# 基于翻译模式的 BPEL 到 LOTOS 映射方法研究*

赵会群，何　霞†
（北方工业大学 信息工程学院，北京 100041）

**摘　要**：为了实现由 BPEL 描述的 Web 服务组合到 LOTOS 的自动化转换，提出一种基于翻译模式的转换算法。从 BPEL 语言的 XML schema 定义出发，分析 BPEL 本身的语言结构，得到 BPEL 语言的产生式。再根据 BPEL 到 LOTOS 的语义映射规则，设计 BPEL 到 LOTOS 的翻译模式。同时，在基本映射规则之上，给出 BPEL 到 LOTOS 的数据类型和故障处理机制的转换规则。最后结合 Web 服务实例，验证该工具的可行性。

**关键词**：Web 服务组合；BPEL；LOTOS；翻译模式；形式化验证；映射规则
**中图分类号**：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2013)06-1751-05
doi:10.3969/j.issn.1001-3695.2013.06.039

# Study on translation model based mapping method from BPEL to LOTOS

ZHAO Hui-qun, HE Xia†
(*School of Information Engineering, North China University of Technology, Beijing* 100041, *China*)

**Abstract**: In order to achieve the automated conversion from Web service composition described by BPEL to LOTOS, this paper presented a conversion algorithm based on the translation model. Firstly, starting from the the BPEL language of XML schema definition, it analyzed BPEL language structure and got the BPEL language production. Then, depending on BPEL to LOTOS semantic mapping rules, it designed BPEL to LOTOS translation mode. Meanwhile, above the basic mapping rules, it gave transformation rules of BPEL to LOTOS data types and fault handling mechanisms. Finally, it verified the feasibility of the tool by using a Web service instance.

**Key words**: Web service composition; BPEL; LOTOS; translation model; formal verification; mapping rules

BPEL[1]（business process execution language，业务流程执行语言）定义了如何表示业务流程中的活动，是实现 Web 服务组合的重要途径。由于服务协同的动态性、网络运行环境的异构性，以及松耦合的服务开发模式，导致 Web 服务组合流程的开发和运行过程的不确定性。Web 服务组合验证能提前发现 Web 服务组合中存在的缺陷，弥补了 Web 服务测试的不足，提高了 Web 服务组合执行的成功率，增强了用户满意度，对 Web 服务组合的正确运行具有十分重要的意义。

近年来，业界对于 Web 服务组合验证问题已有了一些研究，主要分为 Petri 网[2]、进程代数[3]和自动机理论[4]三类。而为了验证 Web 服务组合的有效性，文献[5]提出了一种基于概率模型检测的 Web 服务组合验证方法。从上述研究方法看，现有的工作大都集中采用某种形式化方法对 Web 服务组合进行建模。其主流思想是针对被验证属性选择自动验证工具（通常是模型检测工具），提出该工具输入语言的形式模型，然后通过 BPEL 到形式模型以及形式模型到输入语言的转换，实现属性的自动验证。本文也采用同样的验证方法。形式化描述语言 LOTOS[6~9]（language of temporal ordering specification），是一种面向协议验证，抽象级别高，其描述的协议较容易转换成 CCS 模型、FSM 模型和 Petri 网模型。由此，使用 LOTOS 对 BPEL 描述的服务组合进行建模，并且使用模型检测工具——CADP[10]工具集中的模型检测器 evaluator 对建立的模型进行验证，从而判断服务组合流程是否正确[11]。

