高技术通讯 2012 年 第22 卷 第3 期:326 ~ 334

doi:10.3772/j.issn.1002-0470.2012.03.017

ICEMDA 中的业务对象关联和状态管理①

冯锦丹② 战德臣 聂兰顺 徐晓飞

(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 在可互操作、可配置的企业模型驱动体系结构(ICEMDA)研究的基础上,提出了一种基于关联对象和状态矩阵的业务对象(BO)关联和状态管理模型与处理机制,以解决目前企业软件与应用研究在数据关联复杂的情况下大都忽略了由关联引发的业务状态依赖的问题。借助实例分析了业务关联的重要性,提出了管理模型的总体结构与核心概念,使用关联对象描述连接 BO 之间的直接和间接关联,以状态矩阵描述质变状态与量变状态的混合情况。基于 BO 和工作流引擎的系统架构,以相对松散的 BO 组装实现了该模型,并给出处理关联对象和状态矩阵的相关算法。通过采购管理软件的实例验证了本文方法的有效性。

关键词 业务对象(BO),关联关系,关联对象,状态依赖,状态矩阵,工作流,可互操作、可配置的企业模型驱动体系结构(ICEMDA)

0 引 言

企业应用软件可看作是由业务对象(business object,BO)和业务流程(带着表单跑流程)共同构成的系统[1]。同行业内的 BO 具有相对的稳定性,因此可以 BO 为中心,明确 BO 之间的关系,借助工作流模型将其组装起来,最终转换并生成软件系统[2]。从面向对象的角度来看,BO 应尽可能独立,但从业务需求角度来看,BO 之间存在着密切的数据关联。两个 BO 之间可存在直接或间接的关联关系。复杂关联关系的存在给基于 BO 模型的企业软件与应用开发带来了如下困难和挑战:(1)在设计 BO 的过程中不能忽略关联关系对其自身的影响[3];(2)业务语义上的紧密关联加剧了业务流程的耦合,需建立相对松散灵活的关联关系以提高企业软件与应用的适应性;(3)间接关联引发了 BO 间的状态依赖。如何定义 BO 的关联关系,如何管理由关联引发的 BO 状态变迁,使得 BO 尽可能在逻辑上独立,且在保持系统低耦合组装的前提下实现业务流程的紧密语义联系[4],目前仍是尚未完全解决的问题。已有的关联关系研究主要集中于以下两个方面:(1)面向对象的程序设计领域,研究对象之间关联的分类、模式和语言设计方法[5,6],试图降低对象或类之间的耦合程度;(2)扩展既有建模方法[7,8],从业务语义角度设计关联模型,以提高业务模型的正确性,为程序生成奠定基础。然而大多数研究者忽略了由关联关系所引发的一些问题的研究和处理。企业软件与应用领域暴露出来的基本问题是存在关联关系的两个 BO 之间必然存在状态依赖关系。本研究定位于可互操作、可配置的企业模型驱动体系结构(interoperable configurable enterprise model driven architecture, ICEMDA)[9]的平台无关模型(platform-specific model,PIM)层面,提出了一种基于关联对象和状态矩阵的 BO 关联和状态管理机制,该机制支持以低耦合的组装方式开发企业软件与应用系统。同时通过分析采购业务实例归纳出 BO 关联和状态之间的联系,进而给出 BO、关联关系、业务状态的形式化定义,以及一种协同处理 BO 关联和状态变迁的解决方案。

1 问题与实例

本节以采购业务为例,分析业务单据在其生命周期内的业务活动、关联关系和状态变迁情况。如

① 863 计划(2009AA04Z153,2008AA04Z101,2008GG1000401028)和国家自然科学基金(60773064)资助项目。

② 女,1980 年生,博士生;研究方向:模型驱动架构,软构件与复用;联系人,E-mail: Jindan.Feng@gmail.com

(收稿日期:2010-09-09)

