业务过程的自动配置与管理

吴亚洲 杨启帆 蒋建民 张 仕

(福建师范大学数学与计算机科学学院 福州 350007)

摘 要 一个可配置的业务过程模型可以通过配置来满足组织的特定要求。配置活动需要自动确定一个可配置过程模型的变体,同时保证该特定过程模型的正确性。然而,很少有方法能够解决此问题。提出一种新方法,即先将可配置的过程模型自动分离成原子子过程模型,再由这些模型组合成需要的过程模型,该方法能保证得到的过程模型的正确性(该过程模型无异常行为问题)。与现有的方法相比,新方法与特定的语言无关,而且避免了独立处理配置活动,降低了计算复杂度。

关键词 可配置过程模型,配置,原子,子过程模型

中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.7.028

Automatic Configuration and Management of Business Process

WU Ya-zhou YANG Qi-fan JIANG Jian-min ZHANG Shi

(School of Mathematic and Computer Science, Fujian Normal University, Fuzhou 350007, China)

Abstract A business process model can be configured to meet the specific requirements of the organization through the configuration. Automatic configuration activities need identifying variants of a process model which can be configured, while ensuring the correctness of this particular process model. However, there are few methods to solve this problem. This paper proposed an innovative approach for automatically separating a configurable process model into atomic and correct sub-process models without abnormal behavioral problems. Compared with existing approaches, our approach avoids independently handling the configuration activity and reduces computational complexity. Moreover, our approach is language-independent.

Keywords Configurable process models, Configuration, Atomic, Sub-process models

1 引言

业务过程的配置管理有助于业务过程管理系统的供应商节省成本和时间,避免从头开始创建过程模型。但是,复杂的手工配置活动不但需要花费较大的代价,而且容易出错。因此,需要为可配置的业务模型提供一个自动的可配置的管理工具。

配置管理来源于软件产品生产线工程[14,15,19]。现有的配置方法侧重于业务过程的配置,并且与配置过程模型的生命周期相关。这方面的研究主要集中在两个方面:一方面,文献[6,8,9,12,13,16,17]中的方法通过增加、删除、修改业务活动来实现配置。然而,当一个业务过程模型在应用环境下处理多组织过程时(如在云计算环境),组织需要重构标准的业务过程模型是不切实际的[2]。另一方面,文献[1-5]中的方法只是通过限制一些业务活动的执行来得到配置。本文的方法类似于后者,即通过业务过程的依赖关系将可配置的过程模型自动分离成原子和正确的子过程(子过程模型无异常行为问题)。这些原子的子过程可以组合成该可配置模型的所有可能的变体。这些变体可以满足组织的特定的要求。

与目前的方法相比较,本方法由于在设计时将配置活动纳入到一个可验证的过程模型,因此获得了该过程模型的所有可行的配置。我们的方法避免了单独处理配置活动,从而降低了计算的时间复杂度。也就是说,配置活动不需要建立一个独立的算法来计算配置。而且我们的配置方法与特定的语言无关,并且已经开发了一个原型工具来实现自动配置过程模型。最后,本文的形式化框架与其它主流框架(例如 Petri 网、进程代数和自动机模型等)相比,具有更强的表达能力,具体请考阅文献[21]。

本文第 2 节给出了所要讨论的应用实例;第 3 节介绍了一种称作依赖结构的形式模型;第 4 节给出了可配置的形式化的定义,并确保配置正确的解决方案;第 5 节用实例验证了该方法的正确性;第 6 节是相关工作;最后总结全文。

2 启发性示例

该例子是一个简单的旅游管理系统,其包含旅游代理服务(TAS)和银行服务(BS)两个网络服务,如图 1(a)所示。客

到稿日期:2014-07-14 返修日期:2014-10-25 本文受国家自然基金(61175123),福建省自然基金(2012J05112,2014J01221),上海市可信计算重点实验室开放项目(07dz22304201401)资助。

吴亚洲(1989—),男,硕士生,主要研究方向为软件工程、形式化方法,E-mail:cst.wyz@gmail.com(通信作者);杨启帆(1990—),男,硕士生,主要研究方向为软件工程、形式化方法;蒋建民(1972—),男,博士,副教授,主要研究方向为软件工程、形式化方法;张 仕(1977—),男,博士,副教授,主要研究方向为软件工程建模、分布式并行计算、服务计算。

户通过消息的传递与两个服务相互作用。

客户先发送查询请求消息(*Qreq*)给 TAS。这可能出现一种情况,即使得用户发送的请求不能被执行。在这种情况下,用户从 TAS 中接收一个不可用的通知(*Nan*);否则,TAS 返回查询结果,即查询反馈消息(*Qres*)。最后,旅程查询过程结束。

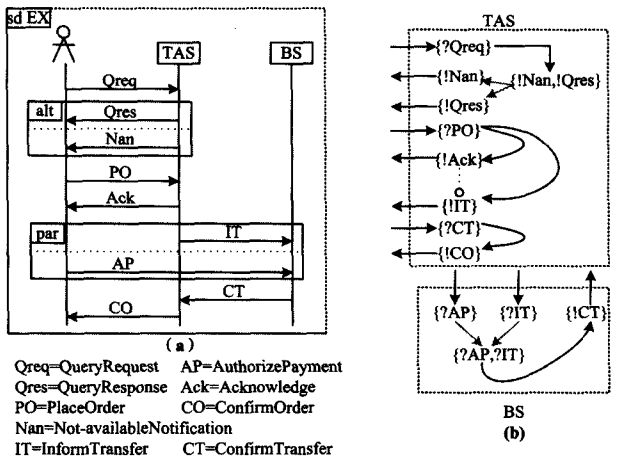

图 1 旅游管理系统

订单过程需要客户通过旅游管理系统下订单(即发送消息 *PO*)。当 TAS 接收到订单后,会发送确认消息 *Ack* 给客户,与此同时,代理向 BS 发送要求转账消息(*IT*)。但是 TAS 于转账消息之前发送确认消息。