由于 LOTOS 本身缺乏状态概念，不易直接自动生成实现代码，而对于每一个 BPEL 描述的 Web 服务组合，都手动地去编写 LOTOS 文件，其工作量可想而知。这就要求相关人员对 BPEL 和 LOTOS 语言本身有较深入的掌握。文献[11]给出了 BPEL 到 LOTOS 的基本映射规则，但却未对 BPEL 到 LOTOS 的自动转换作进一步的研究。文献[12]同样在上述映射规则上进行了 BPEL 到 LOTOS 的自动转换，但是只考虑了常用的组合方式，而 BPEL 是动态组合 Web 服务的，其组合情况千变万化，所以该工具具有一定的局限性。针对以上问题，本文在 BPEL 到 LOTOS 的基本语义映射规则基础上，给出了数据类型和故障处理机制的转换规则，并且从分析 BPEL 的 XML schema 定义出发，提出一种基于翻译模式的转换算法，即采用编译原理中翻译模式的思想实现 BPEL 到 LOTOS 的自动化转换。

## 1　BPEL 到 LOTOS 的映射规则

BPEL 活动包括基本活动和结构化活动。基本活动主要包括接收/发送消息、赋值、调用服务等。结构化活动通过整合基本活动来描述复杂业务流程，包括顺序、选择、循环控制流方式、并发等。LOTOS 行为包括选择、循环、并行、使能、中止等，

**收稿日期**：2012-09-02；**修回日期**：2012-11-01　　**基金项目**：国家自然科学基金资助项目(61070030,61111130121)；北京市市属高校学术创新团队资助项目(PHR201107107)

**作者简介**：赵会群(1960-)，男，教授，博士，主要研究方向为服务计算；何霞(1987-)，女(通信作者)，硕士研究生，主要研究方向为软件测试(hexia_40603@126.com).

且可使用变量声明和数值声明进行进程间通信。BPEL 到 LOTOS 的映射除了基本活动映射规则[11~13]外,还包括数据类型、故障处理映射规则等。

### 1.1 数据类型映射

BPEL 数据类型是基于 XML schema(XSD)的。BPEL 中的变量对应着 WSDL[14](Web services description language)中定义的消息类型。LOTOS 除了基本数据类型(如 natural、char 等)外,还可自定义数据类型。自定义数据类型包括数据域和这些数据域上的操作,这些操作一般为创建该数据类型、增加一个数据元素和取出该数据类型的数据元素等。所以,将 BPEL 中的变量映射为 LOTOS 中的变量时,需要将 BPEL 中变量对应的消息类型映射为 LOTOS 自定义数据类型 。

首先,从 BPEL 本身的 WSDL 和 BPEL 所调用服务的 WSDL 中分析出 BPEL 变量的具体结构,即数据成员。其过程如图 1 所示。

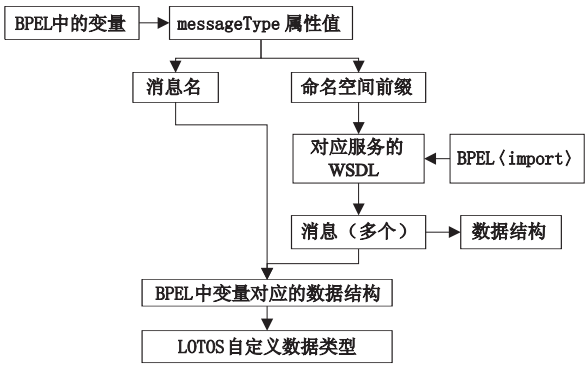


**图1 分析BPEL变量结构过程**

根据分析得到的 BPEL 变量结构,创建 LOTOS 自定义数据类型。每一个自定义数据类型是一个后缀为. lib 的文件。该数据类型包括构造数据类型操作和取数据操作两类操作。

构造数据类型操作:名字 = "cons" + 数据类型名。

取数据操作:名字 = "extract" + 数据类型名 + index(从 1 开始)。

其中,index 为数据成员在该数据类型中的下标。

例 1 BPEL 中一变量定义语句,如〈bpel:variable name = "request" messageType = "tns:getLoanQuoteRequest" /〉,相应有 LOTOS 数据类型文件 GETLOANQUOTEREQUEST. lib。该数据类型定义如下:

```
type GETLOANQUOTEREQUEST is NATURAL
sorts GetLoanQuoteRequest
opns
    consGetLoanQuoteRequest( * ! constructor * ):
      Nat- > GetLoanQuoteRequest
    extractGetLoanQuoteRequest1 :
      GetLoanQuoteRequest - > Nat
eqns
    forall housenumber : Nat
ofsort Nat
    extractGetLoanQuoteRequest1 ( consGetLoanQuoteRequest ( housenumber) ) = housenumber;
endtype
```

