有色 Petri 网协作模型的 BPEL 代码实现*)

邓新国　林子禹　陈伟清　肖如良　李　玲　房丽娜
(武汉大学软件工程国家重点实验室　武汉 430072)

摘　要　Web服务业务流程执行语言(BPEL)提供了有力的技术来聚合封装的功能以及定义高质量的Web服务,然而,尽管功能强大,但是BPEL却难于使用。另一方面,有色Petri(CPNs)可以用于模拟、分析以及校验Web服务。为了在CPNs协作模型和业务流程的执行之间建立一座桥梁,给出了基于CPNs从过程模型驱动演绎出一致的BPEL代码的方法。首先,把CPNs协作模型转换为结构工作流网模型;然后,把工作流网模转换为BPEL代码,最后,通过电话机故障修理的案例研究,说明了算法的有效性。

关键词　Petri网,模型,BPEL,Web服务

Translating Colored Petri Nets Collaboration Model to BPEL

DENG Xin-guo　LIN Zi-yu　CHENG Wei-qing　XIAO Ru-liang　LI Ling　FANG Li-na
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)

Abstract　The business process execution language for Web services (BPEL) provides a powerful technology to aggregate encapsulated functionalities and define high-value Web services. While being a powerful language, BPEL is difficult to use. On the other hand, the colored Petri nets (CPNs) allow for the modeling, analysis and verification of Web services. In order to build a bridge between the CPNs collaboration model and the execution of the business processes, the CPNs based approach for process-model driven deduction of compatible BPEL code was presented. At first, the CPNs collaboration model was transformed into structured workflow net (WF-net) model. Then, the WF-net model was transformed into the BPEL code. At last, the case study of repairing telephones was provided to illustrate the applicability of the transformation algorithm.

Keywords　Petri nets, Model, BPEL, Web services

Web服务业务流程执行语言(BPEL)[1]逐渐成为Web服务实现业务过程的事实标准。许多平台支持BPEL过程的执行。有些平台也为定义BPEL过程提供了图形编辑工具。然而,这些工具直接遵循BPEL的语法而没有提供抽象能力以便于在开发的分析和设计阶段能够使用。另一方面,有色Petri网(CPNs)[2]可用于Web服务的模拟、分析和校验,以便于Web服务随后的实现。而且,许多工具也支持CPNs。为了在CPNs协作模型和业务流程的执行之间建立一座桥梁,需要给出有色Petri网协作模型的BPEL代码实现算法。

从CPNs模型到BPEL的转换应该满足一个关键需求:可读性,也就是人们能够理解生成的目标代码。这个需求很重要,由于转换后的BPEL代码可能需要求精(例如,详细说明数据运算表达式)、测试或者调试。如果BPEL只有机器能够使用而人却不能用的话,那么可以用主流的程序设计语言或者以至机器语言来取代它,但这样将会和BPEL用作服务组合中特定领域语言的目的相违背。

转换CPNs到BPEL有一定的困难,因为CPNs和BPEL本质上表示两种不同种类的语言。CPNs是图形结构的,而BPEL主要是程序块结构(虽然提供带有句法限制的图形结构)。从图形结构到程序块结构语言的映射有一定的挑战性。

本文打算处理上述需求,把CPNs转换为BPEL。这样可以利用模型驱动,基于标准工具来开发面向过程的Web服务。

接下来的各部分逐步阐明了主要思想。第1部分简单介绍了BPEL和Petri网;第2部分给出了有色Petri网协作模型的BPEL代码实现算法;第3部分是电话机故障修理的案例研究:根据需求构造了相应的CPNs协作模型,给出了经过分析、简化后,把原始的无结构CPNs模型转换为结构化的工作流网模型,详细描述了工作流网模型的BPEL代码实现,最后小结全文,略述未来的工作。

1　BPEL和Petri网

BPEL[1]本质上是一种程序设计语言,它扩展了实现Web服务的结构。BPEL过程定义和许多活动相关,这些活动是基本活动或者结构活动。基本活动对应如下的原子活动:receive,接收外部的消息;reply,返回消息给外部;invoke,调用Web服务的操作;assign,拷贝数据;throw,说明活动执行的错误;empty,空操作;wait,等待一段时间;exit,终止整个服务实例等等。为了表示复杂结构定义了如下的结构活动:sequence,定义活动执行的顺序;flow,并行的活动;switch,条件分支的活动;pick,基于互斥条件的时间或者外部触发活动;while,循环结构;还有一些本文没有用到的活动等等。

