Web服务组合事务处理及基于细胞膜演算的正确性分析

姚绍文[1] 唐明靖[1] 危 兵[2]

(云南大学网络智能计算研究室 昆明 650091)[1] (云南省信息技术发展中心 昆明 650091)[2]

摘 要 事务处理是Web服务及其组合服务能否得到广泛应用的关键技术之一。本文分析总结了现有的事务处理模型,针对Web服务组合环境下的异构性、自治性和动态性,提出了一个基于组合Web服务的事务处理模型WS-CTM,并基于细胞膜演算对模型的正确性进行了验证。

关键词 Web服务,事务处理,细胞膜演算,Maude系统

Transaction Processing of Compositive Web Service and its Correctness Analysis of Calculus Based on Cell Membrances

YAO Shao-Wen[1] TANG Ming-Jing[1] WEI Bing[2]

(Intelligent Network Computing Research of Yunnan University, Kunming 650091)[1] (IT Development Center of Yunnan, Kunming 650091)[2]

Abstract Transaction processing is one of key technologies that make Web service and its compositive service can or not be wide used. This paper summarizes and analyzes the exsisting transaction processing models. Then according to compositive Web service characteristics—heterogeneous, autonomous and dynamic, proposes a transaction processing model called WS-CTM which is based on compositive Web service, and verifies its correctness based on membrane calculus at last.

Keywords Web service, Transaction processing, Membrane calculus, Maude system

1 引言

随着Internet和SOA[1](面向服务的软件架构)思想的不断发展,Web服务技术作为一种具体的SOA实现,已经在学术界、工业界得到了极大的关注,相关协议、标准也在陆续制订和完成中。Web服务技术的目标是通过使用Web标准获得应用间的全方位的互操作,它通过开放互联网协议(如SOAP[2],WSDL[3],UDDI[4]),建立一个松耦合的集成模型,实现跨组织的异构系统集成。Web服务组合是指通过集成基本的Web服务,从而产生新的增值Web服务的能力。只有通过服务组合技术,才能使Web服务在跨组织的、异构的系统集成环境中发挥巨大的潜能[5]。这就需要解决Web服务环境中的事务处理问题,以保证Web服务及其组合的可靠性与完整性。

事务处理作为Web服务的关键技术之一,决定着它是否能被广泛采用。事务在Web服务的运行环境中可能跨越多个自治组织,运行时长跨越几个小时,与传统事务处理系统存在很大的区别。传统的分布式事务处理运行在紧耦合的环境中,而Web服务事务处理运行在松耦合的环境中,事务的参加者可能属于不同的、自治的组织和部门,事务的运行需要跨越多个不同组织和防火墙,而且事务可能是持续时间比较长的商业事务。这样,传统的事务处理方式明显不能满足Web服务环境下的事务需求。Web服务事务处理吸取了一些松耦合环境下事务模型的方法,放松了ACID属性中的某些性质。

在Web服务环境下,需要一种灵活的方式来协调多个Web服务参与的复杂业务,提供一种更为松散的事务模型来保证组合任务的正确完成。本文分析总结了现有的事务处理模型,针对Web服务组合环境下的异构性、自治性和动态性,提出了一个基于组合Web服务的事务处理模型WS-CTM。

2 Web服务组合事务模型WS-TCM

2.1 Web服务组合事务类型

Web服务组合事务是跨平台的、跨边界的,事务的参与者通常是分布在网络中不同的自治节点。对于由一组服务$S=\{S_1, S_2, \cdots, S_n\}$相互协作完成的一次复杂业务,其对应的事务是$T=\{T_1, T_2, \cdots, T_n\}$。其中,$S_1, S_2, \cdots, S_n$是根据业务逻辑进行动态组合来提供服务$S$。$S_1, S_2, \cdots, S_n$可能是有独立业务逻辑服务实体,其对应的事务$T_i$可以具有严格的ACID属性,也可以放松ACID属性;并且S_i之间通过不同的组合结构(顺序、选择、替代等)进行服务组合。