在支付过程中,TAS 需要银行执行转账操作。转账消息包括目标账户和转账的金额。但是银行必须在收到确认消息信用卡持卡人的确认消息(*AP*)和 TAS 发送的转账消息(*IT*)后才进行转账支付。如果银行顺利执行转账,确认消息(*CT*)将会被回发送至旅游代理。最后,旅游代理发送订单确认消息(*CO*)给客户。

旅游管理系统通过隐藏和限制业务活动来得到系统的配置,但得到的配置不一定是可行的,而且计算所有配置所需要的时间复杂度也是呈指数级的[1,3-5]。如果把旅游管理系统分离成为查询、订单和支付子系统,且这 3 个子系统都是可行的,显然 3 个子系统可能被用在不同的组织中。然而,我们必须回答的问题是:假设对于一个给定系统,如何可以得到该系统中的所有可行的子系统?又如何可以得到该系统的所有可行的配置呢?

3 形式化模型

通常事件是指活动或动作的出现,而本文的事件是带有数据信息的事件。数据信息不是一种具体的值,而是一种抽象符号。在本文中,事件通常描述消息,事件的数据部分是不可见的。在服务的交互规范中给定一个消息 m(组成或者对象)对应 3 个不同的事件 $!m$、$?m$ 和 $!?m$,分别表示消息 m 对应的发送、接收和存储事件。例如,在第 2 节中消息 *Qreq* 相对应的 3 个事件为 $!Qreq$、$?Qreq$ 和 $!?Qreq$。

这里采用了一个叫做"依赖结构"的模型,该模型以前称为协议结构[20,21],重新命名该模型的主要目的是为了避免词汇和概念混淆。

3.1 依赖结构

为了方面叙述,本文首先给出一些辅助性的记法。假定 M 是事件的集合,$|M|$ 和 $P'(M)$ 分别表示集合 M 中元素的个数和集合 M 的幂集,而且 $P(M)=P'(M)\backslash\{\emptyset\}$。一个"事件集"是指事件的集合,如果一个事件集中的所有事件都已经出现,则称该事件集是可用的,否则称它是不可用的。

定义 1 依赖结构(*DS*)是一个九元组$\langle M,I,T,S,C,\mathbb{P},Out,In,F\rangle$,其中,

- M 是非空的、有限的事件集的集合;
- $I\subseteq P'(M)$是初始可用的事件集的集合;
- $T\subseteq P(M)\times P(M)$是转换关系,并且满足:$\forall A,B\in P(M):(A,B)\in T\Rightarrow A\not\subseteq B\wedge B\not\subseteq A$;
- $S\subseteq P(M)\times P(M)$是同步关系,并且满足:$\forall A,B\in P(M):(A,B)\in S\Rightarrow A\subset B$;
- $C\subseteq P(M)\times P(M)$是选择关系,并且满足:$\forall A,B\in P(M):(A,B)\in C\Rightarrow A\supset B$;
- $\mathbb{P}\subseteq P(M)\times P(M)$是优先级关系,并且满足:$\forall A,B\in P(M):(A,B)\in\mathbb{P}\Rightarrow(\exists A'\in P(M):(A,A')\in T\cup S\cup C\vee(A',A)\in T\cup S\cup C)\wedge(\exists B'\in P(M):(B,B')\in T\cup S\cup C\vee(B',B)\in T\cup S\cup C)$;
- $Out\subseteq P'(M)$是输出接口中事件集的集合;
- $In\subseteq P'(M)$是输入接口中事件集的集合,并且$I\cap In=\emptyset$和$Out\cap In=\emptyset$;
- $F\subseteq P'(M)$是最后可用的事件集的集合。

这里,$\forall A,B\subseteq M,(A,B)\in T$(或$S$、$C$、$\mathbb{P}$)被称为一个是转换依赖(或同步依赖、选择依赖、优先级依赖),分别记作$A\rightarrow_t B$(或$A\rightarrow_s B$、$A\rightarrow_c B$、$A\rightarrow_p B$),都读作"B 依赖 A",并且 A、B 分别称作依赖(A,B)的前置集和后置集。A 中的事件与 B 中的事件分别称作依赖(A,B)的前置事件集和后置事件集。事件集 $A\in In$(或 $A\in Out$)称作输入事件集(或输出事件集)。

为区分事件之间的依赖关系,该模型将依赖关系分为以下 4 类:转换依赖、同步依赖、选择依赖和优先级依赖。将依赖关系分类的目的在于区分顺序依赖、互斥依赖、平行依赖和优先级依赖。该思想是基于对进程代数组合运算、*Petri* 网同步等待机制(对应于同步依赖)和事件结构中的事件依赖(顺序和互斥)关系(对应转换依赖和选择依赖)。优先级依赖用于建模调度策略,限制转换、同步和选择依赖。如果先前被提到的依赖在事件中不存在,那么这些事件之间的关系是独立的(即并行的)。因此,依赖结构可以建模真并发。

如图 1(b)所示,依赖结构可以用图形化的方式来表示,例如,在这些图中,事件集以集合的形式出现,选择(同步或转换)依赖用从一个事件集指向另一个事件集的有向实线来表示,而优先级依赖则使用终端带有一个小圆圈的虚线来表示。

