ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1224−1233
DOI: 10.3724/SP.J.1001.2008.01224

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

扩展呼叫处理语言:一种面向综合通信服务的语言*

杨 骏+, 温嘉佳, 陈俊亮

(北京邮电大学 网络与交换技术国家重点实验室,北京 100876)

Extended-Calling Process Language: An Integrated-Communication-Services-Oriented Language

YANG Qin+, WEN Jia-Jia, CHEN Jun-Liang

(State Key Laboratory of Net Working and Switching Technology, Beijing University of Post and Telecommunication, Beijing 100876, China)

+ Corresponding author: E-mail: yangkiss@people.com.cn, http://www.cs.bupt.cn/cs_web/

Yang Q, Wen JJ, Chen JL. Extended-Calling process language: An integrated-communication-services-oriented language. *Journal of Software*, 2008,19(5):1224−1233. http://www.jos.org.cn/1000-9825/19/1224.htm

Abstract: This paper presents a service flow describing language XPL (extended-calling process language) which mainly focuses on value-added service in telecom field, and the supporting system of this language. XPL has the ability to describe both calling and digital services. This language is highly abstract, easily used and fast in developing service. Due to the base of SOA (services-oriented architecture), the supporting system suits service creation under the condition of net merging.

Key words: service creation; XPL (extended-calling process language); component; software reuse; SOA (services-oriented architecture); Web service; service fragment

摘 要: 介绍了一种面向电信增值业务领域的流程描述语言 XPL(extended-calling process language)以及支持该语言的业务生成系统.XPL 拥有较强的语音类业务描述能力和数据类业务描述能力,具有抽象层次高,使用灵活、简单,开发业务速度快的特点.该业务生成系统基于面向服务的构架(services-oriented architecture,简称 SOA),适用于网络融合条件下的业务生成.

关键词: 业务生成;XPL(extended-calling process language);构件;软件复用;SOA(services-oriented architecture);Web Services;业务片段

中图法分类号: TP393 **文献标识码**: A

随着电信网络和互联网络向下一代网络的方向演进,如何快速、灵活地开发种类丰富的新型增值业务是电信领域和计算机领域所面临的一个重要问题.

业务开发的一个基本问题是如何描述业务需求.XML(extensible markup language)因为具有人和机器易于

* Supported by the National Natural Science Foundation of China under Grant No.60432010 (国家自然科学基金); the National Basic Research Program of China under Grant No.2007CB307100 (国家重点基础研究发展计划(973)); the China Next Generation Internet under Grant No.CNGI-04-14-1T (中国下一代因特网)

Received 2006-10-16; Accepted 2007-01-04

理解、与底层实现语言无关、易于图形化表示等优点而成为业务描述语言的重要发展方向之一.基于 XML 的语言可以分为通用型语言和面向特定领域的专业语言两种类型.通用型语言以IBM的BPEL[1]为代表,主要针对通用型流程的控制,接近高级语言的水平,已经成为工作流领域的工业标准.面向特定领域的专业语言有很多,在电信领域面向呼叫流程控制的具有代表性的语言有 IETF 的 CPL[2,3]、W3C 的 CCXML[4]和 JAIN 论坛的SCML[5,6]等,其语言元素本身就是对呼叫处理的高度抽象,这些专业语言已经或正在成为国际标准.比较而言,通用型语言有适用面广的优势,缺点是语言复杂、开发效率低、对开发人员的要求高.专业语言的优点是语言简单、开发效率高、对开发人员的要求低,但不能描述特定领域以外的业务.快速地开发、部署业务是企业保持竞争力的关键之一,所以,专业语言具有相当大的研究价值.

融合网络条件下通信类业务的特点应该是业务种类繁多、内容丰富、具有个性化,CPL,CCXML,SCML 基本上针对传统的呼叫处理业务,不能描述短信、彩信、数据库操作等数据类业务,而且 CPL,SCML 不能进行并发、循环等操作,受到的局限较大.我们设计并实现了一种面向综合通信服务的专业语言 XPL.XPL 综合了呼叫、短信、彩信、Web 网页、E-mail 等通信手段,支持较为复杂的数据库操作.XPL 提供受到限制的并发和循环操作,确保不会出现死锁和死循环.与目前的 CPL 等专业语言相比,XPL 具有面向领域范围较大、能够适用于较为复杂的业务的优点,更能适用于融合网络条件下的通信类业务.

1 基于 XML 的流程描述语言 XPL

1.1 XPL概述

XPL 是一种基于 XML 的流程描述语言,以 Parlay Web Service 和其他类型的 Web Services 作为 XPL 的通信能力.XPL 定义了一套流程描述 schema,各种通信能力抽象成特定的标签.用 XPL 所描述的业务主要由一个或多个业务脚本文件组成,各个脚本文件可以被指定的触发条件触发执行,触发事件可以是呼叫事件、短信事件、Web 网页点击事件等等.业务开发者在业务流程中可以将呼叫、短信、彩信、Web 网页、E-mail 等通信能力进行灵活的组合,也可以在流程中嵌入较复杂的数据库操作.XPL 具有较好的可扩展性,用户可以自己定义所需要的标签.