基于以上讨论,WS-TCM模型中定义了两种事务类型以满足Web服务组合环境下的事务处理需求。

定义4.1 原子事务(Atomic Transaction, AT)$AT=\{T_1, T_2, \cdots, T_n\}$是指在一个事务作用域内事务参加者$T_i(0 \leqslant i \leqslant n)$要么都提交、要么都撤销的事务,用于协调短生命周期的操作。T_0是事务的发起者,代表全局事务。

原子事务具有ACID属性,所有的事务参加者同步提交。如果参与者都投票标识它们能够成功执行,协调者就要求所有的参加者提交。否则,协调者要求所有的参加者异常终止。

定义4.2 组合事务(Composite Transaction,CT)$CT=\{T_1, T_2, \cdots, T_n\}$是指在一个事务作用域内允许一部分事务候选者$T_i(0 \leqslant i \leqslant n)$提交,而部分候选者$T_j(0 \leqslant j \leqslant n)$不执行或中止的事务。$T_0$是根事务,是事务的发起者,代表全局事务。

组合事务的子事务之间没有锁定机制,其提交前的故障用回滚操作消除,提交后的恢复采用补偿事务来实现。组合

事务放松了隔离性和原子性，适合 Web 服务长事务和松耦合的性质。

2.2 原子事务协调机制与算法

在 Web 服务环境中，事务处理主要解决怎样协调子事务而不是处理每个子事务的内部。本节中讨论的 Web 服务组合事务协调机制参考了文[6]中提出的算法，并考虑到在 Web 服务组合环境中存在大量功能相等的服务，加入了基于功能替代的组合事务策略，得到一个适合服务组合环境下的事务处理机制。

原子事务的协调过程具体说明如下。

事务创建。服务消费方代理接受应用程序的请求，向远程 Web 服务的代理发送协调上下文消息 CC。该消息包含了创建一个事务所需要的必要信息，包括事务类型 AT、事务标识符、CC 发送者的地址、接收者的地址和超时参数等。

事务准备。收到协调者的 Prepare 消息后，每一个参与者 $P_i(i=1,2,\cdots,n)$ 申请执行子事务所需要的资源。如果成功，它向协调器返回一条 Prepared 消息，表示该事务参加者为提交做好了准备；否则返回一条 Notprepared 消息。

事务提交。协调者做出提交/放弃决定。在事务准备时间间隔 T_p 内，如果协调者收到 N 条 Prepared 消息，则说明此时所有参加者都可以提交，协调者于是决定全局提交，发送 Commit 消息给所有的参加者。

原子事务协调算法包括协调者算法和参与者算法两个部分，协调者和参与者分别执行各自的算法，以控制全局事务的提交或放弃。如图 1 所示，t 表示协调者和参与者的等待时间；$n1$ 和 $n2$ 表示协调者收到的消息，数量 N 表示参加者的数量。T_p 表示协调者等待的参与者事务准备超时间隔，T_w 表示协调者等待提交的超时间隔。

```
原子事务协调者算法
输入：所有参加者服务引用，时间参数
输出：全局事务的结果
{step 1: initiate a AT
 agent creates Coordinator;
 agent sends CC to all agents of Pi;
 wait for Response from agent of Pi;
 if timeout exit;
setp 2: prepare for the transaction commit
 send Prepare to all Participants;
 while(t≤Tp)and (n1<N)
   wait for and record income messages;
step 3: commit the transaction
  if(n1=N)and(all n1 messages are Prepared)
   {  record commit in log;
      send Commit to all Participants;
      while(t≤Tw)and(n2<N)
        wait for and record income messages;
   if(n2<n)or(Not N messages are Committed)
      { send Rollback to all Participants;
        exit after receiving all Rollbacked;}
        exit after receiving Committed; }
   else{
  send Abort to all Participants;
  exit after receiving all Aborted;}}
```

```
原子事务参加者算法
输入：CC 消息，时间参数
输出：事务 Ti 的结果
{step 1:join in the transaction
 agent creates Participant after receiving CC;
 agent sends Response to Coordinator;
setp 2: allocate resources
 wait for Prepare from Coordinator;
 if timeout exit;
 success=allocate resources;
 if(success)
   send Prepared to Coordinator;
 else{ send Notprepared to Coordinator;
      exit; }
step 3: commit sub-transaction
 while(t≤Tw)and(message isn't Commit or Abort )
    wait for income messages;
   if (messages is Commit)
     {record commit in log;
       Commit sub-transaction;
 /* if(nested transaction) call 原子事务协调者算法
       send Committed to Coordinator;}
   else{
     cancel allocate;
     exit;}}
```

