文章编号:1001 - 9081(2008)05 - 1295 - 05

工作流建模语言 XPDL 到 CSP 进程的转化研究

郭李华,吕 钊,顾君忠
(华东师范大学 信息科学技术学院,上海 200062)
(huhucalf107@tom.com)

摘 要:针对工作流定义标准语言 XPDL 缺乏形式化语义,提出了将 XPDL 描述转化为通信顺序进程(CSP)的方法,从而可以利用进程代数 CSP 理论以加强对工作流模型的语义描述分析检测。通过实例分析具体说明转化方法的有效性。

关键词:工作流;XPDL;CSP 进程
中图分类号: TP301.2 **文献标志码**:A

Formal semantics of XPDL specification in CSP process

GUO Li-hua, Lü Zhao, GU Jun-zhong
(*School of Information Science and Technology, East China Normal University, Shanghai* 200062, *China*)

Abstract: Aiming at that the notation specification of XPDL widely supported in practical workflow systems does not include a formal semantics, an approach of transforming XPDL to Communicating Sequential Process (CSP) process was proposed to enhance the formal semantic analysis of workflow process by the rigorous CSP theory. The effectiveness of the approach was verified through a case study.

Key words: workflow; XPDL; CSP process

0 引言

当前,工作流作为实现企业业务过程建模、优化、管理、集成和自动化的核心技术,在不同的领域得到了广泛应用[1]。其中工作流的语义分析验证是工作流系统中的重要环节,是保证执行流程正确的基本。由于进程代数 CSP 不仅具有刻画系统行为的模型,还具有进行推理的形式演算系统,能够很好地描述工作流模型各语义属性,比其他工作流描述方法如工作流网、Petri 网更适用于对模型的描述和检测[2]。

XPDL(XML Process Definition Language)是由工作流管理联盟(Workflow Management Coalition, WFMC)所提出的一个标准化规格,以 XML 作为流程定义交换的机制[3]。然而,虽然 XPDL 是业务过程定义的标准,由于 XML 语言缺乏坚实的数学基础和逻辑基础,使得由其描述的模型没有很好的模型分析方法作为支撑。如果直接对 XPDL 进行分析,那么分析过程将变得非常复杂,并且在分析中将会涉及到很多其他的问题。因此,为了使工作流模型能够更好地进行形式化语义描述分析及验证,如何将面向企业过程建模的 XPDL 流程转换成 CSP 进程,进而进行形式化语义分析具有其重要意义。本文分析 CSP 形式化描述和 XPDL 主要元素的基础上,对 XPDL 各元素与 CSP 进程之间的转化映射关系进行分析讨论并举例说明。

1 工作流模型语义的 CSP 描述

1.1 CSP 基本概念

一个 CSP 进程描述为通过通信与其他进程或者外部环境相互交互,一个进程是一序列动作,每个或者一系列动作是一个事件。进程的字母表,即进程所参与的事件的集合,用 αP 表示[4]。以下是一些 CSP 进程的基本运算符号:

$P;Q$ 表示顺序关系;$P \| Q$ 表示并行关系;$P \Pi Q$ 表示内部选择操作;$P \Box Q$ 表示外部选择操作;$P \| \| Q$ 表示并发交错操作;$P \backslash A$ 表示隐藏操作,A 指在 P 的外部环境中被隐藏的事件集;*Skip* 表示进程成功终止执行;*Stop* 表示进程死锁;$P |[A]| Q$ 表示进程 P 和进程 Q 对于集合 A 中事件的交错操作。

CSP 进程描述了行为的模式,如果进程匹配某个行为模式,那么就可以用该行为替换原有的行为:需要一种精化关系来保证一个进程满足另一个进程的所有属性[5]。假设有两个进程 P 和 Q,若 Q 总是遵循这些约束,仅拒绝存在于 P 的失效集合中的事件,并且只接受其迹集合中的事件。此时 Q 具有 P 的所有属性,因此称 Q 是 P 的细化。这里,迹细化记为 $P \sqsubseteq_T Q \triangleq traces(Q) \subseteq traces(P)$[5],失效细化记为 $P \sqsubseteq_F Q \triangleq failures(Q) \subseteq failures(P)$[5]。

