|S_2\rangle_r & \text{otherwise}\end{cases}$$

10. $|[P_g][P_{rep}]^* S\rangle_r =$

$$\begin{cases} [x_{p_{g_r}} = v_{p_{g_r}}](|S\rangle_r . [x_{p_{rep_r}} = v_{prep_r}]|[P_g][P_{rep}]^* S\rangle_r) \\ \qquad \text{if } r \in p_g[R] \wedge r \in p_{rep}[R] \\ [x_{p_{g_r}} = v_{p_{g_r}}](|S\rangle_r . |[P_g][P_{rep}]^* S\rangle_r) \\ \qquad \text{if } r \in p_g[R] \wedge r \notin p_{rep}[R] \\ |S\rangle_r . [x_{p_{rep_r}} = v_{p_{rep_r}}]|[P_g][P_{rep}]^* S]\rangle_r \\ \qquad \text{if } r \notin p_g[R] \wedge r \in p_{rep}[R] \\ |S\rangle_r . |[P_g][P_{rep}]^* S\rangle_r \quad \text{otherwise} \end{cases}$$

其中：

(1)规则 1 指一个 Request 类型的会话只和参与该基本会话的角色 r_1，r_2 有关，如果对角色 r_1 作映射，则 r_1 对应的 Pi-演算进程为 $\overline{ch}x$（r_1 作为请求者从通道 ch 输出变量 x）；如果对角色 r_2 作映射，则 r_2 对应的 Pi-演算进程为 $ch(y)$（r_2 作为被请求者从通道 ch 输入一个变量保存在 y 中）；如果对其他角色作映射，则对应的 Pi-演算进程为 0（其他角色没参与该基本会话）。规则 2,3,5 的映射方法与规则 1 类似。

(2)规则 4 利用一个内部通道 s 上输出变量 e，然后从 s 上输入该变量保存在变量 x 中来模拟变量间的赋值。

(3)规则 6 指如果 $CHO(x_1, x_2, \cdots, x_n) \stackrel{\text{def}}{=} S$，则 $CHO(y_1, y_2, \cdots, y_n)$ 对角色 r 的映射结果为对 Pi-演算进程 $|S\rangle_r$ 的带实参 $(y_1, y_2, \cdots, y_n)$ 的调用。

(4)规则 7 指两个并行执行的会话 $S_1 \parallel S_2$ 对角色 r 的映射得到的也是两个并行执行的 Pi-演算进程。规则 8 和规则 7 类似。

(5)规则 9 指两个选择执行的会话 $[p_1]S_1 + [p_2]S_2$ 对角色 r 的映射得到的也是两个选择执行的 Pi-演算进程 $|S_1\rangle_r$ 和 $|S_2\rangle_r$，如果命题 p_1 和角色 r 有关，则 Pi-演算进程 $|S_1\rangle_r$ 前面要加上命题 p_1 对角色 r 映射得到的子命题 $x_{p_{1_r}} = v_{p_{1_r}}$，如果命题 p_1 和角色 r 无关，则命题 p_1 对角色 r 的映射得到的子命题为真，则 Pi-演算进程 $|S_1\rangle r$ 前不需要加条件判断。命题 p_2 的处理类似。

(6)规则 10 指一个迭代执行的会话 $[P_g][P_{rep}]^* S$ 对角色 r 的映射得到的是一个迭代执行的 Pi-演算进程，这里利用了 Pi-演算进程的递归定义来建模迭代执行的进程。如果角色 r 和命题 P_g 和 P_{rep} 都相关（if $r \in p_g[R] \wedge r \in p_{rep}[R]$），则命题 P_g 和 P_{rep} 对角色 r 分解得到的子命题为：$x_{p_{g_r}} = v_{p_{g_r}}$ 和 $x_{p_{rep_r}} = v_{p_{rep_r}}$，因此映射得到的 Pi-演算进程应该为：

$$|[P_g][P_{rep}]^* S\rangle_r = [x_{p_{g_r}} = v_{p_{g_r}}](|S\rangle_r . [x_{p_{rep_r}} = v_{p_{rep_r}}]|[P_g][P_{rep}]^* S\rangle_r)$$