定义 1. *Service*={(*Process*,*Trigger*)},*Service* 是用 XPL 所书写的业务,*Process* 是可独立运行的业务进程,对应于一个业务脚本,在业务执行环境中,一个 *Process* 会有多个实例同时工作.*Trigger* 是业务开发者所配置的触发该业务进程产生新实例的条件,比如网络上报呼叫事件的特服号码.*Process*=(*Flow*,*DB*),其中,*Flow* 是用 XPL 书写的业务流程脚本,*DB* 是供 *Flow* 调用的数据库操作脚本,*DB*=(*SELECT*,*UPDATE*,*INSERT*,*DELETE*),这 4 种操作符合 ANSI SQL 标准.

Flow 中的基本语法概念用 BNF 定义如下:

(1) 脚本流程入口::=“〈*incoming*〉”,是 *Flow* 主流程的根节点.

(2) 顺序执行::=〈某个标签有单个子标签,则父节点执行完后子节点执行〉.

(3) 选择操作::=〈某个标签有多个子标签,则某一个满足条件的子标签继续执行〉.

(4) 变量操作::=“〈*Declare*〉”|“〈*Assign*〉”,声明变量|给变量赋值,为简化起见,标签省去属性说明.

(5) 脚本子树::=“〈*subaction*〉”,以该标签为根定义一棵脚本子树,表示一段可复用的业务子流程,标签属性 *id* 为子树名称.

(6) 复用子树::=“〈*sub*〉”,在指定的脚本子树以外调用子流程,标签属性 *ref* 为指定的子树名称.

(7) 数据库操作::=“〈*SqlAdd*〉”|“〈*SqlDel*〉”|“〈*SqlUpdate*〉”|“〈*SqlSelect*〉”,调用 *DB* 中定义的方法,为简化起见,标签省去属性和子标签说明.

(8) 循环操作::=“〈*goto*〉”,在指定的脚本子树内使用,重新进入该子流程,标签属性 *ref* 为指定的子树名称.

(9) 业务能力标签::=〈完成特定业务动作〉,包含的标签数量很多,这里列举一些.如〈*sendSMS*〉发送短信、〈*getLocation*〉取得移动用户当前位置、〈*sendMMS*〉发送彩信、〈*makeCall*〉第三方发起的呼叫、〈*proxy*〉转接当前的呼叫等等.为简化起见,标签省去了属性和子标签说明.各种通信能力抽象成的特定标签

就属于业务能力标签的范畴.

(10) 上下文操作::=“*context*”“.”*key*,关键字 *context* 用在标签属性中,用来从业务上下文中抽取信息并为标签的属性赋值,*key* 为上下文中的键.

1.2 XPL的Web Services事件模型

用 XPL 描述的业务通过 Web Services 接口与网络交互,所以,XPL 是一种专业型 Web Services 组合语言.通用型的 Web Services 组合语言抽象层次较低,例如,在 BPEL4WS 中是使用⟨*invoke*⟩和⟨*receive*⟩等通用性的标签来对 Web Services 接口进行操作,操作的细节完全暴露给程序员.而 XPL 面向特定领域,抽象层次较高,业务开发者直接按照需求描述业务,不必关心 Web Services 操作问题,例如,⟨*makeCall*⟩表示试图建立主被叫方之间的通话,⟨*sendSMS*⟩表示发送短信,具体的 Web Services 操作被系统封装.XPL 的 Web Services 事件模型确定了 Web Services 操作的封装机制,该机制包含以下 3 个方面.

1.2.1 业务和网络之间的 Web Services 交互机制

业务流程不会自主执行,当从网络侧调用业务侧的 Web Services 接口时,会触发执行一个业务流程片断,业务流程片断运行结束后将结果返回给网络,业务等待网络的下一次调用.整个树状业务脚本就是由单个或者多个这样的业务流程片段组成的.在业务流程片段的执行过程中,业务可以同步调用网络侧的 Web Services 接口,同步调用结束后业务流程继续执行.业务脚本根据 XPL 标签的类型自动划分出业务流程片断,见定义 2.