图 1 所示,每个与采购需求单关联的单据经过业务处理以后,都要把变更反馈给其前续单据,从而引发前续单据的连锁式反应。这种反馈主要体现在前续单据的状态改变方面。

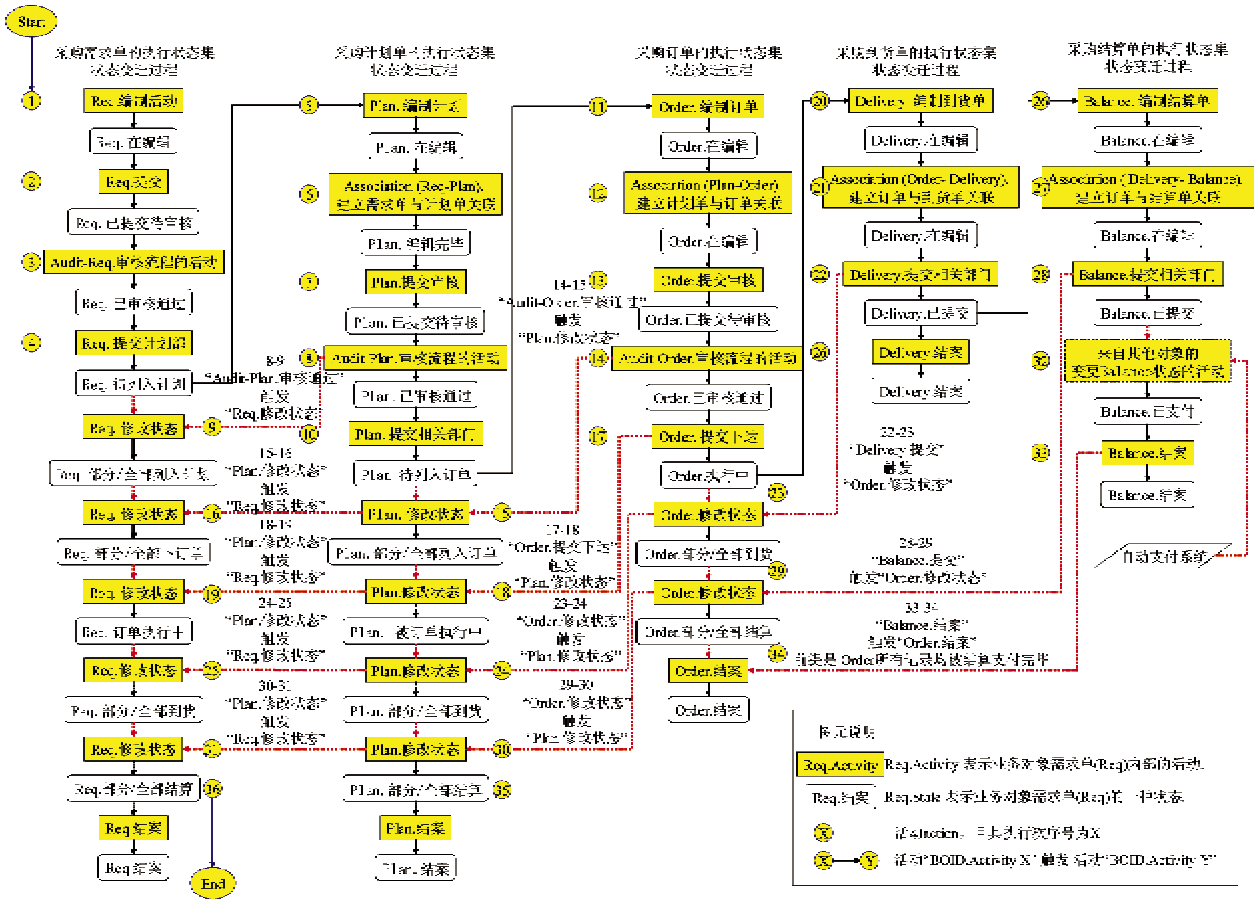

图 1　采购过程中的关联与状态变迁示例

通过分析可发现三种现象:(1)关联支持追踪溯源。在采购流程中,若干个采购需求单实例被汇总编制为采购计划,再依据任务紧急程度或物料类型等基准,将采购计划分解为多个采购订单实例。物料被采购到货以后通过上述关联路径追溯到其被用于满足上游哪项采购需求。因此关联关系支持了业务单据之间的双向追溯。(2)关联具有多样性和可变性。同一领域的企业可具有不同的业务流程。如从提出采购需求到编制采购订单就可遵循以下三种流程:(a)采购需求单汇总后进入编制采购计划环节,依据计划单、比质比价、选择合适的供应商,签订采购订单;(b)自采购需求单直接生成询价单,再签订采购订单;(c)自采购需求直接生成采购订单。(3)关联导致业务状态依赖。前续单据的状态受到与其直接或间接关联的单据的影响。这种影响导致具有关联关系的业务单据之间存在状态依赖性。图 1中 action17 使采购订单实例的状态变为“执行中”,而通过关联路径的反馈使得对应的采购需求单实例在 action19 时变更为“订单执行中”。