目前的实践表明BPEL的抽象能力不适合业务过程的分析、设计。业务过程的分析、设计一般依靠像UML[3]和

*)国家自然科学基金(No. 90604005)。邓新国　博士生,研究方向为软件工程和Web服务;林子禹　博导,研究方向为软件工程和语义Web。

CPNs这样的“高级语言”。因此，从需求建模语言（例如CPNs）到业务流程执行语言BPEL的映射是必要的。

Petri网相关的形式化定义[4]如下：

定义1 （Petri net）

一个三元组$PN=(P,T;F)$是一个Petri net，当且仅当：

(1)$P\cap T=\Phi$；

(2)$P\cup T\neq\Phi$；

(3)$F\subseteq(P\times T)\cup(T\times P)$（“×”为笛卡尔积）；

(4)$dom(F)\cup cod(F)=P\cup T$，其中：

$dom(F)=\{x|\exists y:(x,y)\in F\}$

$cod(F)=\{y|\exists x:(x,y)\in F\}$

它们分别为F的定义域和值域。

其中P和T分别称为PN的库所（place）集和变迁（transition）集，F为流关系（flow relation）。

定义2 （WF-net）

Petri net $PN=(S,T,F)$是工作流网（WF-net），当且仅当：

(1)PN有一个源库所（source place）$i\in P$，使得$i=\Phi$；

(2)PN有一个漏库所（sink place）$o\in P$，使得$o=\Phi$；

(3)每个节点$x\in P\cup T$都属于从i到o的一条路径上。

定义3 （Coloured Petri Nets）

$\sum=(P,T;F,C,I_-,I_+,M_0)$称为有色Petri网，当且仅当：

(1)$(P,T;F)$为Petri网，称为$\sum$的基网；

(2)$C:P\cup T\to\rho(D)$，$\rho(D)$为颜色集D之幂集合，使得：对$p\in P$，$C(p)$是库所p上所有可能的托肯色（资源类）之集合，对$t\in T$，$C(t)$是变迁t上所有可能的出现色之集合；

(3)I_-和I_+分别是$P\times T$上的负函数和正函数，使得对所有$(p,t)\in P\times T$有：$I_-(p,t)\in[C(t)_{MS}\to C(p)_{MS}]_L$，且$I_-(p,t)=0$的充分必要条件是$(p,t)\notin F$，$I_+(p,t)\in[C(t)_{MS}\to C(p)_{MS}]_L$，且$I_+(p,t)=0$的充分必要条件是$(t,p)\notin F$。

(4)$M_0:P\to D_{MS}$，称为$\sum$的初始标识，它必须满足条件$\forall p\in P:M_0(p)\in C(p)_{MS}$，即$M_0(p)$是$p$的托肯色集合上的多重集。

2 有色Petri网协作模型的BPEL代码实现算法

根据需求构造的有色Petri网协作模型含有丰富的数据类型，并且可能是非结构的。抽取数据类型并且改变结构后转换为结构的工作流网模型。工作流网模型生成的BPEL代码求精时可以添加这些数据类型。

工作流网模型的BPEL代码实现方法的基本思想：WF-net的变迁对应于BPEL的原子活动，各种子结构S映射为BPEL相应的结构代码，并且WF-net中的子结构S简化为单个变迁t_S。这个过程一直重复到WF-net中只有一个变迁，这时候就完成了整个WF-net的BPEL代码实现。

生成的BPEL代码的可读性和易维护性很重要。如果形成的BPEL代码太复杂或者很难懂，那么它就不容易被扩展或者定制，因此，我们设法把WF-net的各个部分映射到BPEL最适合的结构，并且在BPEL的众多结构中首选“sequences”，“switch”，“pick”，“while”和“flow”等结构。我们使用迭代的方法把WF-net的各个部分用适合的BPEL结构来封装，下面是具体的算法：