依赖结构用下面的方法描述一个系统(包括服务)。其中,事件是一个带有数据信息的事件,事件集是指系统中的事件的集合。最初可用事件集表示在系统开始运行前这些事件已经出现。转换依赖表示事件的因果关系。同步依赖表示多个事件的同步等待和数据合并。选择依赖表示事件之间相互排斥的关系。优先级依赖在系统中控制事件出现的先后顺序。事件的输出和输入接口表明当系统与外部环境发生通信时,事件出现在输出和输入接口的形式表现为发送和接收事件。最后可用事件集表示当正在执行的系统或者子系统停止运行时所出现的事件。

3.2 执行语义

依赖结构的执行语义模拟了所建模系统的执行过程。正如前文所述，在依赖结构中的每一个依赖（不包含优先级依赖）对应于一个活动（操作或动作），也就是一个执行步骤。一个依赖只有在它的前置集可用的前提下才可能被执行。为此，将计算过程中每个中间步骤下的当前可用的事件集以及当前已被激活的依赖作为一个整体，称为一个“状态”。

定义 2 设 $DS=\langle M,I,T,S,C,P,Out,In,F\rangle$ 是一个依赖结构，DS 的一个状态是一个二元组 $S=\langle\Delta,\Gamma\rangle$，其中，$\Delta\subseteq P'(M)$ 是可用事件集的集合，$\Gamma\subseteq T\cup S\cup C$ 是已激活的依赖事件集合，并且满足 $\forall(A,B)\in\Gamma\Rightarrow A\in\Delta$。$DS$ 的初始化状态是 $S_0=\langle\Delta_0,\Gamma_0\rangle$，其中，$\Delta_0=I$ 和 $\Gamma_0=\{(A,B)\mid A\in I,(A,B)\in T\cup S\cup C\}$。

显然，对于一个依赖结构的初始状态，由于所有的依赖都还未被执行，它由所有初始情况下可用的事件集以及由这些初始可用的事件集作为前置集的依赖构成。

不同类型的依赖在执行语义上存在着一些差异。当转换依赖被执行后，它们的后置集就变得可用。而在同步依赖被执行前，所有相互同步的消息必须都可用并且对应的同步依赖已被激活。对于拥有同一个后置集的两个同步依赖，若其中的一个被执行，那么另外一个也必须被同时执行；否则，其中的任何一个都得不到执行。也就是说，对于相互同步依赖的事件，它们的同步依赖必须是一起被执行。选择依赖则表示的是互斥的选择，也就是说，只要其中的一个选择依赖被执行，拥有相同前置集的其余的选择依赖就将变为没有被激活的依赖。

另一方面，外部环境也可以通过输入事件来改变一个系统的状态。当系统被一个外部事件触发，这个事件将使一个输入事件集变为可用的事件集，进而使以该输入事件集作为前置集的依赖被激活。但是所有能够得到执行的被激活的依赖必须用优先级依赖来加以控制，也即让优先级依赖来决定被激活的依赖的执行顺序。为方便描述，将一个可以使输入事件集 W 可用的触发事件记作“$re(W)$”，称“接收到一个事件集 W”。由此，给出了以下的定义。

定义 3 设 $DS=\langle M,I,T,S,C,P,Out,In,F\rangle$ 是一个依赖结构。

(1)对于 DS 的一个状态 $S=\langle\Delta,\Gamma\rangle$ 是可演化的，如果存在一个依赖 $(A,B)\in\Gamma$ 满足以下条件（$^*B=\{V\in P(M)\mid(V,B)\in\Gamma\cap S\}$ 时，

(i) $(A,B)\in T\cup C$，或者 $(A,B)\in S\wedge\bigcup_{W\in{}^*B}W=B$；

(ii) $\exists C,D\subseteq M:(C,D)\in\Gamma\wedge(D,B)\in P$；

(iii) $\exists E,F\subseteq M:(E,F)\in\Gamma\wedge(E,A)\in P$。

(2)如果存在一个消息集 $W\in In$，使得 $W\notin\Delta$，则称 $S=\langle\Delta,\Gamma\rangle$ 是可输入触发的。

当前可以执行一个或多个依赖的状态称为可演化的状态。在输入接口不存在可用事件的状态称为可输入触发状态。接下来将给出依赖结构的执行规则。

定义 4 设 $DS=\langle M,I,T,S,C,P,Out,In,F\rangle$ 是一个依赖结构，并且 $S_1=\langle\Delta_1,\Gamma_1\rangle$ 与 $S_2=\langle\Delta_2,\Gamma_2\rangle$ 是 DS 的两个状态。有如下情形：

(1)如果 $(A,B)\in\Gamma_1$，S_1 是可演化的，$\Delta_2=\Delta_1\cup\{B\}$，并且

$$\Gamma_2=\begin{cases}(\Gamma_1\backslash\{(A,B)\})\cup\Gamma'':(A,B)\in T\\(\Gamma_1\backslash\{(X,B)\mid(X,B)\in S\})\cup\Gamma'':(A,B)\in S\\(\Gamma_1\backslash\{(A,Y)\mid(A,Y)\in C\})\cup\Gamma'':(A,B)\in C\end{cases}$$

则称 S_1 能通过执行依赖 (A,B) 演化成 S_2，记作 $S_1\xrightarrow{(A,B)}S_2$。这里，$\Gamma''=\{(B,Y)\mid(B,Y)\in T\cup S\cup C\}$。