由此可知,关联关系的管理会直接影响 BO 设计的良构性和可复用性(依据文献[9],将业务单据与其上的活动统一抽象为 BO 以进行 PIM 建模);错误的关联关系会导致 BO 之间错误的组装关系;关联关系协同不好,将导致 BO 的生命周期状态发生混乱。因此,在建立 BO 和业务流程平台无关模型阶段,有必要寻找一种正确建立关联关系的快捷方法,以支持 BO 的状态正确变迁,以及 BO 之间的正确组装;并且这种方法使得 BO 在逻辑上相对独立,在保持系统低耦合组装的前提下体现业务流程的语义联系,在软件应用过程中仍可被复用。

2　BO 的关联和状态管理模型

BO 的关联和状态管理的总体模型如图 2 所示。

2.1　BO

在ICEMDA-PIM层,将计算无关模型中的信息

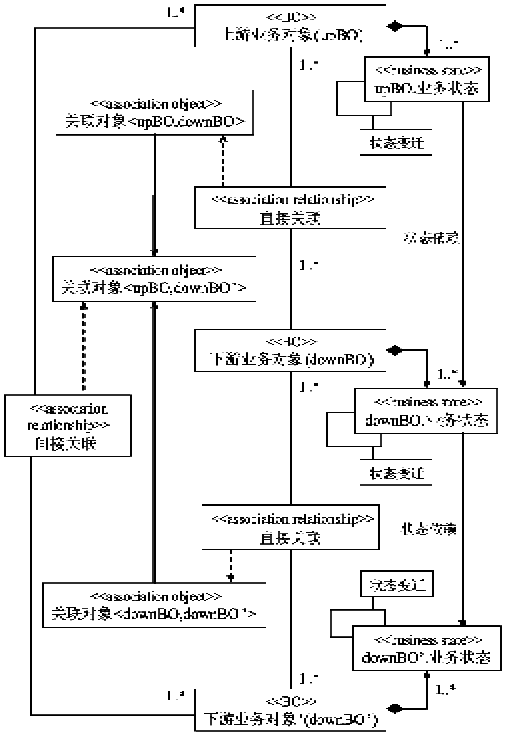

图 2 BO 的关联和状态管理总体模型

实体和围绕信息实体进行的业务活动统一抽象和规范化为一个个 BO。BO 是业务领域内有一定业务功能、相对独立的客观事物的大粒度抽象表达,是业务过程中相对集中的业务数据和业务活动中心。

定义 1(BO) 业务对象可表示为 $BO = (ID, DS, AT, OP, DPR, SS, ST)$,其中 ID 是 BO 的唯一标识;$DS = \{ds \mid ds = (name, \{d\})\}$,$ds$ 是依据数据项 d 之间的语义关系组成的数据集 $name$;AT 是 BO 的活动集,每个活动 at 都是 BO 中若干原子操作 op 的封装结果;$OP = \{op \mid op$ 是 BO 的操作$\}$;$DPR = \{(op, ds)\}$,表示操作对数据集的控制关系集;$SS = \{s \mid s$ 是 BO 的一种状态且 $(\exists s)(s \rightarrow op_1 \wedge \cdots \wedge op_n)\}$;$ST = \{st \mid s_o = st(s_i, at), at \in AT, s_i, s_o \in SS\}$ 表示 BO 的状态变迁关系集,at 可触发 BO 的状态从 s_i 转换到 s_o。

$\forall A, B \in BO$,A 和 B 隶属于同一个业务过程 P,A 的生命周期开始时间早于 B,则 A 称为上游业务对象($upBO$),B 被称为下游业务对象($downBO$)。A 与 B 的上下游关系可表示为 $A \leqslant B$,称为 A 是 B 的上游业务对象。$upBO$ 和 $downBO$ 的关系是相对而言的,当 A 和 B 的业务执行顺序发生变化时,其间的上下游关系也会随之变化。

2.2 关联关系

在上/下游 BO 之间存在两种类型的关联关系:直接关联是两个 BO 在同一业务流程路径上且直接相邻,具有满足关联条件约束的关联对象。间接关联是指在业务流程路径并不相邻的两个 BO,通过若干个与其具有直接关联的 BO 或关联对象仍可有互联系的关系。如采购需求单与采购计划之间存在直接关联,而与采购订单之间存在间接关联。在业务流程和关联关系的基础上,若干 BO 之间形成了一系列的关联路径。下游 BO 的变化都在不同程度上影响着上游 BO。