图 1 原子事务协调算法

原子事务的协调过程是事务协调者和参与者之间执行原子事务协调算法的过程，协调者与参与者之间通过交互一系列消息，来促使一系列的事务状态转变。事务状态的变迁过程可以采用状态转换图来描述。图 2 为原子事务协调者和参与者状态转换图。

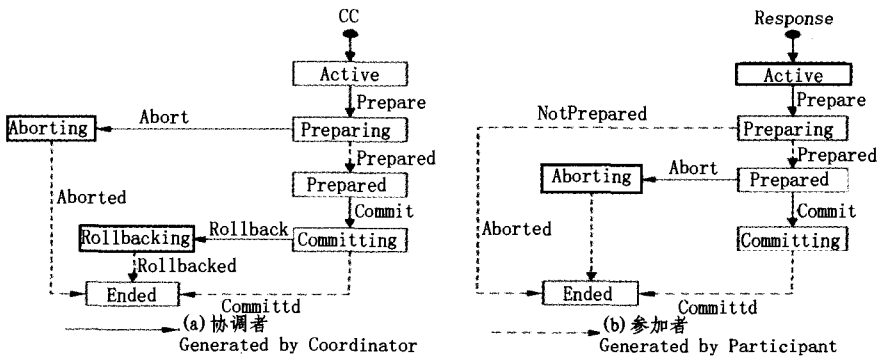

图 2 原子事务模型状态转换图

2.3 组合事务协调机制与算法

组合事务用于协调 Web 服务环境下持续时间较长的业务事务。与原子事务模型相比，组合事务具有如下特点：

①部分子事务的失败不会引起全局事务的中止。组合事务放松了原子事务的原子属性，当部分子事务执行失败时，协调者可以重新选择功能相等的服务，继续完成这部分子事务。

②用户可以选择提交结果。由于在网络上存在多个相同功能类型的服务，组合事务应该可以帮助用户撤销部分服务的提交结果，以便更加符合商业活动的业务逻辑。

③独立提交子事务。组合事务的各个子事务独立提交，

并在提交后立即释放资源。

```
组合事务协调者算法
输入：所有候选者服务引用，时间参数
输出：全局事务的结果
{step 1: initiate a AT //创建一个原子事务
 agent creates Coordinator;
 completed = false;
while(t≤T and not completed)
{agent sends CC to agents of candidates;
 Wait for Response from agent;
setp 2: enroll candidates
 send Enroll to all Candidate;
step 3: confirm/cancel candidates
    wait for and record messages;
  if (messages is Committed)
    {
if(user selects some candidates)
        {send Confirm to candidates;
         Wait for Confirmed messages; }
        else
          {send Cancel to Candidates;
           Wait for Cancelled messages; }
if(CT completes successfully)
Completed=true
} } }
```

```
组合事务候选者算法
输入：CC 消息，时间参数
输出：全局事务的结果
{step 1: join in the transaction
 agent creates Coordinator;
 agent sends Response to Coordinator;
setp 2: commit sub-transaction
 wait for Enroll from Coordinator;
 if timeout exit;
allocate resources;
record commit information in log;
commit and generate compensating transaction;
release resources;
/*if(nested CT transactions) call 组合事务协调者算法
 if(nested AT transactions) call 原子事务协调者算法
step 3: confirm/compensate
if(commit successfully){
send Committed to Coordinator;
while(t≤T)
{wait for incoming messages;
if (message is Cancel)
{execute compensation transaction;
send Canceled to Coordinator ;}
else if (message is Confirm)
{send Confirmed to Coordinator ;}}}}
```