(2)如果 $W\in In$，S_1 是可输入触发的 $\Delta_2=\Delta_1\cup\{W\}$ 且 $\Gamma_2=\Gamma_1\cup\{(W,X)\mid(W,X)\in T\cup S\cup C\}$，则称 S_1 能通过接收事件集 W 演化成 S_2，记作 $S_1\xrightarrow{re(W)}S_2$。该定义实际上给出了一种计算所有可能状态的方法。

3.3 性质

依赖结构可以用于分析系统的行为和性质。本节给出一些依赖结构的性质。

定义 5 设 $DS=\langle M,I,T,S,C,P,Out,In,F\rangle$ 是一个依赖结构，并设 S_0 是 DS 的初始状态。

(i)如果在 DS 中存在一个状态 S 并且存在一个状态序列 $S_1,\cdots,S_{n-1}$，使得 $S_0\xrightarrow{d_1}S_1\cdots S_{n-1}\xrightarrow{d_n}S(d_i\in T\cup S\cup C\cup\{re(V)\mid V\in In\},i\in\{1,\cdots,n\})$，则称状态 S 在 DS 中是可达的。$Sta(DS)$ 表示 DS 中所有状态的集合。

如果在 DS 中存在两个状态 S 和 S' 并且存在状态序列 $S_1',\cdots S_{n-1}'$，使得 $S'\xrightarrow{d_1'}S_1'\cdots S_{n-1}'\xrightarrow{d_n'}S(d_1'\in T\cup S\cup C\cup\{re(V)\mid V\in In\},i\in 1,\cdots,n))$，则状态 S' 可达至状态 S，表示为 $S'\xrightarrow{*}S$。

如果存在一个依赖序列 $\sigma=d_1\cdots d_n(d_i\in T\cup S\cup C\cup\{re(V)\mid V\in In\},i\in 1,\cdots,n\})$，使得 $S_0\xrightarrow{d_1}S_1\cdots S_{n-1}\xrightarrow{d_n}S_n$，则称 σ 是依赖结构 DS 中的一个迹。$|\sigma|=\{d_1,\cdots d_n\}$，表示迹的所有依赖集合。$Tr(DS)$ 表示 DS 的所有迹的集合。

(ii)设 $S=\langle\Delta,\Gamma\rangle\in Sta(DS)$ 并且 $\exists S'\in Sta(DS):S'\xrightarrow{d}S$。如果 $d=(A,B)\in T\cup S\cup C$ 或 $d=re(W)\in\{re(V)\mid V\in In\}$，则称 B 或 W 是 S 的“最近的事件集”。这里定义空集 $\emptyset$ 作为初始状态 S_0 最近的事件集。

(iii)设 $S=\langle\Delta,\Gamma\rangle\in Sta(DS)$ 并且 X 是事件集 S 的最近的事件集。如果 $\Gamma=\emptyset$，并且 S 的最近的事件集 $X\in F$，则称 S 是终止的。如果在 DS 中存在一个状态 S 不是终止的，不是可演化的，也不是输入触发的，则称 S 是死的。

如果在 DS 中存在一个状态 S 是死的，则称 DS 是死锁的；相反，如果在 DS 中任何状态都不是死的，则称 DS 是无死锁的。

对于所有的状态 $S\in Sta(DS)$，S 是终止的，或存在一个终止状态 $S_t\in Sta(DS)$ 使得 $S\xrightarrow{*}S_t$，并令优先级依赖 P 为空集，则称 DS 是弱终止的。

3.4 业务过程建模

如图 1(b)所示，依赖结构用图形化表示两个服务 TAS 和 BS。为了简单起见，只使用图 1(b)中的依赖。

当 TAS 接收到查询请求 $Qreq$ 时，它将 $Qreq$ 发送给 $Qres$ 和 Nan 的同时，产生转换依赖 $\{?Qreq\}\rightarrow_t\{!Nan,!Qres\}$，同时产生两个选择依赖 $\{!Nan,!Qres\}\rightarrow_c\{!Nan\}$ 和 $\{!Nan,!Qres\}\rightarrow_c\{!Qres\}$。一旦 TAS 接收到订单，就立刻发送 Ack 给客户，它可以直接发送 IT 到 BS 用于现金转移，但是 TAS 先于 IT 发送 Ack。那么，可以得到两个转换依赖：$\{?PO\}\rightarrow_t\{!Ack\}$，$\{?PO\}\rightarrow_t\{!IT\}$。当消息 CT 被输入，与

它相对应的消息 CO 将会返回给客户，这两个消息之间的关系是转换依赖的关系$\{?CT\}\rightarrow_t\{!CO\}$。那么，服务 TAS 可以表示为 $DS_{TAS}=\langle M,I,T,S,C,P,F\rangle$，其中 $M=\{?Qreq,!Nan,!Qres,?PO,!Ack,!IT,?CT,!CO\}$，$I=\emptyset$，$T=\{(\{?Qreq\},\{!Qres\}),(\{?PO\},\{!Ack\},(\{?PO\},\{!IT\}),(\{?CT\},\{!CO\})\}$，$S=\emptyset$，$C=\{(\{!Nan,!Qres\},P=\emptyset,Out=\{\{!Nan\},\{!Qres\},\{!Ack\},\{!IT\},\{!CO\},In=\{\{!Qreq\},\{?PO\},\{?CO\}\},F=\{\{!Nan\},\{!Qres\},\{!Ack\},\{!IT\}\}$。