为两个 BO 建立关联关系的核心步骤是定义关联对象和设置关联条件。关联对象是维护两个 BO 的关联数据的特殊业务对象。关联条件是建立关联对象的前提和约束。因此两个 BO 的间接关联可建立在两个直接关联的关联对象上。

定义 2(关联关系) $\forall A, B \in BO$ 且 $A \leqslant B$,A 和 B 的关联关系是一个二元关联,是指 A 和 B 的特定数据集之间的联系,记为 $f(A) \Rightarrow B$。若 $\exists f(A) \Rightarrow B$ 当且仅当对 $\forall \alpha \in Instance(A.ds)$,至少存在一个 $\beta \in Instance(B.ds)$;对 $\exists \beta \in Instance(B.ds)$,至少存在一个 $\alpha \in Instance(A.ds)$。其中 $Instance(BO.DS)$ 表示 BO 数据集所有实例的集合,$BO.ds \subseteq BO$。DS 表示特定数据集。

定义 3(关联对象) 关联对象($AssociationBO$)是一种扩展 BO,可表示为 ABO($ABOID$, $ARDS$, $AOPS$, AFS, ASS),其中 $ABOID$ 是 ABO 的唯一标识;$ARDS$ 是由上下游 BO 的关联键 $AKey(upBO.key, downBO.key)$,关联属性 $AAttribute$ ($upBO.ds$, $downBO.ds$) 和关联项 $Item$ 构成的数据集合,$ARDS = (AKey, AAttribute, Item)$;关联操作集 $AOPS = \{AQuery, ACreate, AInsert, AUpdate, ADelete, AOperate\}$,包含关联的查询、新建、增加、修改、删除和确认生效操作;$AFS = \{$编辑 ABO,查询 ABO,生效 $ABO\}$;$ASS = \{(S, T) \mid S$ 是 ABO 的状态集,T 是 S 的状态变迁关系集$\}$,$S = \{Occupied, Operative\}$,其中 $Occupied$ 是关联占用态($O1$),表示当前 $upBO$ 和 $downBO$ 初步建立了关联,二者的关联属性实例资源被暂时占用,不能被其他关联实例征用。$Operative$ 是关联有效态($O2$),表示此 $f(upBO)$($downBO$ 正式生效并可参与后续业务活动,且二者的关联属性实例资源被永久占用;ABO 在不同状态下具有不同的操作集,$UsableOP(O1) = \{AQuery, AInsert, AUpdate, ADelete, AOperate\}$,$UsableOP(O2) = \{AQuery\}$,$T(BO) = \{t(\mid t$ 表示从前续状态 S_i 通过一系列操作 op 后到达后续状态 $S_j\}$,则 $T(ABO) =$

$\{O1 \xrightarrow{AOPS\{AOperate\}} O2,\ O1 \xrightarrow{\{AOperate\}} O2,\ O1 \xrightarrow{\{AQuery\}} O2\}$。

2.3 业务状态

定义 4(业务状态) $\forall A \in BO$,业务状态是 $Instance(A)$ 在其生命周期内某个阶段内所处的状况和所具有的属性,记为 $State = (Name, SetName, Num, Condition, ActionSet, Type)$,其中,$Name$ 描述了一种状态的名称;Num 描述该状态在此周期内的执行顺序号。以 $SetName$ 为名称的状态集 $StateSet(A) = (SetName, \{State_{Num}\})$,按照 Num 标识的前后关系可被排成一个状态序列;$Condition$ 定义了允许进入当前状态的各项条件;$ActionSet$ 表示 $Name$ 状态下可用的业务活动集;$Type(\{Start, InnerState, OuerState, End\}$ 是 BO 的状态类型。起始状态 $Start$ 即初始状态;内部状态 $InnerState$ 是指那些由 A 自身引发所能到达的状态。对于与 A 关联的其他 BO,通过关联触发 A 所到达的状态,被称为 A 的关联状态或外部状态 $OuerState$。End 是指异常终端或生命周期结束的终止状态。

定义 5(状态变迁) $\forall A \in BO$,状态变迁是指在某个状态集 $StateSet(A)$ 中从前续状态 $State_{(t)}$ 到后续状态 $State_{(t+1)}$ 的转移。