算法 有色Petri网$\sum=(P,T;F,C,I_-,I_+,M_0)$

(1)有色Petri网$\sum$转换为结构WF-net PN = (S,T,F)。

(2) X := PN。

(3) 当X只包含一个变迁，那么转(9)。

(4.1)如果有最大的SEQUENCE结构$S\in[X]$，那么选择S，转(7)。

(4.2) 如果有SWITCH，PICK和WHILE结构中的一种结构$S\in[X]$，那么选择S，转(7)。

(4.3) 如果有最大的FLOW结构$S\in[X]$，那么选择S，转(7)。

(5) 如果有其它的BPEL结构$S\in[X]$，选择S，转(7)。

(6) 选择结构$S\in[X]$，把S映射为BPEL。

(7) 把S转换的BPEL代码添加到单个变迁t_S。

(8) $X:=(X-S)\cup t_S$，返回(3)。

(9) 输出X的BPEL代码。

首先，有色Petri网$\sum$转换为结构WF-net X，如果X只包含一个变迁，那么输出X的BPEL代码；否则，如果有最大顺序结构S，那么选择S，把S映射为BPEL，将S转换的BPEL代码添加到单个变迁t_S，用t_S替换S。先选择顺序结构可以让转换工作尽可能简洁。如果X中没有顺序结构，那么接着考虑其它结构，如果有SWITCH，PICK和WHILE结构中的一种结构S，选择S，进行和顺序结构类似的转换、替换工作；最后考虑最大的FLOW结构。不是所有的WF-net只包含上述结构，如果还有其它的BPEL结构，也进行类似的工作。如果还有BPEL中没有的结构，那么只有人工转换。

3 案例研究：电话机故障修理

协作的例子[5]是关于某个公司修理电话机的过程。这个公司可以修理三种不同类型的电话机（“\T1”，“\T2”和“\T3”）。过程开始时客户登记电话机，接着电话机被送往故障检测部门分析，然后故障被分类（总的来说，这个公司可以修理十种类型的电话故障），一旦确定故障，电话机被送往修理部门，并且写信告知客户电话机的毛病。修理部门有两组：一组修理简单故障，另外一组修理复杂故障。然而，有些种类的故障这两组都能够修理。一旦职员完成修理，电话机被送往质量评价部门检查是否已经修好，如果没有修好，电话机重新被送往修理部门；否则存档修理情况并且把电话机返回给客户。为了节省生产时间，公司对于电话机的某个故障仅仅设法修理有限次数。如果某个话机故障超过限定次数仍然没有修理好，那么存档这种故障，并且返回给客户一部崭新的电话机。

3.1 有色Petri网协作模型

有色Petri网由规范的图形过程描述语言组成，这种语言结合了图形的直观表示和行为的形式语义这两者的优点。有色Petri网包含由弧连接的静态结构（库所，用椭圆表示）和动态结构（变迁，用矩形表示）。库所描述了系统的状态，变迁描述了系统的行为。弧表达式说明有色Petri网在变迁发生的时候是如何改变状态的。每条弧包含名为托肯（tokens）的标识符，这些托肯有给定类型的数据值。

“修理案例”经过简化假设和抽象后，利用CPN[6]工具构造的有色Petri网模型如图1所示。

库所（place）中托肯（tokens）的种类取决于库所的颜色集（colour set）。CPNs模型的颜色集类似于程序设计语言的数据类型，颜色集的值称作颜色（colours）。库所的颜色集一般写在库所的附近，并且用标准元语言（Standard ML）[7]来声明，例如，图1中库所product的颜色集为PHONE，图2列出了图1中模型使用的颜色集，函数以及变量声明。