BS 要先同时接收 IT 和 AP，然后在将两个消息发送给 CT，所以 IT 和 AP 必须合并。那么，得到同步依赖和转换依赖：$\{?IT\}\rightarrow_s\{?IT,?AP\},\{?AP\}\rightarrow_s\{?IT,?AP\},\{?IT,?AP\}\rightarrow_t\{!CT\}$。服务 BS 表示为 $DS_{BS}=\langle M',I',T',S',C'P',Out',In',F'\rangle$，其中 $M'=\{?IT,?AP,!CT,\}$，$I'=\emptyset$，$T'=\{(\{?IT,?AP\},\{!CT\})\}$，$S'=\{(\{?IT\},\{?IT,?AP\}),(\{?AP\},\{?IT,?AP\})\}$，$C'=\emptyset$，$P'=\emptyset$，$Out'=\{\{!CT\}\}$，$In'=\{\{?AP\},\{?IT\}\}$，$F'=\{\{!CT\}\}$。

4 配置

4.1 配置的正确性

本节开始讨论通过依赖结构取得可配置的正确性。本文的方法通过限制行为[1,3-5]来实现配置，即首先将配置过程模型分为原子和正确的子过程模型(没有异常行为的问题)，然后将这些与特定要求相关的原子子过程模型归化为组织提供的特定的过程模型。

首先给出依赖结构的基本定义，然后介绍两种依赖结构结合的概念。

定义 6 定义两种依赖结构为 $DS_1=\langle M_1,I_1,T_1,S_1,C_1,P_1,Out_1,In_1,F_1\rangle$ 和 $DS_2=\langle M_2,I_2,S_2,C_2,P_2,Out_2,In_2,F_2\rangle$。

如果 $M_1\subseteq M_2$，$I_1\subseteq I_2$，$T_1\subseteq T_2$，$S_1\subseteq S_2$，$C_1\subseteq C_2$，$P_1\subseteq P_2$，$Out_1\subseteq Out_2$，$In_1\subseteq In_2$，$F_1\subseteq F_2$ 和 $I_1\cup In_1\neq\emptyset\wedge F_1\neq\emptyset$，则 DS_1 是 DS_2 的子结构，表示为 $DS_1\prec DS_2$。

如果$(T_1\cup S_1\cup C_1)\cap P_2=\emptyset)\wedge((T_2\cup S_2\cup C_2)\cap P_1=\emptyset)$，则 DS_1 和 DS_2 的并定义为 $DS_1\uplus DS_2=\langle M',I',T',S',C',P',Out',In',F'\rangle$。这里 $M'=M_1\cup M_2$，$I'=I_1\cup I_2$，$T'=T_1\cup T_2$，$S'=S_1\cup S_2$，$C'=C_1\cup C_2$，$P'=P_1\cup P_2$，$Out'=Out_1\cup Out_2$，$In'=In_1\cup In_2$ 和 $F'=F_1\cup F_2$。

很明显，并运算满足结合律和交换律。通常情况下，仅有一个过程模型是可以配置的。因此，一个可配置的模型应该是弱终止的，并且至少包含一个可行的子过程模型。

定义 7 设 DS 是一个弱终止的依赖结构，如果存在一个弱终止的依赖结构 $DS^s\prec DS$ 使得 $\exists\sigma\in Tr(DS):\sigma\notin Tr(DS^s)$，则称 DS 是一个可配置的依赖结构。

它如果是一个可配置的依赖结构，则意味着可以分成一些子结构。基于这些性质，可以得到如下定理。

定理 1 如果 DS 是一个可配置的依赖结构，则存在两个弱终止的结构 $DS^s,DS^{s'}\prec DS$ 使得 $DS^s\uplus DS^{s'}=DS$。

证明：由定义 7 可知 DS 是一个可配置的，则可以假定存在一个弱终止的依赖结构 $DS^s\prec DS$ 使得 $\exists\sigma\in Tr(DS):\sigma\notin Tr(DS^s)$。因此我们可以构造一个弱终止的 $DS^{s'}$ 使得 $\sigma\in Tr(DS^{s'})$ 且 $DS^s\uplus DS^{s'}=DS$。

该定理表明一个可配置的依赖结构至少可以划分为两个弱终止的子结构。它可以用来证明定理 3。接下来讨论的是多个依赖结构联合下的弱终止的情况。

定理 2 设 DS 是一个可配置的依赖结构。如果存在 n 个 $DS_1^s,\cdots,DS_n^s$ 依赖结构，对于 $i\in\{1,\cdots,n\}$，$DS_i^s\prec DS$，DS_i^s 是弱终止的，则 $DS_1^s\uplus\cdots\uplus DS_n^s$ 是弱终止的。

证明：设在 $i\in\{1,\cdots,n\}$ 的情况下，依赖结构 $DS_i^s=\langle M_i,T_i,S_i,C_i,P_i,Out_i,In_i,F_i\rangle$ 的初始状态集为 $S_{0i}=\langle\Delta_{0i},\Gamma_{0i}\rangle$。$DS_1^s\uplus\cdots\uplus DS_n^s=DS$ 的初始状态集 $S_0=\langle\Delta_0,\Gamma_0\rangle$。由定义 6 知，$I_1\cup\cdots\cup I_n=I$，$T_1\cup\cdots\cup T_n=T$，$\cdots$，$In_1\cup\cdots\cup In_n=In$ 和 $F_1\cup\cdots\cup F_n$，且由定义 2 知 $\Delta_{0i}\cup\cdots\cup\Delta_{0n}=\Delta_0$，$\Gamma_{0i}\cup\cdots\cup\Gamma_{0n}=\Gamma_0$。因此，通过定义 5 得到，对于任意的 $S=\langle\Delta,\Gamma\rangle\in Sta(DS_1^s\uplus\cdots\uplus DS_n^s=DS$，存在一个状态顺序 $S_1,\cdots,S_n$(这里对 $i\in\{1,\cdots,n\}$，有 $S_i=\langle\Delta_i,\Gamma_i\rangle\in Sta(DS_1^s\uplus\cdots\uplus DS_n^s=DS$)和依赖顺序 $d_1,\cdots,d_n$(d_i 是对 $i\in\{1,\cdots,n\}$，$DS_1^s\uplus\cdots\uplus DS_n^s=DS$)，使得 $S_0\xrightarrow{d_1}S_1\cdots\xrightarrow{d_n}S_n$，$S_n=S$。