将 BO 生命周期划分为 l 个时间区间,$1 \leqslant l \leqslant N$,首个区间的状态为 s_s,区间 N 的状态为 s_e。每个时间区间内的状态可能具有不同或相似的性质。

性质 1 $\forall BO,\ StateSet_i \subset BO,\ s_{im}, s_{in} \in StateSet_i$,$s_{im}$,$s_{in}$ 在 $1 < l < N$ 区间上,若 s_{im} 与任意 s_{in} 都有 $s_{im} \cap s_{in} = \varnothing$,则 s_{im} 和 s_{in} 不相等,其中 s_{im} 可称为质变状态。

性质 2 质变状态 s_{im} 在 $(t_{m-1}, t_m]$ 内划分 k 个时间区间,每个时间段内的状态为 $s_{im}(j)$,$s_{im}(j) \approx s_{im}(j+1)$,$s_{im}(j) \cap s_{im}(j+1) = s_{im}$ 且 $|s_{im}(j)| < |s_{im}(j+1)|$,$0 \leqslant j < k$,则 $s_{im}(j)$ 和 $s_{im}(j+1)$ 相似,其中 $s_{im}(j)$ 可称为量变状态。

为了适应各种管理精细程度的需要,将 BO 状态划分为粗、细两个级别。粗放级别是只关注到状态的质变级别;精细级别是关注到状态的量变级别。BO 在一个时段内至多处于一种质变状态,如“在编辑”和“已下订单”之间不存在任何共同点。量变状态 $s_{im}(j)$ 体现了 s_{im} 在数量或程度上逐渐增加的迁移过程。量变状态也具有持续性且呈现出逐步攀升的势态。如图 1 所述,一个 $Instance(BO)$ 在业务流程中可能经历若干次拆分与合并处理,在某一时刻其可具有多个状态集的若干种量变状态(即复合状态),例如图 3 的 $t4$ 时刻所示。

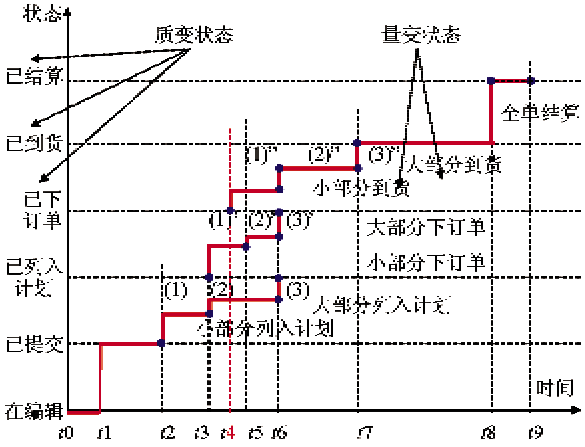

图 3 采购需求单状态的变迁路线示意图

对于同一企业软件与应用而言,管理 BO 的精细程度总是一致的,则量变状态的划分标准也是一致的。在每个质变状态的时间区间内,划分量变状态的数量规模基本相当。因此可采用状态矩阵的形式表达 BO 的一个状态集。

定义 6(状态矩阵) 业务对象的状态矩阵 $\boldsymbol{M}_{SetName} = (a_{ij})_{mn}$,如矩阵

$$\boldsymbol{M} = \begin{bmatrix} S_{11} & a_{12} & \cdots & a_{1n} \\ S_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ S_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

所示,a_{ij} 为状态矩阵中第 i 行第 j 列上的元素,即业务对象的 $SetName$ 状态集内的一个状态为 a_{ij}。

BO 的状态矩阵 $\boldsymbol{M}$ 具有如下性质:(1)当 $j = 1$ 时,a_{i1} 表示 BO 的一种质变状态,记为 S_{i1} 以区分量变状态;下标 i 表示质变状态的序号,则质变状态的序列关系为 $S_{11} < S_{21} < \cdots < S_{m1}$。(2)当 $j \neq 1$ 时,a_{ij} 表示量变状态,j 表示该行量变状态的序号,序列关系为 $a_{12} < a_{13} < \cdots < a_{1n}$。(3)当 $a_{ij} = 0$ 时,表示 BO 不具有这种状态。(4)当 $a_{ij} \neq 0$ 时,表示 BO 这个状态集中包含当前状态。(5)BO 在此状态集中的状态序列为 $S_{11} < a_{12} < \cdots < a_{1n} < S_{21} < a_{22} < \cdots < a_{2n} < \cdots < S_{m1} < a_{m2} < \cdots < a_{mn}$,BO 在其生命周期内遵循此状态变迁次序。(6)$\boldsymbol{M}$ 的最后一行必满足 $S_{m1} \neq 0$,$a_{mj} = 0$,$j = 2, \cdots, n$,表明 BO 的最终状态是质变状态。