定义 2. 如图 1 所示,XPL 标签分为响应标签 A、分段标签 C 和普通标签 G 三种类型,$A\cap C=\varnothing$,$A\cap G=\varnothing$,$C\cap G=\varnothing$.业务 *service*={*fragment*},业务片段 *fragment*=(*Event*,*tree*),*Event* 是网络向业务发起的 Web Services 调用,业务执行环境首先建立满射 f:{*Event*}→*A*.*tree* 是响应 *Event* 执行的一段脚本树,*Flow*={*tree*}(见定义 1).*tree* 的根∈*A*,且若不是⟨*incoming*⟩,则在 *Flow* 中的父节点∈*C*.*tree* 的根和叶之间的节点∈*G*.*tree* 从根执行到某个叶节点,完成对该 *Event* 的响应.若有其他 *tree* 的根是该叶节点的子节点,则说明下一个 *Event* 将会调用执行该 *tree*,若有多个这样的 *tree*,则根据实际发生的 *Event* 按照 f 选择执行相应的 *tree*.*tree* 中节点可以同步调用网络侧 Web Services 接口.*Event* 的输入参数 *InPutMessage* 在 *tree* 执行之前先装入上下文,输出参数 *OutPutMessage* 从上下文中提取.

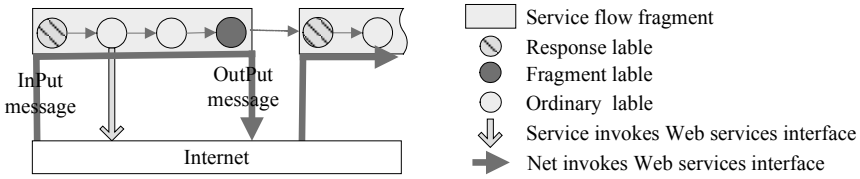


Fig.1 Web Services event model

图 1 Web Services 事件模型

1.2.2 XPL 的消息分发机制

目前,Web Services 标准对有状态服务等交互方式支持不足[7,8],若 *Event* 满足条件 *Trigger*(见定义 1),则说明产生新的业务实例;否则,需将 *Event* 分发给某个特定的业务实例.在 BPEL4WS 中,这种分发是程序员编程控制的.因为 XPL 面向特定的行业,这种分发可以总结抽象出来,从而简化编程工作.在 XPL 中,*Event* 事件主要有呼叫事件、短信通知事件、网页点击事件,其中,主/被叫号码、短信源号码、网页 *id* 等参数和业务实例之间存在对应关系.这些 Web Services 的接口参数事先注册在系统中,称为注册参数.业务和网络之间的 Web Services 调用会对相应的注册参数值进行更新,系统负责动态地维护业务实例和业务注册参数之间的对应关系,并根据 *Event* 事件中的注册参数自动找到对应的业务实例.消息分发机制使得 XPL 只关注于主要业务流程的描述,提高了抽象层次.

1.2.3 Web Services 交互协议和 XPL 的 *schema*

XPL 的 *schema* 包含了二元关系 *R*:*cRa*,其中,*c*∈*C*,*a*∈*A*.比如,分段标签⟨*proxy*⟩的子标签只能从响应标签

⟨*noanswer*⟩,⟨*busy*⟩,⟨*calleeonhook*⟩,⟨*timeout*⟩中选取,表示将当前呼叫转接到指定的被叫方后,网络向业务上报的 Web Services 事件只能是被叫无应答、被叫忙、通话结束被叫挂机,或超时.所以,业务和网络之间的 Web Services 交互协议在一定程度上映射在关系 *R* 中,从而通过 *schema* 可以自动进行 Web Services 互操作的业务逻辑检验.这在通用型语言中是不能实现的,如在 BPEL4WS 中,上例转接当前呼叫这个业务逻辑完全依靠程序员使用⟨*receive*⟩和⟨*invoke*⟩标签对 Web Services 接口操作,BPEL4WS 的 *schema* 不能检查出业务逻辑错误.

1.3 XPL的循环控制机制

专业型语言因为对开发者要求不高,所以非常注意语言的安全性问题.XPL 提出了一种用户行为驱动的循环操作模式,以避免死循环的发生.

XPL 规定:(1) 使用普通标签⟨*goto*⟩进行循环操作时,⟨*goto*⟩和属性 *ref* 指向的普通标签⟨*subaction*⟩之间(即循环体内)至少要有 1 个分段标签和响应标签构成的组合,即在每次循环的过程中,业务都必须等待网络的触发才能继续执行.由于电信网络是一个终端(电话、网页等)用户驱动的网络,所以,网络对业务的 Web Services 调用也是用户行为驱动的结果,即调用的生命周期不长于用户行为的生命周期.另外,考虑到异常情况,XPL 的所有分段标签都有响应子标签⟨*timeout*⟩(超时),定义如下:

定义 3. *timeout*=(*NetException*|*UserNoAction*|*ProtocolProblem*),其中,*NetException* 是指网络出现异常没有调用成功业务侧 Web Services 接口;*UserNoAction* 是指用户长时间无响应,用户行为生命周期已结束;*ProtocolProblem* 是指用户行为和 Web Services 接口之间的失配,没有为用户的相关行为定义 Web Services 接口.