### 1.2 故障处理机制

当 BPEL 中出现故障时,BPEL 中的其他活动将被终止。而故障通过接收 WSDL 定义的故障消息来产生,或者通过使用 throw 元素被明确触发。BPEL 使用〈faultHandlers〉活动来定义 BPEL 中的故障处理。〈faultHandlers〉下包括 0 个或多个的〈catch〉或〈catchall〉子活动。每一个〈catch〉捕获一种故障,并进行相应的故障处理。〈catchall〉元素提供默认的错误处理活动。由此,将〈faultHandlers〉下的每一个子节点映射为一个 LOTOS 故障处理进程。在 LOTOS 中,设计当服务返回值全为 0 时,即捕获到故障,调用相应的故障处理进程。

BPEL 映射为 LOTOS 的故障处理[13]过程如图 2 所示。

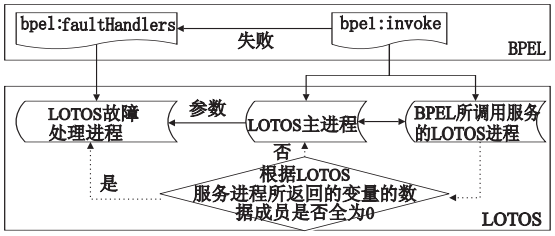


**图2 LOTOS故障处理过程**

### 1.3 文档结构映射

BPEL 中各个 Web 服务之间的交互就是 LOTOS 中各个运行着的进程之间的相互作用。BPEL 所构建的服务系统由客户、BPEL 流程、调用的服务组成。BPEL 流程从接收客户的请求开始,通过调用其他服务来完成客户的请求,并最终将结果返回给客户。将 BPEL 映射为 LOTOS,则需要在 LOTOS 中构建客户端进程、BPEL 流程主进程和 BPEL 所调用服务进程。

LOTOS 文件框架[11]如下:

```
specification specificationName[ gates 列表] :noexit
  library
    ( * 引入需要的 library * )
  endlib
behaviour
  Client[ gates 列表]
  |[ gate] |
    MainProcess[ gates 列表]
  |[ gates 列表] |
  (
  WebServiceP1[ gate]
  |||
  …
  )
where
process MainProcess[ gates 列表] :noexit : =
  ( * 行为 * )
endproc
…
endspec
```

其中:library 部分引入 LOTOS 中所用到的基本数据类型和自定义数据类型;behavior 部分为主行为部分,其实例化客户端进程、BPEL 流程主进程和 BPEL 所调用服务进程,以并行和同步的方式表示 Web 服务之间的交互;where 部分定义了客户端进程、BPEL 流程主进程和 BPEL 所调用服务进程的行为;而 BPEL 流程主进程则是整个 BPEL 流程的体现,定义了 BPEL 调用服务进程的逻辑。

## 2 BPEL 到 LOTOS 的翻译模式设计及转换算法

在 BPEL 到 LOTOS 映射规则基础上,从编译原理[15,16]的角度出发,提出基于翻译模式的转换算法,可一定程度地保障

BPEL 到 LOTOS 转换的通用性。本章首先分析 BPEL 的 XML schema 定义,得到 BPEL 产生式;再根据需要转换得到的 LOTOS 文档分析产生式符号的属性,根据映射规则设计每个产生式的动作,得到翻译模式;最后根据翻译模式设计 BPEL 到 LOTOS 的转换算法。

### 2.1　BPEL 产生式

根据 Web services business process execution language version 2.0 规范,分析 BPEL 主要活动及其子活动的 XML schema 定义,得到 BPEL 产生式如表 1 所示(注:这里只列出了部分产生式)。