即首先判断命题 $x_{p_{g_r}} = v_{p_{g_r}}$，如果该命题为真则首先执行进程 $|S\rangle_r$，再判断命题 $x_{p_{rep_r}} = v_{p_{rep_r}}$，如果该命题为真再迭代执行上述过程。如果角色 r 和命题 P_g 和 P_{rep} 都无关，则得到一个递归定义的 Pi-演算进程：

$$|[P_g][P_{rep}]^* S\rangle_r = |S\rangle_r . |[P_g][P_{rep}]^* S\rangle_r$$

其效果就是不断地重复执行进程 $|S\rangle_r$。其他两种情况处理方法类似。

5 案例研究和模型验证

下面利用 Abstract WS-CDL 形式化模型来描述一个具体 Web 服务组合案例的全局相互交互行为，然后应用所定义的从全局到局部的映射规则，得到每个子 Web 服务的 Pi-演算进程模型。这个 Web 服务组合案例的基于全局的交互如图 2 所示（为节省篇幅，该案例的 WS-CDL 描述略去）。

在这个案例中四个角色分别为：Consumer（记为 C）、WebShop（记为 W）、Delivery（记为 D）和 Bank（记为 B）。Consumer 和 WebShop 间交互的通道为 $c2w$ 和 $w2c$，WebShop 和 Delivery 间交互通道为 $w2d$ 和 $d2r$，WebShop 和 Bank 间交互通道为 $w2b$ 和 $b2w$。四个角色间的交互如下：Consumer 向 WebShop 发送订单（PO）；WebShop 发起 2 个并行的子会话：分别将 PO 转发给 Delivery 和 Bank；Delivery 和 Bank 收到 PO 后分别返回帐单信息（$Bill$）和送货信息（$Send$）给 WebShop，WebShop 分别将帐单信息（$Bill$）和递送信息（$Send$）返回给 Consumer。用 Abstract WS-CDL 可以描述如下：

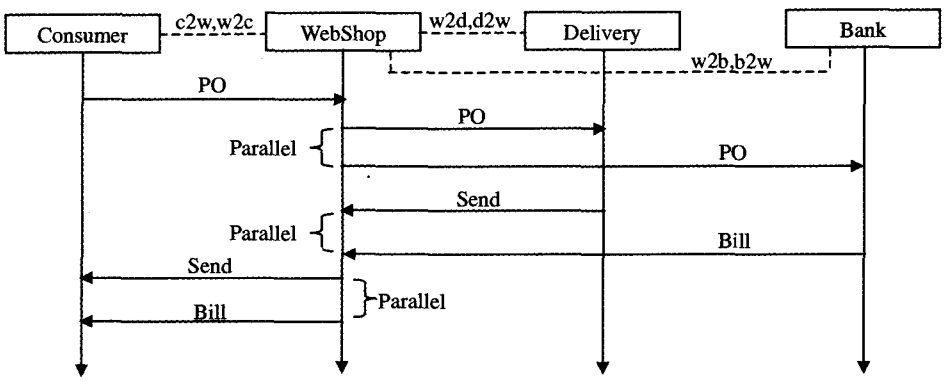

图 2 基于全局的交互

$S_{init} = request(C, W, c2w@w, C[V:PO], W[V:PO])$

$S_{pay} = request(W, B, w2b@b, W[V:PO], B[V:PO]) .$
$respond(W, B, b2w@w, W[V:Bill], B[V:Bill]) .$
$respond(C, W, w2c@c, C[V:Bill], W[V:Bill])>>$

$S_{send} = request(W, D, w2d@d, W[V:PO], D[V:PO]) .$
$respond(W, D, d2w@w, W[V:Send], D[V:Send]) .$
$respond(C, W, w2c@c, C[V:Send], W[V:Send])$

$CHO = Sinit . (S_{pay} \parallel S_{send}) =$
$request(C, W, c2w@w, C[V:PO], W[V:PO]) .$

($request(W, B, w2b@b, W[V:PO], B[V:PO])$.
$respond(W, B, b2w@w, W[V:Bill], B[V:Bill])$.
$respond(C, W, w2c@c, C[V:Bill], W[V:Bill])$ ||
$request(W, D, w2d@d, W[V:PO], D[V:PO])$.
$respond(W, D, d2w@w, W[V:Send], D[V:Send])$.
$respond(C, W, w2c@c, C[V:Send], W[V:Send])$)

现在利用第4节所定义的映射规则,将基于全局的四个角色间的交互模型分别对四个角色做映射,得到描述四个角色局部交互行为的Pi-演算进程如下:

$P_C = \overline{c2w}PO.(w2c(Bill).0 | w2c(Send).0)$

$P_D = w2d(PO).\overline{d2w}Send.0$

$P_B = w2b(PO).\overline{b2w}Bill.0$

$P_W = c2w(PO).(\overline{w2b}PO.b2w(Bill).\overline{w2c}Bill.0 | \overline{w2d}PO.d2w(Send).\overline{w2c}.Send.0)$