图 3　组合事务协调算法

组合事务的协调过程具体说明如下：

事务创建　代理接受应用的请求创建一个事务，类似于原子事务，不同的是事务类型为 CT。并发送事务协调上下文 CC 消息给事务候选者，通知它们加入到事务中。

参加者注册　远程服务根据 CC 中的消息选择是否加入该事务，以 Response 消息向协调者注册。

候选者独立提交各自子事务　各个候选者分配资源，记录操作在日志中，然后各自提交。如果提交成功，每个候选者产生相应的补偿事务，然后返回包含执行结果的 Committed 消息。如果提交不成功，则撤销已经执行的操作，并返回 Abort 消息。

用户确认　用户通过协调者在时间 T 内分别通过 Confirm 或 Cancel 消息来确认或取消成功提交的候选者，但不必应答提交失败的候选者，并继续发送 CC 消息给新的功能相等的候选者，直到成功或尝试 M 次。

成功提交的候选者确认用户的选择　在 T 时间内，如果事务候选者收到 Confirm 消息，它回应 Confirmed 消息，使子事务的结果不可再更改。

在组合事务中，一个重要的参数就是协调者与候选者之间协商的有效时间 T，如图 3 所示。

协调者与候选者执行组合事务协调算法，经过一系列状态变换直到事务结束。图 4 是组合事务协调者与候选者的状态变换图。

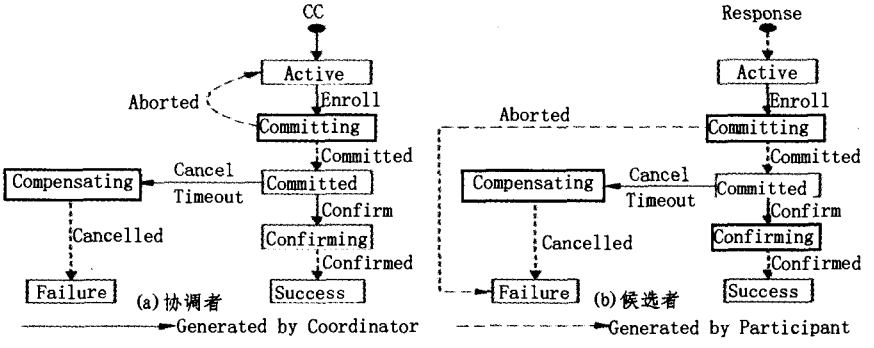

图 4　合事务模型状态转换图

3　基于细胞膜演算验证协调算法正确性

3.1　细胞膜简介

细胞膜计算的基本模型：P-systems[7] 由 Păun 在 1998 年提出，是受生物学上的细胞膜特性启发而发展出来的一种计算模型，并在理论上得到了充分的发展。P-systems 名称的得来源自 Păun systems 的简称。在生物学中，每个细胞膜包含一些分子，这些分子对象能够按照一定的反应规则演化。分子可以在细胞膜之间穿越，细胞膜本身能够溶解或者分裂。P-systems 就是按照这些特性构建出来的计算模型，并且出现了很多种变形。这些 P-systems 变形大部分具有和图灵机等价的表达能力。

3.2　细胞膜演算重写逻辑语义

细胞膜演算作为化学抽象机的一种自然扩展，适合动态变化的移动、分布式计算环境；它基于反应规则，很适合用重写逻辑描述。重写逻辑[8] 是一种统一的并发模型，提供了一个形式化框架，可以描述一些常见的形式化系统。本文采用

重写逻辑对细胞膜演算描述的 Web 服务事务处理模型进行描述与建模的原因有：第一，重写逻辑是一个通用的并发描述工具，是基于代数语义和重写语义来描述系统。第二，重写逻辑具有一个很好的 Model Checking 框架，可以采用线性时序逻辑 Linear Temporal Logic(LTL)进行形式验证与分析。目前，在安全协议、移动计算方面的研究很好地说明了重写逻辑的强大描述能力。