表 1　BPEL 产生式

| BPEL 产生式 |
|---|
| bpel→process |
| process→activity |
| activities→activity activities1 \| ε |
| activity→receive |
| \| reply |
| \| invoke |
| \| assign |
| \| sequence |
| \| flow |
| \| if |
| \| while |
| \| pick |
| sequence→activities |
| flow→links activities |
| if→condition activity elseifs else |
| while→condition activity |
| pick→onMessage onAlarm |

其中:bpel 为产生式开始符号,非终结符号 process 为 BPEL 中的〈process〉活动。为了表示 BPEL 活动下可以有任意多个其他活动,这里引入非终结符 activities。由 activities 可以推导出 activity 的任意组合(至少有一个 activity)。而非终结符 activity 代表任意一个 BPEL 活动。对于 BPEL 中的基本活动还应具有由相应非终结符推导出属性信息的产生式,如 receive→partnerLink portType operation。

对于 BPEL 中的结构化活动,这里以〈if〉活动为例,说明构造结构化活动产生式的方法。根据 XML schema 定义,〈if〉活动的直接子节点包括一个〈condition〉、一个 activity、0 个或多个〈elseif〉和 0 个或 1 个的〈else〉。为了表示可以有 0 个或多个〈elseif〉,引入非终结符号 elseifs,即有产生式 elseifs→elseif elseifs1 \| ε。所以,〈if〉活动的产生式如表 2 所示。

表 2　〈if〉活动产生式

| if→condition activity elseifs else |
|---|
| elseifs→elseif elseifs1 \| ε |
| elseif →condition activity |
| else→activity \| ε |

### 2.2　翻译模式

从 LOTOS 的文档框架可以看出,要得到完整的 LOTOS 文档需要知道 specification、client、mainProcess,以及调用服务时创建的 WebServiceP 进程的名字、门和行为。可将上述四个对象作为 BPEL 产生式的属性。Specification 对应 BPEL 中的〈process〉活动,它的门和行为由 BPEL 中的其他基本活动决定。由此,BPEL 产生式中非终结符 process 有继承属性 s(表示 LOTOS 文档的 specification 对象)。当检测到 createInstance = "yes"的〈receive〉活动时,系统创建 BPEL 程序实例,开始分析 BPEL 流程(这里暂且先只考虑开始活动〈receive〉),而 LOTOS 中表示 BPEL 流程的 mainProcess 进程对象和客户端 client 进程对象也应在此时创建,即 createInstance = "yes"的 BPEL 非终结符 receive 具有综合属性 client(表示 client 进程对象)和 main(表示 mainProcess 进程对象)。当使用〈invoke〉调用服务时,此时应创建代表 Web 服务的 LOTOS 进程 WebServiceP。所以非终结符 invoke 应具有综合属性 service(表示 LOTOS 文档中的 WebServiceP 进程对象)。但是因为该属性并不需要在 BPEL 产生式中进行传递(所表示对象的内容在该〈invoke〉节点可确定),所以翻译模式中可不考虑该属性。而其他 BPEL 活动的翻译代码应包含在 specification、client 和 mainProcess 三个对象之中。所有的非终结符应具有综合属性 code,用于记录将 BPEL 活动翻译为 LOTOS 动作的代码。

对于 createInstance = "yes" 的 receive 的 client 和 main 属性,需要通过 activity 传递给它的兄弟节点,所以其他产生式符号还具有继承属性 c 和 m,分别用于传递 client 和 main 属性。但是除了 createInstance = "yes" 的 receive 的兄弟节点 activities1 之外,其他的非终结符包括 receive(createInstance = "no"),都是从父节点继承 c 和 m 属性的。