定义 7(业务状态依赖) 两个 BO 实例的状态之间存在状态依赖关系,当且仅当 $\forall A, B \in BO$,A 经过关联路径可达到 B,$\forall \alpha \in Instance(A)$,$\forall \beta \in Instance(B)$,$X \in \alpha.State$,$Y \in \beta.State$,对于每个 Y 值都能确定至少一种 X 值,记为 $X \rightarrow Y$。

定理 1 当 $\forall A,B \in BO, \forall \alpha \in Instance(A), \forall \beta \in Instance(B), X \in \alpha.State, Y \in \beta.State$，若 A 和 B 存在关联关系，则 A 和 B 之间必存在 $X \rightarrow Y$。

证明：当 $\exists f(A) \Rightarrow B$，则 $A.ds$ 和 $B.ds$ 存在对应关系。$A.ds$ 和 $B.ds$ 的实例值决定着 A 和 B 在该关联路径上处于何种 s_{im} 和 $s_{im}(j)$ 状态。$B.ds$ 的实例值变化使得 Y 随之变化。$B.ds$ 的实例值变化可影响 A 中与其关联的那部分 ds 的实例值变化，因此 $A.ds$ 的实例值变化将导致 A 的状态变化。所以，当 A 和 B 之间存在关联关系时必然也存在状态依赖关系。

3 基于 BO 关联和状态模型的管理机制

3.1 机制概述

BO 关联与状态模型采用了基于业务对象和工作流引擎的系统架构予以实现，如图 4 所示。

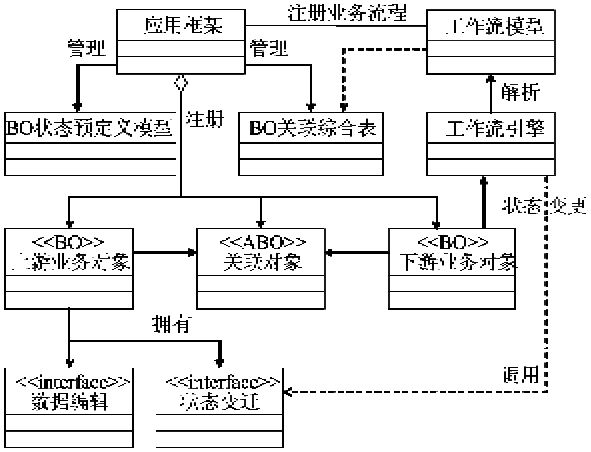

图 4 基于 BO 和工作流引擎的应用系统架构

BO 借助工作流模型描述其间的协作关系和执行顺序，由工作流引擎调度 BO 并将其组装成最终的企业应用系统。应用框架负责注册与管理系统内部的全部业务对象。基于 BO 高内聚低耦合的设计思路，由应用框架负责管理 BO 状态预定义模型和关联综合表。状态预定义模型用于描述每个 BO 的状态集与状态的次序和变迁条件。关联综合表是 *upBO* 与 *downBO* 的关联统计数据存储表。工作流引擎负责状态迁移条件、状态修改指令等相关消息的传递，由业务对象自身的状态控制方法实现其状态改变。

接下来介绍基于关联对象和状态矩阵的 BO 关联和状态处理的核心步骤。

设 $\forall A,B,C \in BO$，BO 可以具有多套状态集，则对 BO 的所有状态集分别执行步骤 1。

步骤 1：状态为 $S_{(t)}$ 的 A 实例，依据状态预定义模型可知后续计划到达 $S_{(t+1)}$ 态。若 $S_{(t+1)} \in InnerState$，处于 $S_{(t)}$ 态的 A 在执行业务活动或属性变更后，若满足预订迁移条件，提交 $S_{(t+1)}$ 到步骤 4，同时执行步骤 2。若 $S_{(t+1)} \in OuterState$，则保持 $S(t)$ 态直到存在外部关系满足 $S_{(t+1)}$ 的变迁条件后提交 $S_{(t+1)}$ 到步骤 4；

步骤 2：依据工作流模型定义的 BO 交互关系获取与 A 实例关联（含间接关联）的全体 ABO 实例集和上/下游 BO 实例。依次对各 ABO 实例执行步骤 3。