当 d_1 执行、S_1 发生时，假设这里存在 $k(k\geqslant1)$ 个依赖结构，且$\{DS^{s1},\cdots,DS^{s*}\}\subseteq\{DS_1^s,\cdots,DS_n^s\}$，由定义 7 知，$d_1$ 被执行时，$k(k\geqslant1)$ 个依赖结构状态发生改变。剩下的 $n-k$ 个依赖结构也以此类推。所以，在对任意 $i\in\{1,\cdots,n\}$，$S_i\in S_i\in Sta(DS_i^s)$ 且 S_i 不是死的情况下，由定义 5 知 $DS_1^s\uplus\cdots\uplus DS_n^s$ 是弱终止的。

定理 2 表明对于可配置的依赖结构，其子结构的并集也是弱终止的。这两个定理证明了一个依赖结构所有可行配置的正确性。

4.2 自动配置和所有的可行配置

本小节将讨论给定一个依赖结构如何得到该依赖结构的所有可行配置。这里，先给出一个依赖结构的原子结构的定义。

定义 8 DS 是一个弱终止的依赖结构，如果 DS 是不可配置的，则说明 DS 是原子的。

接下来，讨论一个依赖结构和它原子子结构之间的关系。

定理 3 如果 DS 是可配置的依赖结构，则这就存在弱终止的依赖结构 $DS_1^s,\cdots,DS_n^s\prec DS$，这里对于 $i\in\{1,\cdots,n\}$，DS_i^s 是原子的，且 $DS_1^s\uplus\cdots\uplus DS_n^s=DS$。

证明：因为 DS 是一个可配置的依赖结构，可以通过定理 1，假设存在两个弱终止依赖结构 $DS^s,DS^{s'}\prec DS$ 且 $DS^s\uplus DS^{s'}=DS$，所以可以将 DS 分成两个弱终止的子结构 DS^s 和 $DS^{s'}$ 且 $DS^s\uplus DS^{s'}=DS$。如果 DS^s 不是原子的，但是可以配置的，则继续由定理 1 可以将 DS^s 划分成 $DS_1^s\uplus DS_2^s=DS^s$。显然，$DS_1^s\uplus DS_2^s\uplus DS^{s'}=DS$。同理将 $DS^{s'}$ 划分为 $DS_1^{s'}$ 和 $DS_2^{s'}$，则 $DS_1^s\uplus DS_2^s\uplus DS_1^{s'}\uplus DS_2^{s'}=DS$。以此类推，可以用同样的方式将所得到的子结构进行划分，直至它们变为原子的子结构。因此，对于一个可以配置的依赖结构，可以得到 DS 的所有原子的子结构，并且所有原子的子结构的并为 DS。

该定理表明一个可配置的依赖结构可以由一些原子的子结构组成，且所有的原子的子结构并集是该可配置依赖结构的本身。

定理 4 如果 DS 是一个可配置的依赖结构，它由 n 个原子结构 $DS_1^s,\cdots,DS_n^s$ 组成，即 $DS_1^s\uplus\cdots\uplus DS_n^s=DS$，则对于所有的 $DS'\prec DS$，存在 k 个原子子结构 $DS_1',\cdots,DS_k'$ 且 $\{DS_1',\cdots,DS_k'\}\subseteq\{DS_1^s,\cdots,DS_n^s\}$，$DS_1'\uplus\cdots\uplus DS_k'=DS'$。

证明：显然，由定理 3 可知，可以将一个依赖结构的所有原子的子结构，通过并运算组合成它的所有可能的子结构（包括依赖结构本身）。而且，由定理 2 知，得到的所有子结构也是弱终止的。由此，该定理得证。

通过该定理可以得出，用一个依赖结构的所有原子子结构组合成它的所有可能性的子结构（包括依赖结构本身）。而且，由定理 2 可知，所有的子结构都是弱终止的。事实上，每个子结构都是依赖结构的变体，并且对应一个配置。因此，所有的配置都应该能够获取到。例如，如果 DS 是一个可配置的依赖的结构，它有 n 个原子的子结构，即有 $C_n^1+\cdots+C_n^n$ 个子结构，也就是 DS 有 $C_n^1+\cdots+C_n^n$ 个配置，所以，该方法不同于现有的方法[1,3-5]。计算子结构的并的时间复杂度是多项式的。从一个依赖结构得到它的所有的可行性的原子的子结构是最复杂的。

5 实例

在以前的工作中，我们已经开发了一个可达性分析的算法[21]，并在文献[22]进行改进；同时，在工具 DAT（依赖分析工具）中已经实现。该算法的时间复杂度呈指数关系。在设计运行时，它能同时获得所有可能的配置。

采用文献[21]中的算法，分析第 2 节所给的应用实例（见图 1(b)）。该系统的依赖结构模型可以获得 4 个子结构。为了简化分析过程，本文只给出有依赖关系的消息。图 2 表示的是 4 个子结构：查询的子结构并不能成功返回结果（见图 2(a)）；查询的子结构可以成功返回结果（见图 2(b)）；预定的子结构没有支付（见图 2(c)）；预定的子结构有支付（见图 2(d)）。显然，这 4 个子结构都是原子的。