3.3 重写逻辑工具——Maude 系统

Maude 系统是一个由 UIUC 大学开发的同时支持等式和重写规则的强大工具，表达能力和通用性强，而且具有良好的运行性能。当前最新版本是 Maude 2.3[9]，它支持两种模型：一种是函数式模型，用来表示成员等式理论[10]；另外一种是系统模型，用来表示通用的重写理论。Maude 2.3 不同于其他一般的形式化系统，它支持高效的 Model Checking，采用线性时序逻辑 LTL(Linear Temporal Logic)模块。

我们采用重写逻辑工具 Maude 定义了一个基本细胞膜验算模型 Membrane。定义如下：

```
mod Membrane is
  protecting QID .
  sorts Name Obj Rule Mem . subsorts Obj Rule < Mem . subsorts Qid < Name .
  subsorts Qid < Obj . op _ _ : Obj Obj -> Obj [ assoc comm id: nul] . op none : -> Rule .
  op <_,_,in,_> : Obj Obj Name -> Rule .
...
```

图 5 Maude 建模细胞膜演算

如图 5 所示，关键字 mod，protecting，sorts，op，vars，rl，crl 是 Maude 系统定义的关键字。mod 定义了一个 Maude 模型，protecting 表示引入其他模型，sorts 和 subsorts 分别用来定义序与子序，op 和 vars 定义模型中的操作符与变量。QID 表示标识符类型为字符串类型。接下来我们定义了四个序：Name，Obj，Rule 和 Mem，分别代表名字、对象、规则和细胞膜。

3.4 组合事务模型协调算法正确性验证

在组合事务模型中，原子事务模型是组合事务模型的特殊形式，其协调算法比较简单。限于篇幅，我们重点对组合事务模型协调算法进行正确性验证。假设一个组合事务由两个子事务组成 $T=\{t_1,t_2\}$，它们分别关联候选者 P1 和 P2。在细胞膜结构中，细胞膜 0 表示全局事务，细胞膜 1 和细胞膜 2 分别代表第一个事务和第二个事务。我们用 Success 表示子事务的成功，Fail 表示子事务的失败。我们采用 Maude2.3 对其建模，如图 6 所示。

```
mod CT is
inc Membrane .op ct : -> Mem .
eq ct =
[0 :
'Begin,< 'Begin, 'Begin1, in, 1 > | < 'Success1, 'Begin2, in, 2 > |< 'Compensate, 'Compensate1, in, 1 >,
[1 :nul, < 'Begin1, 'T1 > | < 'T1, 'Success1, out> | < 'T1, 'Fail, out> |< 'Compensate1, 'Fail, out>, O ]
[2 :nul,< 'Begin2, 'T2 > |< 'T2, 'Success, out> | < 'T2, 'Fail2 > |< 'Fail2, 'Compensate, out>, O ]] .
endm
```

图 6 组合事务运行时模型

如图 6 所示，首先通过 inc Membrane 引入刚才定义的通用模型，定义一个 ct 常数算子，并采用等式将 ct 定义为一个具体的 Web 服务事务结构。开始时，细胞膜 0 中存在项 Begin，而且有重写规则〈'Begin，'Begin1，in，1〉，这两个组合起来，那么项 Begin 会被重写为 Begin1 并被送入到子细胞膜 1 中。也就是通过这条重写规则，子事务被触发运行。在细胞膜 1 中存在 Begin1，并且存在规则〈'Begin1，'T1〉，表示子事务 1 的执行。如果这个子事务执行成功，那么规则〈'Success1，'Begin2，in，2〉将送出一个项 Success1 到细胞膜 0 中，表示第一子事务运行成功。否则，规则〈'T1，'Fail，out〉表示项 Fail 将被送到细胞膜 0 中，整个事务失败。如果第一个事务成功，那么规则〈'Success1，'Begin2，in，2〉表示开始第二个子事务，如果这个事务成功，那么项 Success1 会被送到细胞膜 0 表示整个事务成功。否则，规则〈'Compensate，'Compensate1，in，1〉表示项 Compensate 将被送到细胞膜 0 中，产生一个补偿动作。