1.2 工作流模型中的事件与基本进程

文献[6]中使用 CSP 对 W. van der Aalst 提出的各个控制流工作流模式进行了描述,用于组合建构工作流流程。这里我们结合控制流、数据和资源信息进行扩充描述,以加强工作流模型的语义完整性。

定义 1 为工作流模型定义 start,complete 以及 fail 事件,分别表示流程的开始、完成以及中止。记该进程中所有的事件集合为 αWF。

定义 2 设集合 A 为工作流活动集合,复合事件集合 $\{n: A \cdot init.n\}$ 为工作流活动触发事件集合,集合 $\{n: A \cdot resource.work.n\}$ 为工作流活动执行事件集合,其中 *resource* 为活动的执行者,这里把它定义成一个信道。

收稿日期:2007 - 11 - 29;**修回日期**:2008 - 01 - 02。
基金项目:国家自然科学基金资助项目(60703004);上海市科学与技术发展基金资助项目(055107039)。
作者简介:郭李华(1985 -),女(回族),福建莆田人,硕士研究生,主要研究方向:计算机协同技术; 吕钊(1970 -),女,四川江油人,副教授,博士,主要研究方向:计算机协同技术、形式语义; 顾君忠(1949 -),男,上海人,教授,博士生导师,主要研究方向:计算机协同技术、分布式系统、信息安全。

定义3 将每条转移定义成信道,作为前置活动的输入信道以及后置活动的输出信道。定义基本类型 Data 表示在 CSP 信道上通信的数据。$in?x$ 表示输入数据 x,$ch?x:(\exp)$ 表示输入数据 x 并且 x 符合表达式 exp,$out!x$ 表示输出数据 x。

定义4 用 CSP 进程 $SP(a,b)$ 定义表示一个基本的工作流活动进程,如式(1):

$$\alpha SP(a,b) = \{init.a, init.b, work.a\}$$
$$SP(a,b) = init.a \rightarrow in?x \rightarrow resource.work.a \rightarrow out!y \rightarrow init.b \rightarrow Skip \quad (1)$$

这个进程定义首先触发工作流活动 a,接着输入活动 a 所需的数据变量 x,然后利用资源执行任务,并输出数据变量 y 以供接下去的活动使用,最后触发一个顺序的活动 b。若为更一般的情况,可得

$$P(a,X) = init.a \rightarrow in?x \rightarrow resource1.work.a \rightarrow out!y \rightarrow |||\, b:X \cdot init.b \rightarrow Skip \quad (2)$$

其中 X 为在活动 a 执行后将要被触发的一组工作流活动集。为简化式子,令:

$$PERFORM(a) = init.a \rightarrow in?x \rightarrow resource1.work.a \rightarrow out!y \quad (3)$$

以下针对工作流模型中的几种主要基本模式进行语义描述。

顺序(Sequence):在工作流流程中,活动 b 在活动 a 之后执行。由式(1):

$$SEQ(a,b) = SP(a,b) = init.a \rightarrow in?x \rightarrow resource.work.a \rightarrow out!y \rightarrow init.b \rightarrow Skip \quad (4)$$

与分支(AND-split):当两个或更多任务需并行执行时需要并行分叉。设 a 为某个活动,X 为在活动 a 完成后并发触发的活动非空集合。由式(2),定义:

$$ASP(a,X) = P(a,X) = init.a \rightarrow in?x \rightarrow resource1.work.a \rightarrow out!y \rightarrow |||\, b:X \cdot init.b \rightarrow Skip \quad (5)$$

或分支(XOR-split):在流程的某一点,依据一个结果或流程控制数据,从多个分支路径中选定一个路径。这里定义式(6):

$$XSP(a,Y) = PERFORM.a \rightarrow \Pi b:Y \cdot init.b \rightarrow Skip \quad (6)$$