该实例表明本方法与实际情况一致。一些旅行社企业（组织）可能需要的旅游管理系统只有旅游查询服务，而其它的旅行社可能需要旅游管理系统有查找与预定的服务。甚至一些公司需要旅游管理系统包含旅游查询、预定和支付这 3 项服务。而这些不一样的需求都可以通过自动配置来满足。

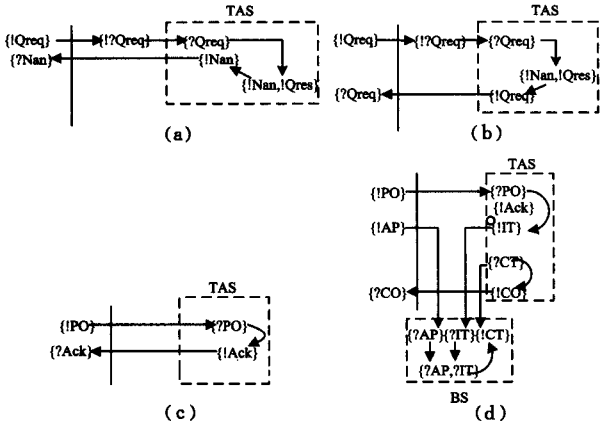

图 2 4 个子结构

6 相关工作

配置是软件产品线工程的基本活动，它为软件系统的通用性和可变性提供了支持。文献[14,15,19]提出了手工的可行的配置的解决方案，但是这需要验证这些方法来确保配置的正确性。在业务过程模型领域，相关的工作主要集中在两个方面。一方面，文献[6,8,9,12,13,16,17]认为在特定的参考过程模型演化过程中，可以通过增加、修改、删除来实现配置。这些方法都考虑配置在业务建模过程中是一个独立的活动，而本文的工作是将配置活动作为可达性分析活动的一部分。另一方面，文献[1,3-5]通过限制行为（模块化和隐藏）来实现配置。本文的工作类似于后者。Van der Aalst 等人在过去的十年，已经提出了许多基于业务过程模型的解决方案。他们的方法包括对配置语言的构造等，例如 C-EPCs[18]、C-iEPCs[11]、C-WFnets[4]和 C-BPEL[7]。然而，这些语言只提供系统的部分配置解决方案，并且是与特定的语言。相比，本文的方法是基于语言无关的并且提供一个全配置的解决方案。

Van der Aalst 等人在文献[4,5]中，首先假设初始配置的过程模型是正确的，从而在确保具体过程模型以及语法和语义的基础上提出一个可配置参考过程模型。该方法需要在配置过程对每一步进行检查以确保配置模型的正确性，要求可配置的过程模型必须是健全和自由选择的 WF-net。这些要求限制了该方法的应用。在文献[1,3]中，Van der Aalst 等人采用将合作伙伴服务进行合成的方式改进了文献[4,5]中的方法，即认为配置过程是一种“外部服务”，然后将它合成为一个“最宽松的合作伙伴”。该方法是 Wolf 提出的，并且由 Wendy 用工具实现了它。用此方法在设计时可以获得所有正确的可行配置，并且避免了对每个配置过程的重复检查。但是，该方法将限制在一个无环的 Petri 网的开放的网络作为一个形式化模型，使得其配置过程是一个独立的活动。相比之下，我们的配置活动是一个可验证的配置过程。

结束语 与现有的方法相比，本文提出的一种新的自动配置业务过程模型的方法可以直接保证配置过程的正确性。本文从理论上证明了该方法，并且用实际的例子验证了该方法。在将来的工作中，我们将完善 DAT 工具，并且用它开发出基于云应用软件系统的实际配置工具。

参 考 文 献

[1] Wil. M. P. van der Aalst, et al. Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis[C]// BPM. 2010:95-111

[2] Wil. M. P. van der Aalst. Business Process Configuration in the Cloud: How to Support and Analyze Multi-tenant Process? [C]// ECOWS. Lugano, Switzerland, 2011:3-10

[3] Wil. M. P. van der Aalst, et al. Ensuring Correctness During Process Configuration via partner synthesis [J]. Information System, 2012, 37(6):574-592

[4] Wil. M. P. van der Aalst, et al. Preserving Correctness During Business Process Model Configuration [J]. Formal Aspects of Computing, 2010, 22(3):459-482

[5] Wil. M. P. van der Aalst, et al. Correctness-Preserving Configuration of Business Process Models[J]. Fundamental Approaches to Sofware Engineering Lecture Notes in Computer Science. 2008, 4961:46-61

[6] Becker J, et al. Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models[OL]. http://udoo. uni-muenster. ed/downloads/publications/1953. pdf

[7] Gottschalk F, et al. Configurable Workflow Models[J]. IJCIS, 2008, 17(2):177-221

（下转第 141 页）

网络收益与网络代价比表示虚拟网络的资源需求量与虚拟网络映射的资源消耗量比。从图 3 中可以看出，DS-VNM 算法的网络收益与网络代价比相对较大，G-VNM 算法的网络收益与网络代价比相对较小。其主要原因是 G-VNM 算法在虚拟网络映射过程中没有考虑节点间的位置关系，虚拟链路的映射路径长度较大，因此虚拟网络占用的链路带宽资源较多，虚拟网络映射的网络代价较大。虚拟网络请求接受率可以反应物理网络资源的利用率水平，从图 4 可以看出，DS-VNM 算法的虚拟网络请求接受率与 TA-VNM 算法和 G-VNM 算法相比，分别提高了 10%和 19%。其主要原因是 DS-VNM 算法在虚拟网络映射过程中充分考虑了节点间的位置关系，虚拟网络被映射到物理网络中能很好地保持原有的拓扑关系，降低了虚拟网络映射的资源消耗，从而为后续的虚拟网络请求提供了更多的网络资源量，因此 DS-VNM 算法的虚拟网络请求接受率相对较高。