为Web服务组合建立形式化模型的目的是为了验证Web服务组合的正确性,以保证那些并发的Web子服务能按照设计规范正确地交互。WS-CDL规范的形式化模型Abstract WS-CDL为我们提供了这样一个机制:第一步,在设计阶段先从全局角度定义好各个Web子服务间交互的规则,利用形式化模型的推演规则对模型所描述的系统进行推演,证明所有子Web服务之间可以正确地交互,最终可以达到一个终止状态(空会话NULL)。第二步,将全局模型利用从Chereography到Orchestration的映射规则得到每个Web子服务的Pi-演算描述,利用Pi-演算的验证工具对每个Web子服务进行验证,这一步的验证包括死锁的检查(即是否存在不可达的交互动作)、活锁的检查(即是否每个交互动作都是可达的)和服务的匹配性(即组合的Web服务和客户能否正确交互)。

利用Abstract WS-CDL定义的推演规则,可以对案例所描述的系统的一种可能的会话顺序推演如下(为了节省篇幅,我们用α^*代表若干步连续的基本会话):

1. $S_{init}.(S_{pay} \| S_{send}) \xrightarrow{true,\alpha} NULL.(S_{pay} \| S_{end})$(规则SEQ2和ATOM)

$\alpha = request(C, W, c2w, C[V:PO], W[V:PO])$

2. $NULL.(S_{pay} \| S_{send}) \xrightarrow{true,\alpha^*} NULL.(NULL \| S_{end})$(规则PAR)

$\alpha^* = request(W, B, w2b, W[V:PO], B[V:PO]).respond(W, B, b2w, W[V:Bill], B[V:Bill]).respond(C, W, w2c, C[V:Bill], W[V:Bill])$

3. $NULL.(NULL \| S_{send}) \xrightarrow{true,\alpha^*} NULL.(NULL \| NULL)$(规则PAR)

$\alpha^* = request(W, D, w2d, W[V:PO], D[V:PO]).respond(W, D, d2w, W[V:Send], D[V:Send]).respond(C, W, w2c, C[V:Send], W[V:Send])$

这里的推演只是一种可能的会话发生顺序。可以用类似的方法对所有可能的会话发生顺序进行推演,都可以得出结论:系统可以达到一个NULL会话最终状态。

验证的第二步对从全局模型映射得到的四个Web子服务的Pi-演算描述进行验证。我们采用Mobility Workbench[13](MWB)作为验证工具。MWB采用标准元语言(Standard ML)开发,运行在新泽西标准元语言编译器(New Jersey SML Compiler version 11.0)环境下。我们利用该工具验证的环境为Windows 2000 Server,采用MWB'99 4.137版。为了和MWB的语法保持一致,我们用$'a\langle x\rangle$表示从a通道输出名字x,用ν表示一个约束名(内部名),除进程名外所有的名字变为小写。

首先利用MBW提供的deadlocks命令,可以验证进程P_C、P_D、P_B、P_W以及闭合系统$P_C | P_D | P_B | P_W$都不存在死锁;利用step命令验证这些进程的所有动作都是可达的。下一步验证客户进程P_C和组合的Web服务进程$P_D | P_B | P_W$之间是匹配的,即P_C和$P_D | P_B | P_W$之间能否正确地交互。为了验证进程间的匹配性,就要用到Pi-演算理论中的弱相似概念。两个进程P和Q弱相似是指:外部观察者所能观察到的P和Q的外部行为和变迁是一致的。Pi-演算以形式化的方法给出了两个进程行为等价的精确定义[14]。利用文[15]提出的方法,我们首先将客户进程P_C反转(输出变输入,输入变输出),得到P_C的反转进程RP_C和组合的Web服务进程$P_D | P_B | P_W$:

$RP_C = c2w(po).('w2c\langle bill\rangle.0 | 'w2c\langle send\rangle.0)$

$P_D | P_B | P_W = w2d(po).'d2w\langle send\rangle.0 | w2b(po).'b2w\langle bill\rangle.0 | c2w(po).('w2b\langle po\rangle.b2w(bill).'w2c\langle bill\rangle.0 | 'w2d\langle po\rangle.d2w(send).'w2c\langle send\rangle.0)$

如果RP_C和$P_D | P_B | P_W$是弱相似的,那么客户进程P_C和组合的Web服务进程$P_D | P_B | P_W$之间是匹配的。在MWB运行环境下通过运行weq命令,验证了RP_C和$P_D | P_B | P_W$是弱相似的。因此,客户进程P_C和组合的Web服务进程$P_D | P_B | P_W$之间是匹配的。