简单 BPEL 实例的语法分析树的依赖图如图 3 所示。

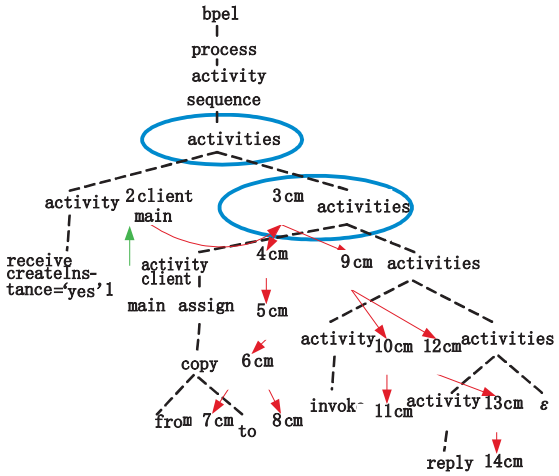


图3　简单BPEL实例依赖图

从图 3 中可知,activities→activity activities1 产生式根据 activity 推导出的 receive 情况的不同(createInstance 值为"yes"或"no"),具有不同的语义动作。而其他非终结符也只有在 BPEL 中检测到 createInstance = "yes" 的 receive 之后才具有继承属性 c 和 m。根据综合属性从下往上,继承属性从上往下和在兄弟节点之间传递的原则,设计一个继承属性 ci 和相应的综合属性 createI。ci 在检测到〈process〉时设置为"false"。当检测到 createInstance = "yes" 的〈receive〉时,设置 createI 为"true",而在 activities→ativity activities1 产生式中设置 ativities.ci = activity.createI。

部分翻译模式如表 3 所示。

上述翻译模式中,对 BPEL 的每一个活动都有一个翻译函数,用于实现将该活动映射为 LOTOS 动作,如 compileIf(…)用于翻译〈if〉活动为 LOTOS 代码,其中的参数由表示 specification、client 和 mainProcess 对象的属性决定。checkCreateInstance()函数用于检测〈receive〉的 createInstance 属性是否为"yes",是则返回"true",否则返回"false"。newClient()函数用于创建一个 LOTOS 客户端进程对象,newMain()函数用于创建 BPEL 主流程对应的 LOTOS 主进程对象。

对 BPEL 基本活动,如:receive partnerLink portType operation 有{ receive.code = compileReceive(…) }。而以〈if〉为例说明 BPEL 结构化活动语义动作的设计。如表 4 所示。

表 3 BPEL 到 LOTOS 映射的翻译模式

| 产生式 | 语义动作 |
|---|---|
| bpel→ process | { process. s = newSpecification( ); } { bpel. code = process. code} |
| process→ activity | { activity. s = process. s; activity. ci = 'false' } { process. code = compileProcess( …) ||activity. code } |
| activities→ | { activity. s = activities. s; activity. ci = activities. ci; if activity. ci ! = "false" then activity. c = activities. c;activity. m = activities. m } |
| activity | { activities1. s = activities. s; activities1. ci = activities. ci; if activity. createI ! = "false" then activities1. c = activity. client;activities1. m = activity. main;activities1. ci = activity. createI; else activities1. c = activities. c; activities1. m = activities. m;activities1. ci = activities. ci } |
| activities1 | {activities. code = activity. code \|\| activities1. code } |
| activities→ε | { activities. code = null } |
| activity→ receive | { receive. s = activity. s; receive. ci = activity. ci; if receive. ci! = "false" then receive. c = activity. c; receive. m = activity. m } { if checkCreateInstance( )! = "false" then activity. creatI = "true"; activity. client = newClient( ); activity. main = newMain( ); activity. code = receive. code } |
| activity→ reply | { reply. s = activity. s; reply. ci = activity. ci; if reply. ci! = "false" then reply. c = activity. c; reply. m = activity. m} { activity. code = reply. code } |
| activity→ invoke | { invoke. s = activity. s; invoke. ci = activity. ci; if invoke. ci! = "false" then invoke. c = activity. c; invoke. m = activity. m} { activity. code = invoke. code } |
| activity→ sequence | { sequence. s = activity. s; sequence. ci = activity. ci; if sequence. ci! = "false" then sequence. c = activity. c; sequence. m = activity. m} {activity. code = sequence. code} |
| activity→ if | { if. s = activity. s; if. ci = activity. ci; if if. ci! = "false" then if. c = activity. c; if. m = activity. m} { activity. code = if. code } |