分段标签执行之后,若在非⟨*timeout*⟩子标签的路径上发生回溯,则用户、网络、业务构成一个受用户控制的系统,用户的此次行为驱动此次循环,业务不能自由进入下一次循环;若在⟨*timeout*⟩子标签的路径上发生回溯,因为 *timeout* 事件不是由终端用户驱动的,则业务和网络构成一个自激励的系统,仍可能出现死循环,故 XPL 规定:(2) 若⟨*timeout*⟩是⟨*goto*⟩的祖先节点,则⟨*timeout*⟩和⟨*goto*⟩之间至少要有 1 个分段标签和响应标签构成的组合(如图 2 所示).

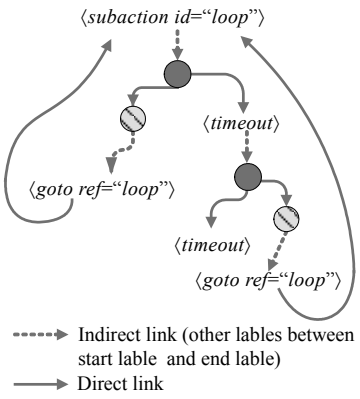


Fig.2 The loop of XPL

图 2 XPL 的循环

通过规定(1)、规定(2)的约束和定义 3,XPL 规定了其循环是一种用户行为驱动的循环操作,循环的生命周期不长于用户行为的生命周期,业务流程没有死循环,随着用户的退出而最终结束.

1.4 XPL的并发控制机制

XPL 提供一种简单、松散的并发操作.不像 BEPL 的并发操作那样是在业务流程内部控制发起,在 XPL 中,由网络触发出的 *Process* 的多个实例通过共享消息队列通信,完成实例之间的协作,形成并发操作.消息发送者将消息放入消息队列,由消息接收者自己收取,消息内容是一系列字符串组成的集合,由业务开发者自己定义,相当于确定了实例之间交互的协议.XPL 规定了消息操作的框架,定义如下:

定义 4. 实例之间通信 *Communication*=(*Notify*,*Pick*),*Notify*=(*Q_Name*,*MsgContent*,*Order*,*Clear*)向指定的消息队列发送消息.*Q_Name* 是目的消息队列名,若不存在,则新建一个消息队列.*MsgContent*={*S*}是消息内容,*S* 为一字符串,*Order* 是对队列中消息的操作顺序,*Clear* 表示是否先清空该队列.*Pick*=(*Q_Name*,*Block*,*TimeOut*,*Order*,*Condition*,*Clear*)从指定的队列接收消息.*Block* 表示若无满足条件 *Condition* 的消息,是否需要阻塞等待该消息,*TimeOut* 为等待时长.*Condition*=(*MatchField*,*MatchContent*,*MatchType*)是 3 个等长数组构成的三元组,数组 *MatchFeild* 的每个元素为待比较的 *S* 在 *MsgContent* 中的位置,数组 *MatchContent* 的每个元素是和指定的 *S* 作比较的字符串,数组 *MatchType* 的每个元素是字符串匹配策略.*Pick* 动作将符合 *Condition* 的 *MsgContent* 载入上下文.

系统限制了 *TimeOut* 的最小时长,接收消息动作 *Pick* 不能无限阻塞,同时,XPL 不存在死循环,故实例之间的并发不会出现死锁.

Flow 中的实例通信操作标签用 BNF 定义如下:

(1) 发送消息::=“⟨*Notify*⟩”,为简化起见,标签省去属性说明,完成定义 4 中的 *Notify* 功能.

(2) 接收消息::=“⟨*Pick*⟩”,为简化起见,标签省去属性说明,完成定义 4 中的 *Pick* 功能.

(3) 消息的使用::=“*getMessage*(“*field*”)”,取得业务上下文中 *MsgContent* 中的信息.*field* 为待取的字符串在 *MsgContent* 中的位置,关键字 *getMessage* 用来给标签的属性赋值.

2 XPL 与其他语言的比较

XPL 作为一种面向特定领域的语言与通用型语言如 BPEL 的差别前面已经作了说明.下面主要通过 XPL 和 CPL,CCXML,SCML 四种同样面向通信领域的流程描述语言的对比,来进一步说明 XPL 的特点.

表 1 说明了这 4 种语言的底层协议和底层通信接口模型.CPL 主要面向 SIP[9]和 H.323[10],不是与底层协议完全无关的语言.CCXML,SCML,XPL 是与底层协议无关的语言.CCXML,SCML 基于 Java 接口,XPL 主要基于 Parlay Web Services 及扩展的接口,所以,XPL 描述生成的业务更适合在融合网络的条件下运行,XPL 更适合第三方增值业务提供商用来开发业务.