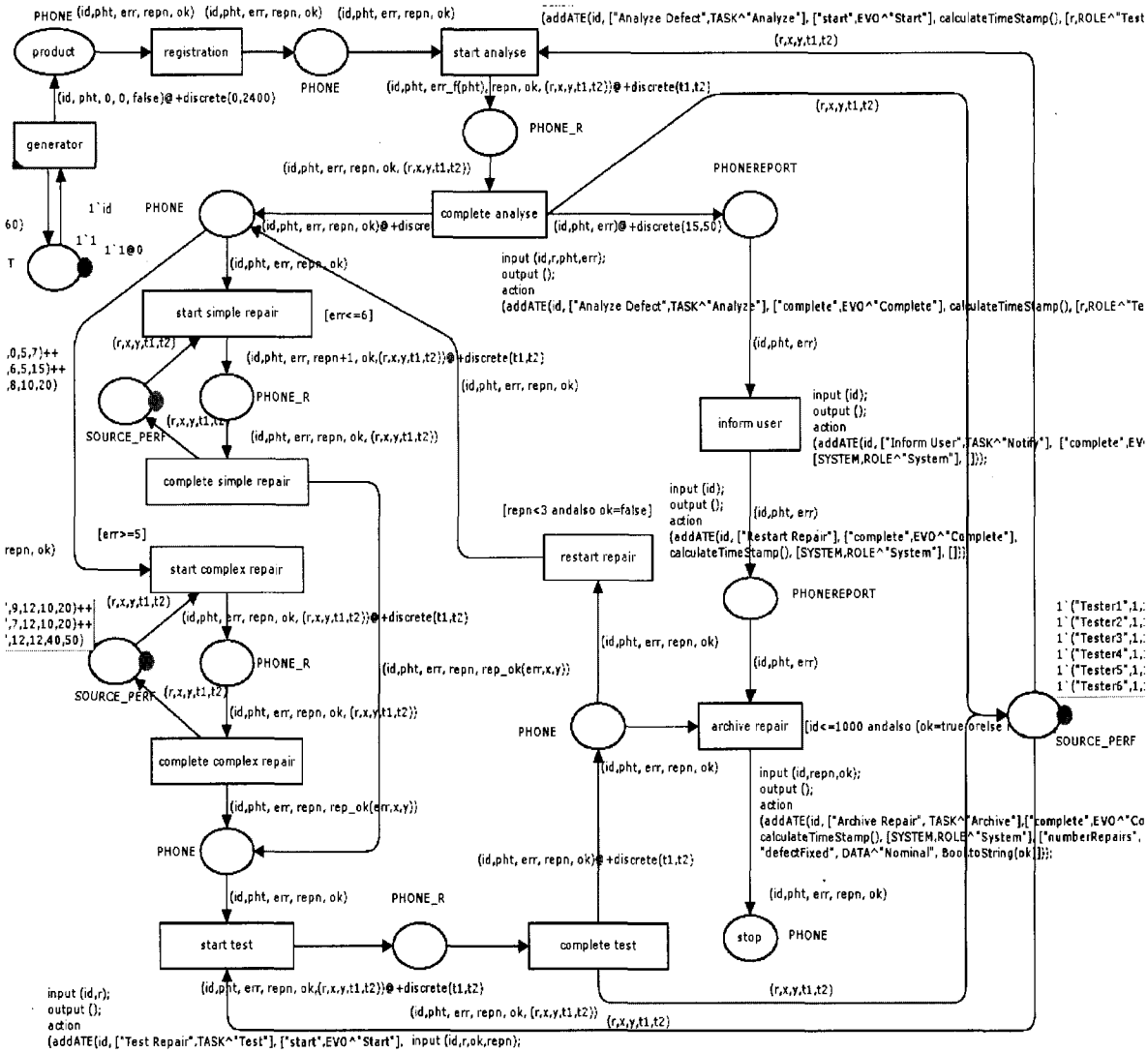

图 1　修理案例的有色 Petri 网模型

```
colset UNIT = unit;
colset INT = int timed;
colset BOOL = bool;
colset STRING = string;
colset ERRORTYPE = int with 0..10;
colset NREPAIRS = int;
colset RESOURCE = string;
colset SOURCE_PERF = product STRING * INT * INT * INT;
colset PHONET = subset STRING with ["T1", "T2", "T3"];
colset PHONE = product INT * PHONET * ERRORTYPE * NREPAIRS * BOOL timed;
colset PHONE_R = product INT * PHONET * ERRORTYPE * NREPAIRS * BOOL * SOURCE_PERF timed;
colset PHONEREPORT = product INT * PHONET * ERRORTYPE timed;
colset ERRT1 = int with 1..10;
colset ERRT2 = int with 6..10;
colset ERRT3 = int with 1..8;
colset AN_TEST = subset STRING with ["Tester1", "Tester2", "Tester3", "Tester4", "Tester5", "Tester6"];
var id, x, y,t1,t2: INT;
var repn: NREPAIRS;
var phone: PHONE;
var pht: PHONET;
var err: ERRORTYPE;
var ok: BOOL;
var r: RESOURCE;
val SYSTEM = "System";
var tester: AN_TEST;
fun err_ff(t:PHONET) = if t="T2" then ERRT2.ran() else ERRT3.ran() ;
fun err_f(t:PHONET) =
  if (discrete(1,5)<=3 orelse t="T1") then ERRT1.ran() else err_ff(t);
fun rep_ok(err:ERRORTYPE, x,y:INT) = if (discrete(x,y)>=err) then true else false;
val FILE = "log/repairExample";
val FILE_EXTENSION = ".cpnxml";
val SUPER="http://www.jp-super.org/ontologies";
val EVO=SUPER^"EVO#";
val TASK=SUPER^"TASK#";
val ROLE=SUPER^"ROLE#";
val DATA=SUPER^"DATA#";
```