表 4 〈if〉活动的翻译模式

| 产生式 | 语义动作 |
|---|---|
| if→ codition | { condition. m = if. m } { activity. s = if. s; activity. ci = if. ci; if activity. ci! = "false" then activity. c = if. c; activity. m = if. m } |
| activity | { elseifs. s = if. s; elseifs. ci = if. ci; if elseifs. ci! = "false" then elseifs. c = if. c; elseifs. m = if. m } |
| elseifs | { else. s = if. s; else. ci = if. ci; if else. ci! = "false" then else. c = if. c; else. m = if. m } |
| else | { if. code = compileIf( …) \|\| condition. code \|\| activity. code \|\| elseifs. code \|\| else. code } |
| elseifs→ | { elseif. s = elseifs. s; elseif. ci = elseifs. ci; if elseif. ci! = "false" then elseif. c = elseifs. c; elseif. m = elseifs. m } |
| elseif | { elseifs1. s = elseifs. s; elseifs1. ci = elseifs. ci; if elseifs1. ci'! = "false" then elseifs1. c = elseifs. c; elseifs1. m = elseifs. m } |
| elseifs1 | {elseifs. code = elseif. code\|\|elseifs1. code} |
| elseifs→ε | { elseifs. code = null } |
| elseif→ condition | { condition. m = elseif. m } { activity. s = elseif. s; activity. ci = elseif. ci; if activity. ci! = "false" then activity. c = elseif. c; activity. m = elseif. m } |
| activity | { elseif. code = compileElseIf( …) \|\| condition. code \|\| activity. code } |
| else→ | { activity. s = else. s; activity. ci = else. ci; if activity. ci! = "false" then activity. c = else. c; activity. m = else. m } |
| activity | { else. code = compileElse( …) \|\| activity. code } |
| else→ε | { else. code = null } |

### 2.3 基于翻译模式的转换算法

本文使用 dom4j 解析 BPEL 文档,使用 wsdl4j 解析 WSDL 文档。不对 BPEL 作具体的词法分析和语法分析,而在构建语法分析树的过程中按照上述翻译模式进行翻译。以〈receive〉和〈if〉为例,翻译过程以伪代码形式显示如下:

获得 BPEL 文档对象;

根据 BPEL 根节点〈process〉创建 specification 对象;

根据〈import〉分析 BPEL 所调用的服务;

在主界面中输入所调用服务的 WSDL 地址;

根据〈partnerLinks〉创建 BPEL 所调用服务对应的 LOTOS 进程对象;

根据〈variables〉和 WSDL 地址,生成 BPEL 变量对应的 LOTOS 自定义数据类型文件;

```
//开始解析 BPEL 主流程
compileActivity(process 对象下的子节点){
if(子节点为〈receive〉){
  if(checkCreateInstance( ) = = "true"){
    调用 newClient( )实例化 LOTOS 客户端进程对象;调用 newMain( )实例化 LOTOS 主进程对象;对变量 createI 赋值为"true";
  }
  compileReceive( …);
}…
if(子节点为〈if〉){
  翻译〈condition〉;
  compileActivity( …);
   compileElseIfs( …);
   compileElse( …);
}
}
```

## 3 案例

房屋贷款案例:银行根据"购房者"目前拥有的房屋数量提供不同的首付比例和贷款利率。首先根据客户的姓名调用 HouseLoanAgency 服务,查询客户已有的房屋数量,然后根据返回的房屋数量调用 Bank0 ~ Bank3 服务,获得相应的贷款首付和贷款利率。Bank0 ~ Bank3 服务分别负责处理已有房屋数量为 0、1、2 和大于等于 3 的客户房屋贷款需求。