Table 1 The protocol and communication interface of these language

表 1 语言的协议和通信接口模型

	The protocol	The interface model
CPL	SIP, H.323	None
CCXML	Irrespective	JTAPI (Java telephone API)
SCML	Irrespective	JCC (Java calling control)
XPL	Irrespective	Parlay Web services and its extension

表 2 说明了这 4 种语言所适用的业务种类.与其余 3 种专业语言相比,XPL 对数据类业务的丰富支持是其最为显著的特点.CPL 以呼转为设计重点,呼叫处理模型简单.JCC 相当于 JTAPI 的简化,因而 SCML 的呼叫处理模型弱于 CCXML,XPL 的呼叫处理模型相对而言弱于 CCXML 和 SCML.CCXML 可以配合 Voice XML[11]使用,Voice XML 是开发对话系统的语言,功能非常强大,XPL 也支持放音收号,能力弱于 Voice XML.总的来看,XPL 是一种面向综合性通信服务的语言,因此,它不追求那些使用频度偏低的功能,而把重点放在集成丰富实用的通信手段上.

Table 2 Application of services kinds

表 2 适用业务种类

	Calling services	UI	SMS/MMS	Web dialing	E-mail	Database services
CPL	Weak	No	No	No	Simply Yes	No Definition
CCXML	Strongest	With voice XML	No	No	No	No Definition
SCML	Stronger	No	No	Simply Yes	No	No Definition
XPL	Strong	Yes	Yes	Yes	Yes	Yes

表 3 说明了这 4 种语言在变量控制和流程控制方面的一些差异.CPL,SCML 这两种语言比较简单,不支持循环和并发操作.CCXML 虽然具有循环和并发能力,但是其没有在安全性方面作充分的考虑,不能排除业务流程出现死循环和死锁的可能.XPL 提供受到限制的循环和并发操作,在一定程度上丰富了 XPL 的流程控制能力,适于 XPL 面向领域范围较大、面对的业务需求较复杂的情况,同时又不至于带来业务开发上的安全问题.

Table 3 Control of variable and flow

表 3 变量、流程控制

	Variable operation	Switch operation	Loop operation	Concurrent operation	Dead-Loop/Dead-Lock
CPL	No	Yes	No	No	No
CCXML	Yes	Yes	Yes	Yes	Yes
SCML	No	Yes	No	No	No
XPL	Yes	Yes	Yes	Yes	No

3 支持 XPL 的业务生成系统

3.1 业务生成系统概况

我们实现了一个比较完整的支持 XPL 的业务生成系统,其主要结构如图 3 所示.该系统是国家发改委组织实施的 CNGI(中国下一代因特网)通用业务平台项目的一个子系统,另一个子系统 Web Services 网关系统屏蔽了各种异构网络的差异,向业务生成系统开放底层网络能力,该网关系统主要包括 Parlay Web Services 及其扩展接口.

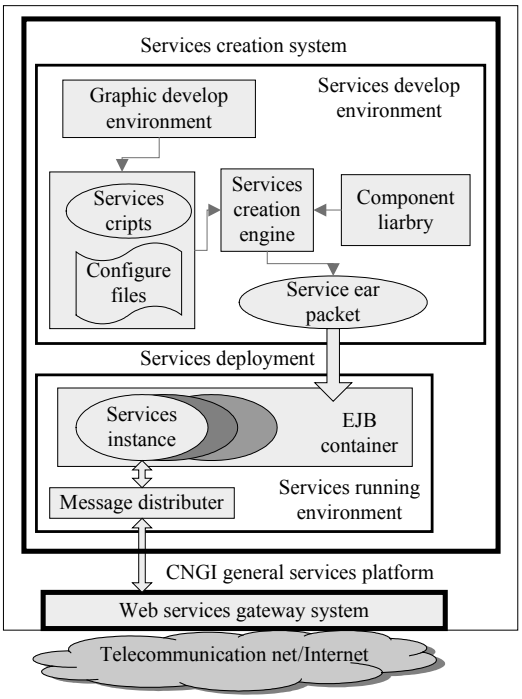


Fig.3 Structure of the services creation system

图 3 业务生成系统结构

业务开发者使用 XPL 按照需求组织业务流程,可以使用图形化的开发方式通过拖动、连接图标来组织业务流程,由图形化开发环境来转成脚本文件,也可以直接书写脚本.业务生成引擎将对脚本文件进行 XPL 的 Schema 验证,如果报错,则中断执行.此外,当需要对 XPL 进行扩展引入新的 Web Services 接口时,按照第 1.2.2 节中所述的消息分发机制来配置消息分发器,将新引入的 Web Services 接口融入系统.

业务生成引擎将 XPL 脚本转化为 EJB,在这个过程中,从构件库中加载所需的构件,最后自动打成 ear 包.ear 包部署在 EJB 容器中运行.消息分发器为网络发起的 Web Services 请求找到或发起业务实例.整个业务生成系统采用 J2EE 的体系结构,使用 JAVA 语言实现业务开发环境,以 Weblogic 作为业务运行环境的运行支撑环境,以 Eclipse 作为图形化开发环境的支撑环境.业务生成系统和 Web Services 网关系统共同构成面向服务的构架(services-oriented architecture,简称 SOA).我们目前的实现版本基于 EJB2.0,定义 1 中所述的 DB 在实现时采用 EJB 的 CMP 配置文件和 EJB-QL 组成的数据库操作脚本(参看 CMP,EJB-QL 的相关文献[12],这里不再说明).