图 2　颜色集，函数和变量

3.2　把有色 Petri 网协作模型转换为结构工作流网模型

在 WF-net [4] 中，变迁表示活动，库所表示这些活动的前、后置条件(pre- and post-conditions)。WF-net 有一个输入库所 i 和一个输出库所 o，因为 WF-net 表示的任何案例，当它进入工作流管理(WFM)系统时会被创建，而一旦由系统处理完，就会被删除，即 WF-net 明确描述了案例的生命周期，详细说明了案例的动态行为。

WF-net 是没有数据、层次、时间和其它扩展的经典 Petri net，因此它的应用有限，然而，我们可以使用 WF-net 作为理论基础，通过它来简化 CPNs，方法是抽象 CPNs 的数据流，捕获 CPNs 的控制流结构。

为了产生 BPEL 模板代码，需要把无结构的 CPNs 模型转换为结构的 WF-net 模型，这可以通过抽象 CPNs 的数据，改变它的结构来实现。图 3 所示是把图 1 中修理案例的原始无结构有色 Petri 网转换成了结构的工作流网。

3.3　把工作流网模型转换为 BPEL 代码

使用第 2 节中的算法可以产生图 3 中 WF-net 的 BPEL 代码。首先，查找最大的 SEQUENCE 结构。图 3 的四个最大 SEQUENCE 结构 S 可以分别用一个变迁 t_S 代替，并且把 S 转换的 BPEL 代码(即，顺序活动)添加到相应的替换变迁 t_S。图 3 中的第一个顺序结构由变迁 Registercomplete，AnalyzeDefect(AnalyzeDefectstart 和 AnalyzeDefectcomplete)组成，我们使用变迁 Fragment0complete 替换这个结构。类似地，替换剩下的顺序结构后产生了三个相应的新变迁 Repair(Simple)，Repair(Complex) 和 TestRepair 。经过这些替换后现在再没有顺序结构了。接着，我们使用变迁 Repair 替换变迁 Repair(Simple) 和 Repair(Complex)组成的分支结构。由于变迁 Repair 的引入，它和变迁 TestRepair 又组成了新的

顺序结构，显然，它是最大顺序结构，因此我们用变迁 Fragment5complete 替换这个顺序结构。图 3 简化顺序和分支结构后的 BPEL 代码片段和工作流网，分别在清单 1、清单 2 和图 4 中显示。清单 2 中假设两个分支的条件分别为“expr1”和“expr2”。

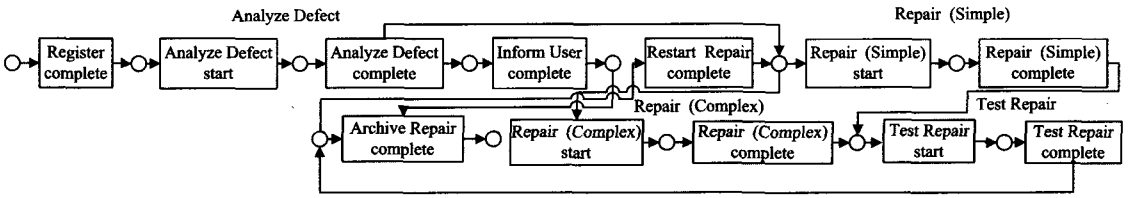

图 3　原始无结构有色 Petri 网的结构工作流网

清单 1　片段 Fragment0complete

```
〈sequence name= “Fragment0complete”〉
  〈invoke name= “Registercomplete”/〉
  〈invoke name= “AnalyzeDefectstart”/〉
  〈invoke name= “AnalyzeDefectcomplete”/〉
〈/sequence〉
```

清单 2　片段 Fragment5complete

```
〈sequence name=“Fragment5complete ”〉
  〈switch name=“Repair ”〉
    〈case condition =“decision = &quot;expr1&quot; ”〉
      〈sequence name= “Repair(Simple)”〉
        〈invoke name= “Repair(Simple)start”/〉
        〈invoke name= “Repair(Simple)complete”/〉
      〈/sequence〉
    〈/case〉
    〈case condition =“decision = &quot;expr2&quot; ”〉
      〈sequence name= “Repair(Complex) ”〉
        〈invoke name= “Repair(Complex)start”/〉
        〈invoke name= “Repair(Complex)complete”/〉
      〈/sequence〉
    〈/case〉
  〈/ switch〉
  〈sequence name=“TestRepair ”〉
    〈invoke name= “TestRepairstart”/〉
        〈invoke name= “TestRepaircomplete”/〉
    〈/sequence〉
〈/sequence〉
```