输入房屋贷款业务的 BPEL 文件地址和相应服务 WSDL 地址,完成对 BPEL 到 LOTOS 的转换。工具界面如图 4 所示。转换后得到五个自定义数据类型文件和一个 LOTOS 文件。简化的 LOTOS 文件如下:

```
specification houseloanbrokerProcess[In, Out, G1, G2, G3, G4, G5] : noexit
  Library( *引入所需的数据类型 * )
    DT1, …, NATURAL
  endlib
behavior
  Client[In, Out]( *客户端 * )
  |[In, Out]|
  houseloanbrokerProcess[In, Out, G1, G2, G3, G4, G5]
  ( *BPEL 流程主进程 * )
  |[G1, G2, G3, G4, G5]|
  (
    HouseLoanAgency[G1]( *代理服务进程 * )
  |||
    Bank0[G2]( *BPEL 所调用服务的进程 * )
  |||
  …
  )
where
```

```
process houseloanbrokerProcess [ In, Out, G1, G2, G3, G4, G5 ] :
noexit :=
  In ? a :DT1;( * 主进程接收客户端发来的请求 * )
    G1 ! consDT2(extractDT11(a)); G1 ? b :DT3;
    ( * 调用 houseLoanAgency 服务 * )
  [extractDT31(b) =0] ->
    Ca_UnknownNAME[Out](a)( * 故障处理进程 * )
  []
  (
  ([extractDT31(b) =1] ->
    G2 ! consDT1(extractDT3(b)); G2 ? c :DT4;
    ( * 调用 Bank0 服务 * )
    exit(extractDT41(c),extractDT42(c)))
  []
  …
  ([NOT(extractDT31(b) = =1) and NOT(extractDT31(b) = =
2) and NOT(extractDT31(b) = =3)] ->
    G5 ! consDT1(extractDT3(b)); G5 ? f :DT4;
    ( * 调用 Bank3 服务处理房屋数量 > =3 的情况 * )
    exit(extractDT41(f),extractDT42(f)))
)
    > > accept g,h :Nat in Out ! consDT4(g,h);stop
( * 将结果发送回客户端 * )
endproc
…
endspec
```

上述文件只列出了主进程的部分内容,并对门名和自定义数据类型名作了简化,以便显示。其中,In 和 Out 是客户端与主进程间交互的门,G1 ~ G5 分别为 BPEL 与调用的服务之间交互的门,DT1 ~ DT5 为 LOTOS 自定义数据类型。