图 4 所示的是使用图形化开发环境开发业务的情景.窗口中间部分是开发业务流程的界面,右侧部分显示的是当前可使用的标签信息.

目前,通用业务平台的 Web Services 网关系统已经接入中国联通的网络,业务生成系统生成的业务成功地在实际网络上运行,XPL 向实用化的目标迈出了重要的一步.

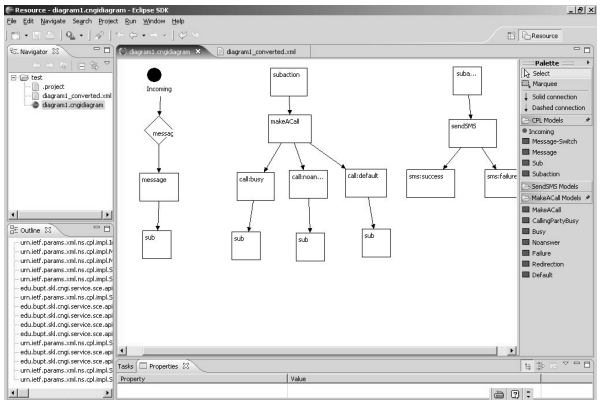


Fig.4 The graphic developing interface

图 4 图形化开发界面

3.2 关键实现技术

这里主要讲述系统核心部件即业务生成引擎的主要实现技术.业务生成技术的本质就是某种形式的软件复用技术.基于构件的软件复用是目前研究和应用的重点[13].当前,基于接口的构件组装方法讨论得较多[14−16],这种方法依靠构件直接调用其他构件的接口(或者通过所谓的连接子进行调用)完成构件的组装.XPL 这种应用场合的构件组装与一般意义上的构件组装区别在于:

(1) 业务开发者用 XPL 描述业务流程,并不直接进行构件组装,构件组装需由系统完成,系统在构件组装的实现上需要统一的方法来自动实现;

(2) 如果沿用基于接口的构件组装模式,那么,像顺序执行、选择类流程操作也必须实现在构件的内部,构件会比较复杂,可以考虑将构件中流程控制等共性的东西从构件中剥离出去,系统在完成从 XPL 到可执行代码转化的过程中再把这些共性的东西加入进去.

基于此,我们设计了一套基于上下文的构件模型和针对该模型的脚本转换——构件粘合算法.该算法按照业务脚本产生“胶水代码”,由“胶水代码”把构件粘合起来,以控制构件的执行.构件模型见定义 5.

定义 5. XPL 的标签是需求的描述单元,对应的能力实现单元是构件.每个构件都有特定的前置条件键 *P* 和后置条件键 *E*(*P* 和 *E* 也可为空).构件在运行前先从上下文中载入前置条件,运行结束后向上下文回写后置条件.构件是一个类,必须实现 4 种方法:

void *loadContext*()::=〈从上下文中加载 *P* 所对应的信息〉

boolean *willBeTrigered*()::=〈判断该标签是否满足运行的条件,实现 XPL 的选择操作〉

void *process*()::=〈实现该标签的主要业务逻辑〉

void *fillContext*()::=〈将标签运行结果写入到上下文中 *E* 所对应的地方〉

脚本转换——构件粘合算法.

1. 按照定义 2 将 XPL 脚本文件拆成多棵业务片段构成的树 *tree*,每棵 *tree* 对应一个 EJB 的 Remote 接口中的方法,该方法将用来响应 *Event* 事件(见定义 2),因为要满足各种方法之间被网络调用的次序,方法名定为从 *Flow* 的根到 *tree* 的根所经过标签的名称组成的字符串.
2. 将〈*subaction*〉进行与响应标签同样的处理,即〈*subaction*〉定义的脚本子树也拆成业务逻辑片段,与步骤 1 不同的是,对应的方法被业务自己调用,因此对应一个 Bean 类中的方法,方法名为〈*subaction*〉属性 *id* 的值.
3. 对在步骤 1、步骤 2 中形成的所有业务片段树进行一种特殊的前序遍历,在遍历的过程中把标签对应的构件用“胶水代码”粘合起来,形成业务片段对应的 EJB 方法(包括步骤 1 中形成的 EJB 的 Remote 接口中的方法和步骤 2 中形成的 Bean 类中的方法)内的代码.如下述伪码所示:

translationFromTreeToCode (*Node node*){

在 EJB 方法函数体的末尾插入以下斜体字符串形成函数体中的代码:

"*Component com=new*"+标签 node 对应的构件类的名称+"();" //生成该标签对应的构件

"*com.loadContext*();" //载入该构件的前置条件

"if(*com.willBeTrigered*()){" //判断前置条件能否触发该构件,若不能,则尝试运行下一个兄弟节点

"*ret=com.process*();" //前置条件满足,执行该构件的功能

"*ret.fillContext*();" //向业务上下文回写该构件的后置条件

"*recordRunTimeLogicTrace*("+标签的名称+");" //记录已经运行过的标签形成的轨迹

Node childNode=getnextChild(*node*); //取得 node 的下一个子节点

While (*childNode*!=null)

translationFromTreeToCode(*childNode*);

在 EJB 方法函数体的末尾插入字符串形成函数体中的代码:"}"

}

斜体部分即为将在EJB方法中形成的"胶水代码",*recordRunTimeLogicTrace* 为一种辅助方法,作用配合EJB的 Remote 接口中的方法的名称,系统在响应 *Event* 事件时就能正确选择调用业务片段.

这种基于上下文的构件组装机制使得构件本身得到简化,形式上更加规整,制作也变得容易,有利于扩展新的构件.构件组装方法体现为一种算法,与纯粹的基于接口的构件组装方法相比虽然复杂,但对业务开发者是透明的,且相对稳定,可以进行充分复用.这种构件复用模型是适用于 XPL 的一种更高抽象层次的构件复用模型.

4 XPL 应用示例

XPL 的 schema 规模较为庞大,这里列举两个业务实例来说明使用 XPL 描述业务的情况.

4.1 点击拨号业务

该实例比较简单,下面给出完整的业务脚本.业务场景是:用户从 Web 页面发起一个点击拨号业务,试图建立主叫方和被叫方的通话,若不成功,则向主叫方和被叫方发送一条短信通知;若成功,则向主叫方和被叫方发送一条彩信名片,为便于描述,彩信名片采用网络上的指定 URL.这个业务实例包含了短信和彩信操作,CPL,CCXML,SCML 均无法描述.

```
〈incoming〉
  〈makeCall callingparty="context.callingparty",calledparty="context.calledparty"〉
    〈timeout〉
        〈sub ref="failure"/〉
    〈/timeout〉
    〈succeed〉
        〈sub ref="success"/〉
    〈/succeed〉
  〈/makeCall〉
〈/incoming〉
〈subaction id="failure"〉
   〈sendSMS des="context.callingparty",content="您有一个未接通的呼叫"〉
       〈sendSMS des="context.calledparty",content="您有一个未接通的呼叫"/〉
   〈/sendSMS〉
〈/subaction〉
〈subaction id="success"〉
```

```
〈sendMMS des="context.callingparty",content="您有一张彩信名片",URL="http://*****.jpg"〉
    〈sendMMS des="context.calledparty",content="您有一张彩信名片",URL="http://*****.jpg"/〉
〈/sendSMS〉
〈/subaction〉
```

4.2 语音陪聊业务

该业务实例的脚本比较复杂,为了节省篇幅,这里只对 XPL 的描述情况作一介绍.

业务场景是:有陪聊代理多人,代理使用手机以短信的方式通知业务其开始工作,开始接受客户的电话咨询;客户拨打特服号码,业务分配一个代理为其他服务.

该业务由两个脚本构成:脚本 login.xml 用来处理代理通过短信登陆/注销系统;脚本 service.xml 用来处理用户的呼叫.代理首先通过短信发送验证码到短信特服号码,login.xml 使用数据库操作进行验证码和短信源号码(即手机号)的匹配验证:若不匹配,则向代理发送短信提示错误;若匹配,则将该手机号放入共享消息队列.当客户拨打呼叫特服号码时,service.xml 首先将用户排队.若该用户排第 1 位,service.xml 查看共享消息队列中有无空闲代理,如果有,则将当前的呼叫对所有的空闲坐席进行顺呼,若成功则进入通话状态,通话完代理挂机后业务向客户发送短信回执.由于空闲代理可以自由使用手机与客户以外的人通话,故上述转接有可能不成功,此时,业务选择一个空闲最久的代理用短信提醒他主动给该客户回电.如果该客户不是排第 1 位或者没有发现空闲的代理则向客户循环放音等待.service.xml 通过共享消息队列操作完成代理和客户的排队,避免"饥饿"现象的发生.

这项业务综合使用了呼叫、放音、短信、数据库操作等基本能力以及循环、并发等流程控制手段,并实现了代理和客户的排队.CPL,CCXML,SCML 无法完成这样的逻辑复杂、内容丰富的业务.