步骤 3：用关联对象数据统计算法计算 ABO 实例的数据，将计算结果与状态预定义模型进行对比，判定相应上下游 BO 实例的后续状态值。转入步骤 4，并输入如下参数：上下游 BO 实例，当前状态值，后续状态值。

步骤 4：维护各 BO 实例的状态矩阵，执行基于状态矩阵的状态变迁算法，根据输入的参数，从可能的复合状态中获取后续时刻的新矩阵，输出最终状态值。

步骤 5：由工作流引擎将步骤 4 输出的消息传递给每个 BO 的状态控制方法以实现状态修改。

3.2 相关算法

3.2.1 关联对象数据统计算法

算法基本思想是借助当前业务对象 C 与直接关联的前续 BO 之间的数据满足情况，向前递归计算出 C 对关联流程中所有前续 BO 的数据满足情况。

算法 1 关联对象数据统计算法

输入：当前 C，与 C 有关的业务流程 BP。

输出：C 所在业务流程上所有关联对象的下游业务对象关联综合表。

（1）令关联路径集 $ARBPSet = \varnothing$；

（2）依据 BP 选取以当前 C 为终点的前续关联路径集 $ARBPSet$；

（3）$\forall ARBP \in ARBPSet$，令当前存放关联对象的堆栈 $StackBefore$ 和 $StackBack$ 为空；

（4）$\exists A,B \in BO, A \lessdot B, A,B$ 属于 $ARBP$，按照 $ARBP$ 执行正序将其上所有关联对象 $ABO(A-B)$ 入堆栈 $StackBefore$，令 $X = C$；

（5）当 $X = StackBefore$ 栈顶关联对象 $ABO(A-B)_i$ 的后续对象 B，与 $Instance(B)$ 关联的所有数据 $Instance(A)$ 的关联键 $Instance(A.key)$ 放入集合 $ASet$；

(6) $\forall Instance(A.key) \in ASet$, 当 $Instance(A.key) \neq \varnothing$, 统计 $Instance(A.key)$ 在 $ABO(A-B)_i$ 中所有被生效的关联值之和 $Sum(Instance(Item^{ABO(A-B)i}))$, 并将结果写入 A 的 *downBO* 关联综合表 $Statistics(A-downBOs)$; 若 A 实例中关联键的关联属性数值存在被合并或拆分情况时,可依据预定义关联规则自动填写数据或人工手动调整;

(7) 执行(6)至 $ASet = \varnothing$, 从 *StackBefore* 将 $ABO(A-B)_i$ 弹出压栈入 *StackBack*, 若 *getlength* (*StackBack*) > 1, 则执行(8)。令 $X = ABO(A-B)_i$ 的前续业务对象 A,循环执行(5)直至 *StackBefore* 为空;

(8) 获取 *StackBack* 栈顶元素 $ABO(A-B)_j$ 的后续对象 $Instance(B_j)$, 通过查询 $Instance(B_j)$ 与栈底元素 *StackBack*[0] 后续对象 X 的下游对象关联综合表,建立前续对象 A_j 与 X 间接关联数据并写入 A_j 的 *downBO* 关联综合表 $Statistics(A_j \rightarrow downBOs)$, 写入过程中也需依据一定的关联规则;

在步骤(6)和(8)中,下游业务对象关联综合表 $Statistics(A(downBOs)$ 是由 *BO* 关联键、被关联属性、下游业务对象类型、下游业务对象关联键和属性、关联总值共同构成二维表,反映了 A 与直接或间接关联对象的具体满足情况。

3.2.2 基于状态矩阵的状态变迁算法

由算法 1 的关联数据结果与状态定义进行比对,获取当前的新状态集。在上一时刻的 *BO* 状态矩阵中添加新状态,统一判定当前时刻的状态矩阵取值,得到 BO 最终的状态矩阵。

算法 2 基于状态矩阵的状态变迁算法

输入: *BO* 在 t 时刻的实时状态矩阵,在 $t+1$ 时刻被触发的新状态集 $X_{(t+1)}$。

输出: *BO* 在 $t+1$ 时刻的实时状态矩阵 $\boldsymbol{M}$。

(1) 构造当前 t 时刻的实时状态矩阵 $\boldsymbol{M}_{(t)}$;

(2) 添加新 X 到 $\boldsymbol{M}$,依据前述构造方法;