我们可以运行 rew 命令来运行这个模型，如图 7 所示。

系统命令 rew 是个项重写命令，表示将这个模型按照一定的重写策略对项进行重写。从图 7 可以看出，在第 4 次重写时，项'Fail 出现在细胞膜 0 中，表示事务失败。

接下来我们采用 LTL 模型检验技术，对模型的正确性进行验证。首先，定义模型检验需要的 Maude 模型。如图 8，首先定义一个状态序，subsort Mem＜State。然后定义两个原子命题 Committed 和 Aborted，如 eq ['0：'Success O1，R1，M1] |= Committed('0) = true。表示细胞膜 0 中出现对象 'Success 时表示整个事务成功。根据原子命题的定义，我们接下来可以定义一个正确性公式，eq Correctness = 〈〉(Committed('0) ∨ Aborted('0))。这个正确性公式表示要么两个事务成功提交，要么整个事务失败，没有其他结果。运行结果如图 9 示，表明模型的正确性是满足的。

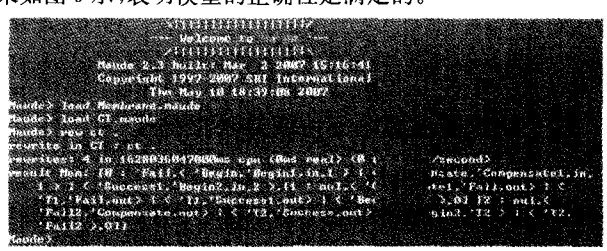

图 7 rew 命令结果图

(下转第 122 页)

题及解决方案和工程现场视频等的编辑功能，方便领导和项目经理及时掌握项目进行过程中所遇到的问题，并可查看工程施工的现场情况。

新建项目时，需要选择适合此工程项目的工作流模板，工作流模板列表由工作流流程管理模块提供，并调用工作流引擎中的相应方法将其实例化为本项目的工作流程。用户可设定项目的计划开始时间，工作流引擎将根据每个任务的工期自动计算出所有任务的计划起止时间。用户也可编辑项目流程中每个任务的负责人或角色、资金费用、工期、起止时间等信息。图4所示为工作流实例化后的项目流程信息。工作流引擎将解释任务分配、角色分配及状态定义，引导业务活动的顺利执行，这样便完成了在多个参与者之间按照某种预定义的规则传递文件、信息或任务来完成业务目标的过程。

任务节点	任务节点后继	计划开始时间	计划结束时间	计划费用	负责人	详细设置
1. 滨海新区规划	2. 近期建设规划	2006-12-30	2006-1-9	—	—	编辑
2 近期建设规划	201 重大项目计划前论证 3. 建设计划编制	2007-1-9	2006-1-19	—	—	编辑
3 建设计划编制	4. 建设计划批准	2007-1-20	2007-1-25	—	—	编辑
4 建设计划批准	401 建设计划报备 5. 项目组织实施方案	2007-1-25	2007-2-10	—	—	编辑
5 项目组织实施方案	501 市政工程规划设计条件、方案、成 6. 项目方案(预可研)设计	2007-2-10	2007-3-2	—	—	编辑

图4 项目流程信息(部分)

3.4 系统特色

本系统的特色是对项目流程采用图形化的管理方式。项目流程实例化后，将生成对应于该项目的项目流程图，如图5所示。点击任务名的链接，可执行对此任务的相关操作，不同的角色有不同的操作权限。流程图中还标明了各个任务的执行情况(状态)，有完成、正在进行、未开始、超期未开始、超期未完成五个状态，并分别用不同的颜色表示。在流程的执行过程中，工作流引擎将调用相关方法对工程执行的进度进行管理，并根据任务的状态提供相应的预警功能以及报表、统计分析等其他功能。

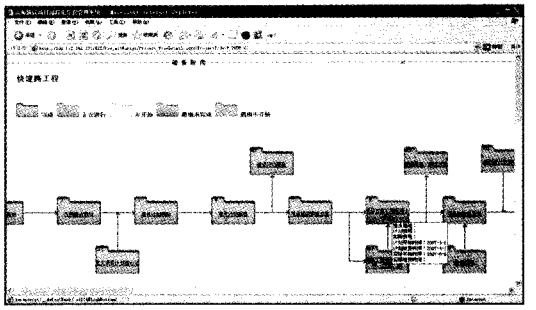

图5 项目流程(部分)