5 总结与展望

我们在 CNGI 通用业务平台项目的实施过程中,以 CPL 为蓝本,通过增加各种新的功能标签,引入循环、并发等手段来使 CPL 的功能不断扩充,突破了 CPL 只能描述简单的呼转业务的局限,能够描述复杂的呼叫类和数据类业务,最终发展出了基于 Web Services 的面向综合通信服务的语言 XPL.XPL 具有 CPL 所不具有的众多语言元素,实际上已经可以看作是一种独立的新语言.与其他同类的专业语言相比,XPL 集成了多种通信能力,涵盖的领域更广、流程控制手段也更加丰富和安全.但也由于 XPL 的起点(CPL)过于简单,使得某些功能(比如 CCXML 所具有的电话会议)目前在 XPL 上还没有实现.今后可以继续深入扩展和优化 XPL 的业务模型.XPL 的循环和并发控制功能还不是特别灵活,如何在保证安全的前提下加强这种功能,继续扩展 XPL 的适用范围,值得进一步深入研究.

另外,下一步的工作还包括从形式语言的角度来证明 XPL 的完备性和无二义性,可以采用的思路是证明 XPL 和其他形式化语言的等价性,比如 SDL、工作流语言等,可以通过 XPL 和这些语言之间的对比映射来证明等价性.

References:

[1] Business process execution language for Web services. version 1.1. 2002. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[2] Lennox J, Wu X, Schulzrinne H. RFC 3880, call processing language (CPL): A language for user control of Internet telephony services. 2004. http://www.ietf.org/rfc/rfc3880.txt

[3] Lennox J, Schulzrinne H. RFC 2824, call processing language framework and requirements. 2000. http://www.ietf.org/rfc/rfc2824.txt

[4] Voice browser call control: CCXML version 1.0. 2005. http://www.w3.org/TR/2005/WD-ccxml-20050629/

[5] Bakker JL, Jain R. Next generation service creation using XML scripting language. In: Proc. of the IEEE ICC 2002. New York, 2002. 2001−2007.

[6] Bakker JL, Jain R. A service creation markup language for scripting next generation network services. 2001. http://ietfreport.isoc.org/all-ids/draft-bakker-jain-scml-00.txt

[7] Parastatidis S, Webber J, Watson P, Rischbeck T. A grid application framework based on Web services specifications and practices. Technical Report, CS-TR-825, Newcastle: University of Newcastle, 2004.

[8] Foster I, Kessleman C, Nick JM, Tuecke S. The physiology of the grid—An open grid services architecture for distributed systems integration. In: Open Grid Service Infrastructure WG, Global Grid Forum. 2002. http://www.globus.org/research/papers/ogsa.pdf

[9] Rosenberg J, Schulzrinne H, Camarillo G, Johnston A, Peterson J, Sparks R, Handley M, Schooler E. SIP: Session initiation protocol. IETF RFC 3261, 2002. http://www.ietf.org/rfc/rfc3261.txt

[10] ITU-T Recommendation H.323. Packet-Based multimedia communications systems (7/03). 2003. http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-H.323-200307-P

[11] McGlashan S, Burnett DC, Carter J. Voice extensible markup language (VoiceXML) version 2.0, In: McGlashan S, Burnett DC, Carter J, Danielsen P, Ferrans J, Hunt A, Lucas B, Porter B, Rehor K, Tryphonas S, eds. W3C Candidate Recommendation 2003. 2003. http://www.w3.org/TR/voicexml20/

[12] Roman E. Mastering Enterprise JavaBeans. 2nd ed., New York: Robert Ipsen, 2002.

[13] Yang FQ, Mei H, Li KQ. Software reuse and software component technology. Acta Electronica Sinica, 1999,27(2):68−75 (in Chinese with English abstract).

[14] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Journal of Software, 2003,14(4):721−732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[15] Zhang SK, Zhang WJ, Chang X, Wang LF, Yang FQ. Building and assembling reusable components based on software architecture. Journal of Software, 2001,12(9):1351−1359 (in Chinese with English abstract).

[16] Xiang JL, Yang J, Mei H. ABC-Tool—An architecture-based component composition tool. Journal of Computer Research and Development, 2004,41(6):956−964 (in Chinese with English abstract).

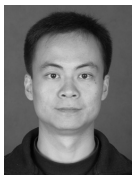
附中文参考文献:

[13] 杨芙清,梅宏,李克勤.软件复用与软件构件技术.电子学报,1999,27(2):68−75.

[14] 梅宏,陈锋,冯耀东,杨杰.ABC:基于体系结构、面向构件的软件开发方法.软件学报,2003,14(4):7−21. http://www.jos.org.cn/1000-9825/14/721.htm

[15] 张世琨,张文娟,常欣,王立福,杨芙清.基于软件体系结构的可复用构件制作和组装.软件学报,2001,12(9):1351−1359.

[16] 向俊莲,杨杰,梅宏.基于软件体系结构的构件组装工具 ABC-Tool.计算机研究与发展,2004,41(6):956−964.



杨骏(1978－),男,河南洛阳人,博士,助理研究员,主要研究领域为通信软件,融合网络业务.

温嘉佳(1979－),男,博士,主要研究领域为Web Services,动态服务组合技术.

陈俊亮(1933－),男,教授,博士生导师,中国科学院院士,中国工程院院士,主要研究领域为通信网,通信软件,网络智能化.