总结 WS-CDL规范从全局的角度描述了一组Web服务之间协作、交互的规则。为了保证它们彼此间交互的正确性,一种有效的办法就是将WS-CDL所描述的Web服务组合形式化并且加以验证。本文针对WS-CDL规范建立了一个基于全局的形式化模型的框架Abstract WS-CDL,包括形式化描述的语法,同构关系和操作语义。同时,我们还定义了一套从该模型框架到基于Pi-演算描述的Orchestration的映射规则,并且通过具体案例给出了全局和局部二个层次的模型验证方法。

将来的研究工作包括:如何自动地将一个WS-CDL文档翻译成Abstract WS-CDL;如何实现Abstract WS-CDL的自动验证;如何自动地将Abstract WS-CDL映射到Orchestration层。

参考文献

1 Bucchiarone A, Gnesi S. A Survey on Services Composition Languages and Models. In: Proceedings of the International Workshop on Web Services Modeling and Testing, ITALY, 2006. 51~63

2 Dijkman R, Dumas M. Service-oriented Design: A Multi-viewpoint Approach. International Journal of Cooperative Information Systems, 2004, 13(4): 337~368

3 Andrews T, Cubera F, Dholakia H, et al. Business Process Execution Language for Web Services Version1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2003-

05-05
4 Kavantzas N, Burdett D, Rizinger G, et al. Web Service Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, 2005-11
5 Brogi A, Canal C, Pimentel E, et al. Formalizing Web Service Choreographies. Electronic Notes in Theoretical Computer Science, 2004, 105: 73～94
6 Arkin A, Askary S, Fordin S, et al. Web Service Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/wsci/, 2002-08
7 Foster H, Uchitel S, Magee J, et al. Model-Based analysis of Obligations in Web Service Choreography. In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), USA, 2006. 149～157
8 Yeung W L, Wang Ji, Dong Wei. Verifying Choreographic Descriptions of Web Services Based on CSP. In: Proceedings of the IEEE Services Computing Workshops (SCW'06), USA, 2006. 97～104
9 Busi N, Gorrieri R, Guidi C, et al. Towards a formal framework for Choreography. In: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2005), USA, 2005. 107～112
10 Gorrieri R, Guidi C, Lucci R. Reasoning about interaction patterns in Choreography. In: Proceedings of 2nd International Workshop on Web Services and Formal Methods (WS-FM '05), France, 2005. 333～348
11 Mendling J, Hafner M. From WS-CDL Choreography to BPEL Process Orchestration. wi.wu-wien.ac.at/home/mendling/publications/TR06-CDL.pdf
12 Milner R. Communicating and Mobile Systems: The Pi Calculus. Cambridge: Cambridge University Press, 1999
13 Victor B, Moller F. The Mobility Workbench-A Tool for the Pi-Calculus. In: Proceedings of the 6th International Conference on Computer Aided Verification, USA, 1994. 428～440
14 Sangiorgi D. A Theory of Bisimulation for the pi-Calculus. Acta Informatica, 1996, 3(1): 69～97
15 Salaün G, Bordeaux L, Schaerf M. Describing and Reasoning on Web Services Using Process Algebra. In: Proceedings of the IEEE International Conference on Web Services (ICWS'04), USA, 2004. 43～50

(上接第4页)

在开发阶段提交的大量缺陷是编码的错误导致(返回值检查、空指针引用等)的,在发布以后提交的缺陷则大多是由于功能设计导致的缺陷或者逻辑错误[10]。程序缺陷分析的研究也面临同样的问题,最典型的就是缺陷和更改之间没有关联。虽然通过文本分析这样的启发式方法能够建立缺陷和更改之间的关联,但是如果开发项目的管理没有保障,开发人员仅仅根据自己的习惯添加说明,那么这种方法的准确性和完备性就无法得到保证。