结论 本文设计并实现的滨海新区项目流程化信息管理系统，为滨海新区重大建设项目搭建了一个安全的、有效的和可行的管理监控平台。采用工作流技术真正实现了项目流程的自动化管理，并实现了图形化的管理模式，使得整个项目流程更清晰、管理更有效。该管理系统具有很强的适应性和扩展性，可以改变传统的政府办公模式，为数字化城市的建设提供新途径。

参考文献

1 Trappey A J C, Chiang Tzu-An, Ke Sam. Developing an intelligent workflow system to manage project processes with dynamic resource control. Journal of the Chinese Institute of Industrial Engineers, 2006, 23(6): 484～493
2 何跃，等. 基于WEB的工作流管理系统设计与实现. 计算机工程与应用，2005，41(33)：201～205
3 Richter J, Balena F. NET 框架程序设计 [M]. 武汉：华中科技大学出版社，2004
4 范玉顺. 工作流管理技术基础——实现企业业务过程重组、过程管理与过程自动化的核心技术[M]. 北京：清华大学出版社，施普林格出版社，2001
5 田熙清，等. 基于工作流技术的项目管理系统的分析和设计. 计算机工程与应用，2003，39(8)：131～134

(上接第112页)

```
mod CHECK-CT is
    inc CT . inc MODEL-CHECKER .inc LTL-SIMPLIFIER .
    subsort Mem < State .
    op Committed : Name -> Prop . op Aborted : Name -> Prop .op Correctness : -> Formula .
    vars O1 O2 : Obj .
    eq ['0 : 'Success O1, R1, M1 ] |= Committed('0) = true .eq ['0 : 'Fail O1, R1, M1 ] |= Aborted('0) = true .
    eq Correctness = <> (Committed('0) \/ Aborted('0) ) .
endm
```

图8 LTL模型定义

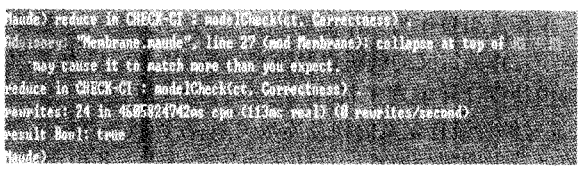

图9 LTL正确性检验结果

结论 Web服务技术作为一种面向服务的分布式计算模型，可以方便地实现跨平台、语言独立、松耦合的异构系统的交互与集成。但是由于Web服务的环境下的异构性、自治性、动态性和不可靠性，能否提供事务处理能力成为Web服务能否广泛应用到电子商务、电子政务等领域的关键技术之一。

本文提出的WS-CTM事务模型能给Web服务及其组合服务提供良好的事务支持。同时基于细胞膜演算对模型进行了形式化建模，并结合线性时序逻辑对模型的正确性进行了验证。

参考文献

1 Newcomer E, Lomow G [美]. Understanding SOA with Web Services. 电子工业出版社，2006
2 SOAP 1.2 Specification. http://www.w3.org/2000/xp/Group/
3 WSDL 1.1 Specification. http://www.w3.org/TR/wsdl
4 UDDI Specification. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec
5 李景霞. Web服务组合综述. 计算机应用研究，2005
6 唐飞龙，李明禄. 一个Web服务事务处理模型：结构、算法和事务补偿. 电子学报，2003，31(12A)：2074～2079
7 Păun G H. Computing with membrances. Journal of Computer and System Sciences, 2000, 61(1): 108～143
8 Bruni R, Meseguer J. Generalized rewrite theories. Lecture Notes in Computer Science, 2003(2719): 252～266
9 Maude2.3. http://maude.cs.uiuc.edu/
10 Bruni R, Meseguer J. Generalized rewrite theories. Lecture Notes in Computer Science, 2003(2719): 252～266