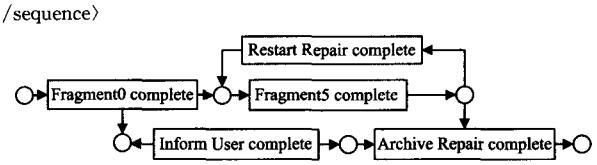

图 4　简化顺序和分支结构后的工作流网

图 4 的工作流网中有个以变迁 Fragment5complete 开始、变迁 ArchiveRepaircomplete 结束的 WHILE 循环结构，这个循环结构可以替换成如清单 3 所示带有片断代码的单个变迁 RepairTestcomplete。其中，假设循环成立的条件是满足“expr3”而不满足“ expr4”。替换 WHILE 结构后的工作流网如图 5 所示。

清单 3　片段 RepairTestcomplete

```
〈sequence name= “RepairTestcomplete ”〉
  〈invoke name=“Fragment5complete ”/〉
  〈while condition=“expr3 and ! expr4”〉
    〈invoke name=“RestartRepaircomplete ”/〉
    〈invoke name=“Fragment5complete”/〉
  〈/while〉
〈/sequence〉
```

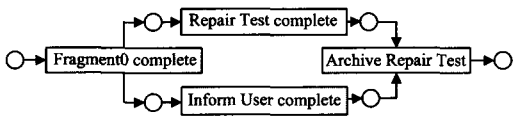

图 5　简化 while 循环结构后的工作流网

图 5 所示的工作流网是并行结构，可以映射为 BPEL 的 flow 结构，因此这个网的 BPEL 代码实现简单明了。

清单 4　案例的 BPEL 代码

```
〈flow name=“complete”〉
〈links〉
    〈link name=“ Fragment0complete _ RepairTestcomplete ”/〉
    〈link name=“ Fragment0complete _InformUsercomplete”/〉
    〈link name=“ RepairTestcomplete _ArchiveRepaircomplete”/〉
    〈link name=“ InformUsercomplete_ ArchiveRepaircomplete ”/〉
    〈/links〉
    〈 invoke name=“ Fragment0complete ”〉
      〈source linkName = “ Fragment0complete_ RepairTestcomplete ”/〉
      〈source linkName = “Fragment0complete _ InformUsercomplete””/〉
      〈〈 Fragment0complete 〉〉
    〈/ invoke 〉
    〈invoke name=“ RepairTestcomplete ”〉
      〈target linkName = “ Fragment0complete _ RepairTestcomplete ”/〉
      〈source linkName=“RepairTestcomplete _ArchiveRepaircomplete”/〉
    〈〈 RepairTestcomplete 〉〉
    〈/invoke〉
    〈invoke name=“ InformUsercomplete ”〉
      〈target linkName = “ Fragment0complete _InformUsercomplete”/〉
      〈source linkName = “ InformUsercomplete_ ArchiveRepaircomplete ”/〉
    〈/invoke〉
    〈invoke name = “ ArchiveRepaircomplete ” joinCondition = “All”〉
      〈target linkName=“ RepairTestcomplete _ArchiveRepaircomplete”/〉
      〈target linkName=“ InformUsercomplete_ArchiveRepaircomplete ”/〉
    〈/invoke〉
    〈/flow〉
```

清单 4 和图 6 分别显示了最后简化步骤后的 BPEL 代码和工作流网图形。转换后最终结果的工作流网只含有一个变

迁 complete，即整个过程的 BPEL 代码注释为 complete，因此，我们提供的迭代方法可以把工作流网转换为可读的 BPEL 模板代码。

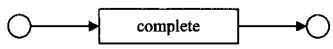

图 6 简化流(flow)结构为一个变迁后的工作流网

结束语 本文给出了有色 Petri 网协作模型的 BPEL 代码实现算法：首先，把有色 Petri 网协作模型转换为结构 WF-net 模型；接着，使用归纳法把 WF-net 模型的子结构替换成标注有 BPEL 代码的变迁。