与接合(AND-join):当几个并行任务都完成后下一任务才能开始执行时需要同步。

$$AJP(X,a) = |||\, k:X \cdot SP(k,a) \mid [\{init.a\}] \mid SP(a,acts)^{[5]} \quad (7)$$

其中 X 为流入的任务集合。

或接合(XOR-join):欲将选择执行的路径合并成一个路径需用“合并”模式。

$$XJP(X,a) = \Box k:X \cdot SP(k,a) \mid [\{init.a\}] \mid SP(a,acts) \quad (8)$$

2 XPDL 中各主要元素与 CSP 进程的映射

2.1 转移元素(Transition)的映射

XPDL 中转移元素信息描述工作流执行期间任务间可能的转移,以及允许或禁止转移的条件。将每个转移定义为一个信道。如 < Transition Id = " ID" From = " ID1" To = "ID2"/ > 对应信道 Tra_ID,作为 ACT_ID1 进程的输出同时作为 ACT _ ID2 的输入。若存在条件子元素 Condition,如 < Condition > orderType == " PO" </Condition > ,则将其在描述活动进程时进行结合。

2.2 活动元素(Activity)的映射

工作流活动定义用于定义组成工作流过程的每一基本任务,是组成工作流流程的基本元素,因此也是转化为 CSP 进程过程中的重要部分。

Performer:Activity 元素中的 performer 子元素指向工作流参与者实体,将它定义成一个信道。如任务 ShipOrder 的 XPDL 定义中存在 < Performer > Shipper </Performer > ,则就可定义 Shipper. work. shiporder 这个复合事件来表示活动 shiporder 的执行。

Implementation:Activity 元素中的 Implementation 子元素表示任务的执行方式,分别有 No implementation,Tool 以及 Subflow。对于这些不同的任务类型,我们将分别构建不同的 CSP 进程:

1)Activity 的 Implementation 子元素为 No Implementation,即本任务的执行不是由调用的应用或程序支撑的。

2)Activity 的 Implementation 子元素为 Tool,表示任务由一个或多个应用程序执行。

其中 Actual Parameters 子元素代表传递给相应应用程序的参数列表。可在相应应用中查询该参数 MODE 属性是为 IN 还是 OUT,对应参数为输入参数或输出参数。

为以上两种类型的活动定义对应的事件。如: < Activity Id = "36" Name = " Ship Order" > 对应的事件为 act_36。

3)Activity 的 Implementation 子元素为 Subflow,表示该任务为子流程,将子流程活动定义成与其对应的进程。子流程可异步或同步执行。

TransitionRestriction: Activity 元素中的 TransitionRestriction 子元素是用于定义任务的流入转移和流出转移限制。

如果任务中没有该子元素,则该任务所对应的进程可与后置任务进程进行顺序组合。

表1 顺序活动 CSP 映射

顺序活动	XPDL 描述	对应 CSP 进程
Act1 → Act2	< Activities > < Activity Id = "Act1" > ... </Activity > < Activity Id = "Act2" > ... </Activity > </Activities > < Transitions > < Transition Id = "Tra5" From = "Act1" To = "Act2"/ > </Transitions >	let ACT_Act1 = SEQ(act_Act1, act_Act2) within ACT_Act1 |[init. act_Act2] | ACT_Act2

如果任务的 XPDL 定义中存在 TransitionRestriction 子元素,可以分成以下几种情况:

1)接合 Join,描述一个活动有多个流入转移的语义。

(1)Join Type 为 AND,表示接合该任务所有流入转移的并发线程要求同步。只有所有流入路径转移条件为真,任务才被触发执行。

表 2　聚合活动 CSP 映射

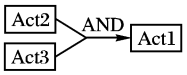
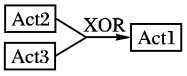
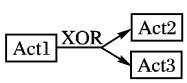
聚合活动	XPDL 描述	对应 CSP 进程
	< Activity Id = "Act1" Name = “同步聚合” > < TransitionRestrictions > < TransitionRestriction > < Join Type = "AND"/ > </TransitionRestriction > </TransitionRestrictions > </Activity > < Transitions > < Transition Id = "Tra5" From = "Act2" To = "Act1"/ > < Transition Id = "Tra6" From = "Act3" To = "Act1"/ > </Transitions >	ACT_Act1 = AJP({ act_Act2, act_Act3}, act_Act1)
	< Activity Id = "Act1" Name = “选择聚合” > < TransitionRestrictions > < TransitionRestriction > < Join Type = "XOR"/ > </TransitionRestriction > </TransitionRestrictions > </Activity > < Transitions > < Transition Id = "Tra5" From = "Act2" To = "Act1"/ > < Transition Id = "Tra6" From = "Act3" To = "Act1"/ > </Transitions >	ACT_Act1 = XJP({ act_Act2, act_Act3}, act_Act1)

表 3　分支活动 CSP 映射

分支活动	XPDL 描述	对应 CSP 进程
	< Activity Id = "Act1" Name = "同步分支" > < TransitionRestrictions > < TransitionRestriction > < Split Type = "AND" > < TransitionRefs > < TransitionRefId = "Tra5"/ > < TransitionRefId = "Tra6"/ > </TransitionRefs > </Split > </TransitionRestriction > </TransitionRestrictions > </Activity > < Transitions > < Transition Id = "Tra5" From = "Act1" To = "Act2"/ > < Transition Id = "Tra6" From = "Act1" To = "Act3"/ > </Transitions >	let ACT_Act1 = ASE(act_Act1, { act_Act2, act_Act3}) within ACT_ Act1 ‖ [init. act _ Act2, init. act _ Act3] ‖ (ACT_Act2 ‖‖ ACT_Act3)
	< Activity Id = "Act1" Name = “选择分支” > < TransitionRestrictions > < TransitionRestriction > < Split Type = "XOR" > < TransitionRefs > < TransitionRefId = "Tra5"/ > < TransitionRefId = "Tra6"/ > </TransitionRefs > </Split > </TransitionRestriction > </TransitionRestrictions > </Activity > < Transitions > < Transition Id = "Tra5" From = "Act1" To = "Act2" > < Condition > orderType == "PO" </Condition > </Transition > < Transition Id = "Tra6" From = "Act1" To = "Act3"/ > < Condition > orderType == "Credit" </Condition > </Transition > </Transitions >	ACT_Act1 = let exp(act_Act2) = orderType == "PO" exp(act_Act3) = orderType == "Credit" within XSE(act_Act1, { act_Act2, act_Act3}) ACT _ Act1 ‖ [init. act _ Act2, init. act _ Act3] ‖(ACT_Act2□ ACT_Act3)

(2)Join Type 为 XOR,表示接合可选线程,不要求同步。当任一流入转移条件为真时,任务即开始执行。

2)分叉 Split,描述一个任务存在多条流出转移的语义。其中 Transition Refs 代表从任务流出的转移列表,每个转移由 ID 进行标识。

(1)Split Type 为 AND,这表示流出的转移表现为许多可能的并发线程。若存在转移条件,则并行执行的实际线程数量取决于与每一转移关联的条件,它们是同时计算的。因此需要把前面定义的 CSP 并行分叉进程与条件转移结合起来,定义如下:

$$ASE(a,X) = PERFORM(a) \rightarrow |||k:X \cdot (\exp(k) \rightarrow init.k) \rightarrow Skip \quad (9)$$

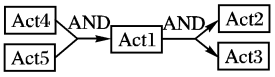
其中 X 为任务流出连接的任务集合,即 Transition Refs 中各转移的 TO 目标任务集合,$\exp(k)$ 则可通过各个转移的 Condition 属性获得。

(2)Split Type 为 XOR,表示任务流出转移的标识符列表提供可选的执行转移。哪条转移路径被选中取决于每一条转移的条件。定义为:

$$XSE(a,X) = PERFORM.a \rightarrow \Box k:X \cdot (\exp(k) \rightarrow init.k) \rightarrow Skip \quad (10)$$