参 考 文 献

1 Ximbiot. Version Management with CVS. http://ximbiot.com/cvs/manual/.
2 Collins-Sussman B, Fitzpatrick B W, Pilato C M. Version Control with Subversion. 1 ed. http://svnbook.red-bean.com/en/1.1/svn-book.html, 2005
3 BugZilla. Homepage. http://www.bugzilla.org/
4 Godfrey M, Dong X, Kasper C, et al. Four Interesting Ways in Which History Can Teach Us About Software. In: 1st International Workshop on Mining Software Repositories(MSR-04), 2004
5 Kagdi H, Collard M, Maletic J. Towards a Taxonomy of Approaches for Mining of Source Code Repositories. In: 2nd International Workshop on Mining Software Repositories (MSR 2005), Saint Louis, Missouri, USA, 2005
6 MSR. Mining Software Repositories Homepage. http://msr.uwaterloo.ca
7 Hassan A E, Holt R C, Mockus A. Report on MSR 2004. In: International Workshop on Mining Software Repositories, 2004
8 Hassan A E, Holt R C, Diehl S. Report on MSR 2005. In: International Workshop on Mining Software Repositories, 2005
9 Fenton N E, Neil M. A critique of software defect prediction models. Software Engineering, IEEE Transactions on, 1999, 25: 675～689
10 Williams C C, Hollingsworth J K. Bug Driven Bug Finders. In: 1st International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, Scotland, 2004
11 Williams C C, Hollingsworth J K. Automatic mining of source code repositories to improve bug finding techniques. Software Engineering, IEEE Transactions on, 2005, 31: 466～480
12 Görg C. Weißgerber P. Error Detection by Refactoring Reconstruction. In: 2nd International Workshop on Mining Software Repositories, St Louis, Missouri, 2005
13 Nagappan N, Ball T, Zeller A. Mining Metrics to Predict Component Failures. In: 3rd International Workshop on Mining Software Repositories (MSR 2006), Shanghai China, 2006
14 Atkins D L, Ball T, Graves T L, et al. Using version control data to evaluate the impact of software tools: a case study of the Version Editor. Software Engineering, IEEE Transactions on, 2002, 28: 625～637
15 Ohlsson N, Alberg H. Predicting fault-prone software modules in telephone switches. Software Engineering, IEEE Transactions on, 1996, 22: 886～894
16 Khoshgoftaar T M, Allen E B, Halstead R, et al. Detection of fault-prone software modules during a spiral life cycle. In: Proceedings of the 1996 IEEE Conference on Software Maintenance, ICSM, Monterey, CA, USA, Nov. 1996
17 Gefen D, Schneberger S L. The non-homogeneous maintenance periods: a case study of software modifications. In: Software Maintenance 1996, Proceedings., International Conference on, 1996
18 Ostrand T J, Weyuker E J, et al. Predicting the Location and Number of Faults in Large Software Systems. Software Engineering, IEEE Transactions on, 2005, 31: 340～355
19 Fenton N E, Pfleeger S L. Software Metrics: A Rigious & Practical Approach: PWS Publishing Company, a division of Thomson Learning, 1997
20 Padberg F, Ragg T, Schoknecht R. Using machine learning for estimating the defect content after an inspection. Software Engineering, IEEE Transactions on, 2004, 30: 17～28
21 Hassan A E, Holt R C. The Top Ten List: Dynamic Fault Prediction. In: 21st IEEE International Conference on Software Maintenance (ICSM 2005), 2005
22 Willianms C C, Hollingsworth J K. Bug Driven Bug Finders. In: 1st International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, Scotland, 2004
23 Schwaber C, Orlov L M, Zetie C, et al. The Forrester Wave(tm): Process-Centric Software Configuration Management. Cambridge, MA 02139 USA: Forrester Research Inc, November 14, 2005
24 Zeller A. Yesterday, My Program Worked. Today, It Does Not. Why? 1999
25 Mockus A, Votta L G. Identifying Reasons for Software Changes using Historic Databases. In: International Conference on Software Maintenance, San Jose, California, 2000
26 Fischer M, Pinzger M, Gall H. Analyzing and relating bug report data for feature tracking. In: Reverse Engineering, 2003. WCRE 2003. Proceedings. 10th Working Conference on, 2003
27 Fischer M, Pinzger M, Gall H. Populating a Release History Database from version control and bug tracking systems. In: Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, 2003
28 Sliwerski J, Zimmermann T, Zeller A. When Do Changes Induce Fixes? In: 2nd International Workshop on Mining Software Repositories (MSR 2005), St Louis, Missouri, 2005
29 Purushothaman R, Perry D E. Toward understanding the rhetoric of small source code changes. Software Engineering, IEEE Transactions on, 2005, 31: 511～526
30 Sandusky R J, Gasser L, Ripoche G. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OOS Development Community. In: 1st International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, Scotland, 2004
31 Zimmermann T, WeiBgerber P. Preprocessing is a Prerequisite for Any Analysis of CVS Data. In: 1st International Workshop on Mining Software Repositories (MSR), Edinburgh, UK, 2004