在案例研究中运用转换算法，首先根据电话机故障修理的需求，创建了该案例的有色 Petri 网协作模型；接着经过分析和简化之后把原始的有色 Petri 网协作模型转换为结构 WF-net 模型；最后，实现了 WF-net 模型的 BPEL 代码实现，显示了转换算法的实用性。

由于主要的目标是产生可读的、易维护的 BPEL 代码，因此没有针对复杂的完全转换(例如，没有使用事件处理结构而是识别典型的 BPEL 结构)。

未来的工作将引入辅助变量或者借鉴程序设计方法学，把无结构的 WF-net 模型转换为 BPEL 代码。

参 考 文 献

[1] Arkin A，Askary S，Bloch B，et al. Web Services Business Process Execution Language Version 2.0. Working Draft. WS-BPEL TC OASIS，May 2005

[2] Jensen K. An Introduction to the Theoretical Aspects of Coloured Petri Nets. http://www.daimi.au.dk/~kjensen/papers_books/rex.pdf

[3] Object Management Group. Unified Modeling Language(UML) 2.1.1. http://www.omg.org/technology/documents/formal/uml.htm

[4] 袁崇义. Petri 网原理与应用[M]. 北京：电子工出版社，2005

[5] http://is.tm.tue.nl/~cgunther/dev/prom/

[6] CPN Group，University of Aarhus，Denmark. CPN Tools Home Page. http://wiki.daimi.au.dk/cpntools/

[7] Ullman J D. Elements of ML Programming. Prentice-Hall，1998

(上接第 235 页)

[2] Bertino E，Ferrari E，Atluri V. The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information and System Security，1999，2(1)：65-104

[3] Wainer J，Barthelmess P，Kumar A. W-RBAC-A workflow security model incorporating controlled overriding of constraints. International Journal of Cooperative Information Systems，2003，12(4)：455-486

[4] Enhydra Shark. http://www.enhydra.org/workflow/shark/index.html

[5] Fitzpatrick G. The Locales Framework：understanding and Designing for Cooperative Work. Australia：Ph.D. Thesis. The Univ. of Queensland，1999

[6] Fitzpatrick G，Mansfield T，Kaplan S. Locales Framework Exploring foundations for collaboration support // IEEE Proceedings of OzCH1. Hamilton，New Zealand，1996：34-41

[7] Crampton J. A reference monitor for workflow systems with constrained task execution // Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies (SACMAT 2005). Stockholm，Sweden，2005：38-47

[8] Crampton J. An algebraic approach to the analysis of constrained workflow systems // Proceedings of the 3rd Workshop on Foundations of Computer Security. Turku，Finland，2004：61-74

[9] Jboss jBMP. http://www.jboss.com/products/jbpm

[10] Yale CAS. http://www.ja-sig.org/products/cas/index.html

(上接第 250 页)

[2] Henricksen K，Indulska J. A Software Engineering Framework for Context-aware Pervasive Computing // Pervasive Computing and Communications，2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference. 2004：77-86

[3] Brown P J，Jones G J F. Context-aware retrieval：exploring a new environment for information retrieval and information filtering. Personal and Ubiquitous Computing，2001，5(4)：253-263

[4] Pascoe J. The stick-e note architecture：extending the interface beyond the user // Proceedings of International Conference on Intelligent User Interfaces. Orlando，USA，1997：261-264

[5] Ranganathan A，Chetan S. Olympus--A High-level Programming Model for Pervasive Computing Environments // Pervasive Computing and Communications，2005. PerCom 2005. Third IEEE International Conference. 2005：7-16

[6] Keays R，Rakotonirainy A. Context-oriented Programming // Proceedings of the 3rd ACM International Workshop. San Diego，CA，USA，2003

[7] Rakotonirainy A. Context-oriented Programming for pervasive space // Proceedings of the ACM Dynamic Languages Symposium. 2005

[8] Román M，Hess C，Cerqueira R，et al. Gaia：A middleware infrastructure to enable active space. IEEE Pervasive Computing Magazine，2002，1：74-83

[9] Capra L，Emmerich W，Mascolo C. CARISMA：Context-aware Reflective Middleware System for Mobile Applications. IEEE transactions on software engineering，2003

[10] Capra L，Emmerich W，Mascolo C. A Micro-economic Approach to Conflict Resolution in Mobile Computing // SIGSOFT. 2002

[11] Arntzen I M. A Programmable Structure for Pervasive Computing // Proceedings of the IEEE/ACS International Conference on Pervasive Services. 2004