3)既有分叉 Split 又有接合 Join,表示任务存在有多个流入和多个转出。则可把上述定义组合起来即可。

表 4 同步聚合同步分支活动 CSP 映射

同步聚合分支活动	XPDL 描述	对应 CSP 进程
	< Activity Id = "Act1" Name = "同步聚合同步分支" > < TransitionRestrictions > < TransitionRestriction > < Join Type = "AND"/ > </TransitionRestriction > < TransitionRestriction > < Split Type = "AND" > < TransitionRefs > < TransitionRefId = "Tra5"/ > < TransitionRefId = "Tra6"/ > </TransitionRefs > </Split > </TransitionRestriction > </TransitionRestrictions > </Activity > < Transitions > < Transition Id = "Tra5" From = "Act1" To = "Act2"/ > < Transition Id = "Tra6" From = "Act1" To = "Act3"/ > < Transition Id = "Tra7" From = "Act4" To = "Act1"/ > < Transition Id = "Tra8" From = "Act5" To = "Act1"/ > </Transitions >	ACT_Act1 = AJP({act_Act4, act_Act5}, act_Act1) |[init. act_Act1]| ASE(act_Act1, {act_Act2, act_Act3})

2.3 工作流过程元素(Workflow Process)的映射

XPDL 中给出组成工作流过程元素的详细说明。它分别包含任务、转移(可选)、应用以及过程有关数据实体的定义或声明。

对于每一个流程,定义其 CSP 进程为 WF_ID,其中 ID 为流程属性 ID,该进程是由所有活动进程组合起来的。并定义各个集合与流程 XPDL 描述中的各个集合元素相对应,为每个流程分别定义活动集合 W_{wf}、数据集 DS_{wf}、数据类型集合 DTS_{wf} 以及资源集合 RS_{wf}。

3 基于 CSP 的语义属性分析

通过将 XPDL 流程描述转换为 CSP 进程,就可以利用 CSP 的精化概念对工作流模型进行语义分析及检测。

故障偏差求精(Failure Divergences Refinement,FDR)是 CSP 进程的模型自动检测器。FDR 模型检测器将两个 CSP 进程——规约进程和实现进程作为输入,其分析过程就是检查实现进程是否是规约进程的求精。于是,我们就可以定义需要分析检测的各种语义属性进程 SPEC 作为规约,然后使用工具 FDR 来检测断言 $SPEC \sqsubseteq_T WF$ 和 $SPEC \sqsubseteq_F WF$,从而判断工作流模型是否符合该语义属性。以下给出示例:

1)死锁:进程不能进行任何推进。

$$DF \sqsubseteq_T WF,\ DF \sqsubseteq_F WF \quad (11)$$

断言(11)判断进程 WF 是否为死锁进程,其中 $DF = \Pi x: \alpha WF \cdot x \rightarrow DF$[6]。

2)数据匹配:包括数据类型、变量个数、初始值等。

$$DIO \sqsubseteq_T WF \backslash (\Sigma \backslash \alpha DIO),\ DIO \sqsubseteq_F WF \backslash (\Sigma \backslash \alpha DIO) \quad (12)$$

表 5 工作流过程各集合元素

工作流过程 XPDL 描述	各集合元素及其意义
< WorkflowProcess Id = "2" Name = " FillOrder" >	
< Formal parameters > ... </Formal parameters >	Activities 组成过程活动列表
< DataFields > ... </DataFields >	Applications 工作流应用声明列表
< Participants > ... </Participants >	Data Fields 工作流有关数据列表
< Applications > ... </Applications >	Formal Parameters 可传递到过程的参数列表
< Activities > ... </Activities >	Participants 过程执行中的资源列表
< Transitions > ... </Transitions >	Transitions 连接过程任务的转移的列表
</WorkflowProcess >	

断言(12)判断进程 WF 是否符合数据限定属性,其中:

$DIO = init.a \rightarrow in?x:Input(a) \rightarrow resource1.work.a \rightarrow out!y:Output(a) \rightarrow DIO$,$Input(a)$ 和 $Output(a)$ 分别表示活动 a 应符合的输入数据和输出数据条件。

3)资源冲突:为活动定义的执行资源不匹配。

$$GL \sqsubseteq_T WF \backslash (\Sigma \backslash \alpha GL), GL \sqsubseteq_F WF \backslash (\Sigma \backslash \alpha GL) \quad (13)$$

断言(13)判断进程 WF 是否符合资源匹配属性,其中 $GL = resource:R(a) \cdot resource.acquire \rightarrow resource.use \rightarrow perform.a \rightarrow resource.release \rightarrow GL, R(a)$ 表示活动 a 执行所需要的资源。

当然,对于活锁或特定控制流行为属性,以及一些更为复杂和具体的数据及资源冲突等语义属性,也可用 CSP 进程表示,并通过工具 FDR 进行检测。由此可见,CSP 具有刻画系统行为的模型,能够更好地表述模型的语义特性;再者,CSP 具有进行推理的形式演算系统,要比 Petri 网化简规约更适用于模型的计算和语义特性检测;并且由于自动检测工具 FDR 的支持,检测方法也会相对容易。

4　实例说明

下面以工作流过程定义语言——XPDL 规范[3]样例中的一个子流程 FillOrder 的 XPDL 描述为例,如图 1 所示,具体说明 XPDL 描述与 CSP 进程的转化方法。

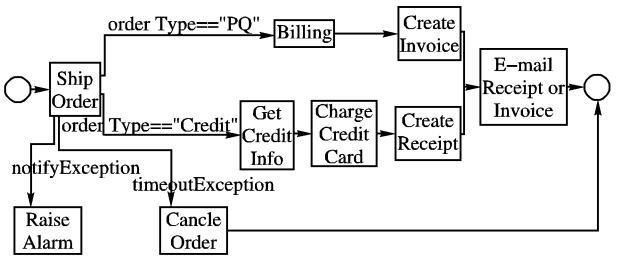


图 1　FillOrder 子流程

流程 FILLORDER 的 XPDL 描述到 CSP 进程的转化过程如下,将所有活动进程用活动名称表示:

1)根据该流程的 XPDL 描述中的各元素,定义 WF_2 表示描述该模型的 CSP 进程,W_{wf} 表示 WF_2 中的活动集,DS_{wf} 表示流程数据集合,DTS_{wf} 表示数据类型集合,RS_{wf} 表示 WF_2 的资源集。用事件 $init.fault$ 表示流程的失败结束,事件 $init.succ$ 表示流程的成功结束[5]。

W_{wf} = { shiporder, raisealarm, cancelorder, billing, createinvoice, getcreditinfo, chargecreditcard, createreceipt, emailreceipteorinvoice}

RS_{wf} = {Shipper, DBConnection, Sytem}

DS_{wf} = { orderNumber, orderType, emailAddress, creditInfo, docURI, Status}

DTS_{wf} = {STRING, INTEGER, CreditInfo}

αWF = { $a:W_{wf}$, $resource:RS_{wf} \cdot init.a$, $resource.work.a$} ∪ {$start$, $complete$, $fail$, $init.fault$, $init.succ$ }

2)根据流程中各个活动 XPDL 描述的各个属性元素

S_{wf} = {SHIPORDER, RAISEALARM, CANCELORDER, PODEAL, CREDITDEAL, EMAIL}

S_{wf} 表示工作流流程的 CSP 进程集合,集合 S_{wf} 中的进程定义如下:

SHIPORDER =
　let exp(raisealarm) = durtime(shiporde) >3 days
　　exp(cancelorder) = durtime(shiporde) >5 days
　　exp(billing) = orderType: PO, exp(getcreditinfo) = orderType: Credit
　within XSE (shiporder, (raisealarm, cancelorder, billing, getcreditinfo))
RAISEALARM = SP(raisealarm, shiporder)
CANCELORDER = SP(cancelorder, fault)
PODEAL = SEQ(billing, createinvoice, emailreceiptorinvoice)
CREDITDEAL = SEQ(getcreditinfo, chargcreditcard, createreceipt, emailreceiptorinvoice)
EMAIL = SP(emailreceiptorinvoice, succ)

3)将各个活动进程进行组合,得到 CSP 进程 WF_2 为:

WF_2 =
　let
　　START = $start \rightarrow init.shiporder \rightarrow Skip$
　　FIN = $init.succ \rightarrow complete \rightarrow Skip$
　　FAULT = $init.fault \rightarrow cancel \rightarrow Skip$
　　WF_C = (SHIPORDER |[$init.billing$, $init.getcreditinfo$, $init.raisealarm$, $init.cancelorder$]| (PODEAL□CREDITDEAL□RAISEALARM□CANCELORDER)) |[$init.emailreceiptorinvoice$]| EMAIL
　within (START |[$init.shiporder$]| (WF_C |[$init.succ$, $init.fault$]| (FIN□FAULT))); WF

4)最后,利用 FDR 以及 CSP 的精化特性,可以很容易对转化后的 CSP 进程进行语义分析和检测。如前面定义:$DF = \Pi x:\alpha WF \cdot x \rightarrow DF$;$DIO = init.a \rightarrow in?x:Input(a) \rightarrow resource1.work.a \rightarrow out!y:Output(a) \rightarrow DIO$,我们可以检测断言 $DF \sqsubseteq_T WF\_2$,$DF \sqsubseteq_F WF\_2$ 以及 $DIO \sqsubseteq_T WF\_2 \backslash (\Sigma \backslash \alpha DIO)$,$DIO \sqsubseteq_F WF\_2 \backslash (\Sigma \backslash \alpha DIO)$ 分别用于分析流程死锁以及活动输入输出数据是否匹配的特性。

5　结语

在工作流研究领域存在着各式各样的工作流模型描述方法,而 CSP 基于严密的语义特性表达能力和推理演算分析能力显示出其优越性,有利于对工作流模型进行语义描述分析和检测。XPDL 是工作流管理联盟定义的 XML 格式的过程定义交换语言,它具有简单,适应性强和标准化程度高的特性已经得到广泛的应用,各种工作流产品对 XPDL 的支持也得到大大强化[1]。因此,研究 XPDL 到 CSP 进程的映射,是有其理论和现实的意义的。本文提出了如何将 XPDL 流程描述成 CSP 进程的方法,从而对任何符合 XPDL 标准的工作流模型均可以映射到 CSP 进程,以利用进程代数 CSP 的成熟理论对模型进行分析。进一步的工作是使用扩展时间的 CSP 对模型中的时间特性进行描述,以取得更完整的模型特性分析。

参考文献:

[1] 唐邦志,魏生民,景韶宇,等. 工作流网 XPDL 映射[J]. 计算机工程与应用,2003,39(36):41 - 44.

[2] GUO LI-HUA, Lü ZHAO, GU JUN-ZHONG. A semantic modeling and verification approach of workflow process based on CSP[C]// Proceedings of the 2nd International Conference in Communications and Networking in China. Piscataway: IEEE Press, 2007:165 - 169.

[3] Workflow Management Coalition. Workflow process definition interface-XML process definition language, WfMC-TC-1025[R]. Hingham, MA: WFMC, 2001.

[4] HOARE C A R. Communicating sequential processes[M]. New Jersey: Prentice-Hall, 1985.

[5] Formal Systems (Europe) Ltd. Failures-divergences refinement[EB/OL]. [2007 - 05 - 12]. http://www.fsel.com/documentation/fdr2/html/index.html.

[6] WONG P Y H, GIBBONS J. A process-algebraic approach to workflow specification and refinement[C]// Proceedings of the 6th International Symposium on Software Compositon, LNCS4829. Berlin: Springer, 2007:51 - 65.