使用 CADP 对转换后得到的 LOTOS 文件进行检测,得到的状态空间如图 5 所示。

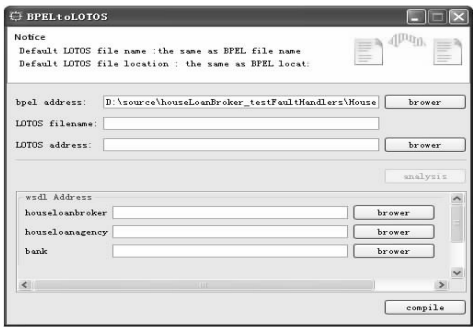


**图4　BPEL到LOTOS建模工具界面**

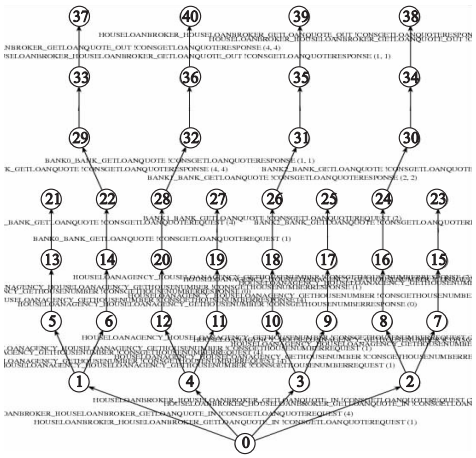

**图5　房屋贷款案例状态空间**

## 4　结束语

Web 服务组合的 LOTOS 建模是 Web 服务组合验证的一个重要方法。但大多数研究都停留在如何对 BPEL 进行 LOTOS 建模,而对如何实现 BPEL 到 LOTOS 的自动化转换研究较少。本文鉴于编译原理的思想,提出一种基于翻译模式的 BPEL 到 LOTOS 的转换算法。从 BPEL 的 XML schema 定义出发,可对 BPEL 文档中的各种情况考虑周全,保障转换过程一定的通用性。根据翻译模式进行翻译,可使翻译过程有条不紊。与其他相关工作比起来,本文丰富了 BPEL 到 LOTOS 的映射规则(增加了数据类型映射,并重新设计了故障处理机制),对 BPEL 到 LOTOS 的通用转换进行了探索。

下一步工作中,需要建立更加完善的 BPEL 到 LOTOS 的自动化转换工具,争取最大程度地实现转换过程的通用性,主要是对 BPEL 中其他非常见的和扩展的活动进行翻译,以及对 BPEL 到 LOTOS 转换的正确性和完备性进行验证。

**参考文献:**


[1] ALVES A, ARKIN A, ASKARY S, *et al*. Web services business process execution language version 2.0[EB/OL]. (2007-04-11)[2012-08-19]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[2] 倪悦, 范玉顺. 基于着色 Petri 网的语义 Web 服务组合形式化验证[J]. 清华大学学报, 2010,50(5):714-717.

[3] 毛瑞雪. 基于 Pi 演算的 Web 服务组合过程框架[J]. 计算机与数字工程,2011,39(7): 12-16.

[4] 陈宁, 于守建, 乐嘉锦. 基于有限状态自动机的组合 Web 服务性能优化[J]. 计算机应用与软件,2011,28(10): 97-99.

[5] 王晶,戎玫,张广泉,等. 基于概率模型检测的 Web 服务组合验证[J]. 计算机科学,2012,39(1):120-123.

[6] LOGRIPPO L, FACI M, HAJ-HUSSEIN M. An introduction to LOTOS: learning by examples[D]. Ottawa:University of Ottawa,1999.

[7] TURNER K J. The formal specification language LOTOS: a course for users[D]. Scotland: University of Stirling, 1996.

[8] BOLOGNESI T, BRINKSMA E. Introduction to the ISO specification language LOTOS[D]. Enschede: University of Twente,1988.

[9] 李益昌. 形式描述技术 LOTOS 及其在实时系统的扩展和应用[D]. 广州: 广东工业大学,2004.

[10] SALAIIN G, FERRARA A, CHIRICHIELLO A. Negotiation among Web services using LOTOS/CADP[C]//Lecture Notes in Computer Science. Berlin: Spring-Verlag,2004:198-212.

[11] FERRARA A. Web services: a process algebra approach[C]//Proc of the 2nd International Conference on Service Oriented Computing. New York: ACM Press,2004:242-251.

[12] ZHAO Hui-qun,WANG Wen-wen,SUN Jing, *et al*. Research on formal modeling and verification of BPEL-based Web service composition [C]//Proc of Computer and Information Science. [S. l.]: IEEE/ACIS,2012: 631-636.

[13] LONG Hui-yun,LI Jian-shi. A process algebra approach of BPEL4WS [J]. Journal of Information and Computing Science, 2009, 4(2):93-98.

[14] CHINNICI R, MOREAU J J, RYMAN A, *et al*. Web services description language (WSDL) version 2.0[EB/OL]. (2007-06-26)[2012-08-19]. http://www.w3.org/TR/wsdl20/.

[15] AHO A V, LAM M S, SETHI R, *et al*. Compilers principles, techniques and tools[M]. 赵建华,郑滔,戴新宇, 译. 2nd ed. Beijing: National Defence Industry Press,2009:194.

[16] 陈火旺, 刘春林. 程序设计语言编译原理[M]. 3 版. 北京: 国防工业出版社, 2010:213.