结束语 在现有研究工作的基础上，文中通过建立双网搜索映射模型，同步搜索虚拟网络和物理网络中映射节点，协调映射节点以及其邻接链路，将虚拟网络中相邻的虚拟节点依次映射到物理网络中邻接的物理节点之上，使虚拟网络被映射后仍然能很好地保持原有的网络拓扑结构。仿真实验表明，DS-VNM 算法有效地降低了虚拟链路的映射路径长度和虚拟网络映射的网络代价，显著地提高了网络收益与网络代价比和虚拟网络请求接受率；文中提出的 DS-VNM 算法是可行的、有效的。

参 考 文 献

[1] Fischer A, Botero J F, Beck M T, et al. Virtual network embedding: A survey[J]. IEEE communication surveys and tutorials, 2013, 15(4): 1888-1906

[2] Cordella L P, Foggia P, Sansone C, et al. A (sub) graph isomorphism algorithm for matching large graphs[J]. IEEE Transactions on Pattern Aanalysis and Machine Intelligence, 2004, 26(4): 1367-1382

[3] Lischka J, Karl H. A virtual network mapping algorithm based on subgraph isomorphism detecting[C]// Proceeding of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures. Barcelona, Spain, 2009: 81-88

[4] 魏晓辉，邹磊，李洪亮. 基于优化的同构子图搜索的虚拟网络映射算法[J]. 吉林大学学报，2013，43(1)：165-171
Wei Xiao-hui, Zhou Lei, Li Hong-Liang. Virtual network embedding algorithm based on improved sub-graph isomorphism search[J]. Journal of Jinlin University, 2013, 43(1): 165-171

[5] Chowdhury M, Rahman M R, Boutaba R. ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping[J]. IEEE/ACM Transactions on Networking, 2012, 34(3): 206-219

[6] Yu Min-lan, Yi Yung, Jennifer R, et al. Rethinking virtual network embedding: substrate support for path splitting and migration[J]. ACM SIGCOMM Computer communication review, 2008, 38(2): 17-29

[7] Eppstein D. Finding the k shortest paths[J]. SIAM Journal on Computing, 1998, 28(2): 652-763

[8] Li Xiao-ling, Wang Huai-min, Guo Chang-guo, et al. Topology awareness algorithm for virtual network mapping[J]. Journal of Zhejiang University-Science C(Computers & Electronic), 2012, 13(3): 178-186

[9] Zegura E, Calvert K, Bhattacharjee S. How to model an internetwork[C]// Proceeding of the 15th Annual Joint Conference of the IEEE Computer and Communications Society. San Francisco CA, USA, 1996: 594-602

(上接第 133 页)

[8] Hallberbach A, Bauer T, Reichert M. Guaranteeing Soundness of Configurable Process Variants in Provop[C]// CEC. IEEE, 2009: 98-105

[9] Hallberbach A, Bauer T, Reichert M. Capturing Variability in Business Process Models: The Provop Approach[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2010, 22(6/7): 519-546

[10] Haugen K E, Husa R K, Stolen R K. STAIRS towards formal design with sequence diagrams[J]. Software and Systems Modeling, 2005, 4(4): 355-367

[11] Rosa M L, Dumas M, Hofstede A T, et al. Configurable Multi-Perspective Business Process Models[J]. Information Systems, 2011, 36(2): 313-340

[12] Li C, Reichert M, Wombacher A. Discovering Reference Models by Mining Process Variants Using a Heuristic Approach[C]// BPM. 2009: 344-362

[13] Li C, Reichert M, Wombacher A. The Minadept Clustering Approach for Discovering Reference Process Models Out of Process Variants[J]. International Journational of Cooperative Information Systems, 2010, 19(3): 159-203

[14] Schroeter J, Mucha P, Muth M, et al. Dynamic Configuration Management of Cloud-based Applications[C]// SPLC. 2012: 171-178

[15] Mietzner R, et al. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications[C]// PESOS. 2009: 18-25

[16] Reinhartz-Berger I, et al. Organizational Reference Models: Supporting an Adequate Design of Local Business Processes[J]. IJBPIM, 2009, 4(2): 134-149

[17] Reinhartz-Berger I, et al. Extending the Adaptability of Reference Models[J]. IEEE Transactions on Systems, Man and Cybernetics (Part A), 2011, 40(5): 1045-1056

[18] Rosemann M, Wil. M. P. van der Aalst. A Configurable Reference Modeling Language[J]. Information Systems, 2007, 32(1): 1-23

[19] Ruehl S T, Andelfinger U. Applying software product lines to create customizable software-as-a-service applications[C]// SPLC Workshops. 2011: 1-16

[20] Jiang Jian-min, Zhang Shi, Gong Ping, et al. Modeling and analyzing mixed communications in service-oriented trustworthy software[J]. Science China Information Science, 2012, 55(12): 2738-2756

[21] Jiang Jian-min, Zhang Shi, Gong Ping, et al. Message dependency-based adaptation of services[C]// APSCC. 2011: 442-449

[22] Jiang Jian-min, Zhang Shi, Gong Ping, et al. Configuring Business Process Models[J]. ACM SEN, Software Engineering Notes, 2013(4)