(3) 逐行扫描行向量,当 $S_{i1}(i = 1,\cdots,m) \neq 0$ 时,将 $i-1$ 行整行删除,直到所有行扫描结束;

(4) 逐行扫描行向量,当第 i 行存在 $a_{ij} \neq 0(j > 1)$ 时,取该行中 j 最大位置的量变状态作为 $\boldsymbol{M}(t+1)$ 的一个元素,将该行其他 a 元素删除;

(5) 矩阵中所有剩余的非零元素就是 $\boldsymbol{M}_{(t+1)}$ 的元素。

3.2.3 算法性能分析

基于关联的业务对象状态处理过程中,算法都具有多项式时间复杂度,其中算法 1 具有最大的时间复杂度。若一个业务过程中存在 $n+1$ 个 *BO*, n 个 *ABO*, 每个 *BO* 实例与后续 *BO* 之间存在 m_i 个关联元组,则关联对象数据统计算法的复杂度为 $O(m_i \times n^2)$。该算法的执行效率与业务过程中存在的关联对象数目和每个关联对象的有效关联元组数目有关。在最坏的情况下,业务过程中末尾的 BO 触发了状态变迁过程,则每个关联路上的关联对象的数据都要被重新统计,时间复杂度将以指数增加。该算法的执行频率与 BO 状态管理的粗细级别有关。状态管理的级别越细,该算法被执行的频率越高。在最坏的情况下,BO 的每次状态变迁都启动该算法统计前续所有 ABO。

4 实例研究

针对图 1 的采购实例,采购需求单 (*Req*) 的执行状态集为 $ExecuteState = \{S_{11}, S_{21}, a_{22}, a_{23}, S_{31}, a_{32}, S_{41}, a_{42}, a_{43}, S_{51}, S_{61}, S_{71}\}$, S_{i1} 依次表示 5 种质变状态,即 S_{11}: 在编辑, S_{21}: 已提交, S_{31}: 已全部列入计划, S_{41}: 已全部下订单, S_{51}: 已全部到货, S_{61}: 已全部结算, S_{71}: 结案。a 表示 5 种量变状态, a_{22}: 小于总额 1/2 列入计划, a_{23}: 大于等于 1/2 列入计划, a_{32}: 部分下订单, a_{42}: 小于总额1/2到货, a_{43}: 大于等于总额 1/2 到货。根据本文模型建立 *Req* 与后续单据的关联对象(见表 1 - 表 4)。

表 1 需求单与计划单的直接关联对象

需求单号	需求项号	计划单号	计划项号	物料编码	需求量	关联项	计划量	状态
REQ11001	R1	PLAN11001	P1	M1	100	100	200	已生效
REQ11001	R2	PLAN11001	P2	M2	150	100	100	已生效
REQ11001	R2	PLAN15002	2	M2	150	50	50	已生效
REQ11001	R3	PLAN15002	1	M3	200	200	200	已生效
REQ12001	1	PLAN11001	P1	M1	100	100	200	已生效
REQ12001	2	PLAN11001	P3	M4	200	150	150	已生效

表 2 计划单与订单的直接关联对象

计划单号	订单项号	到货单	到货项号	物料编码	订货数量	关联项	到货数量	状态
PLAN11001	P1	ORD12001	1	M1	200	200	200	已生效
PLAN11001	P2	ORD12002	1	M2	100	100	100	已生效
PLAN11001	P3	ORD12003	1	M4	150	150	150	已生效
PLAN15002	1	ORD16001	1	M3	200	200	200	已生效
PLAN15002	2	ORD16002	1	M2	50	50	200	已生效

表 3 订单与到货单的直接关联对象

订单号	订单项号	到货单	到货项号	物料编码	订货数量	关联项	到货数量	状态
ORD12001	1	A1	1	M1	200	100	100	已生效
ORD12001	1	A2	2	M1	200	100	100	已生效
ORD12002	1	B	1	M2	100	100	100	已生效
ORD16001	1	C	1	M3	200	200	250	已生效
ORD16002	1	C	2	M2	50	50	250	已生效
ORD12003	1	D1	1	M4	150	50	50	已生效
ORD12003	1	D2	1	M4	150	30	30	拟定
ORD12003	1	D3	1	M4	150	20	20	拟定

表 4 需求单与到货单的间接关联对象

需求单号	需求项号	物料编码	需求数量	到货单	到货项号	间接关联项	到货量	状态
REQ11001	R1	M1	100	A1	1	100	100	已生效
REQ11001	R2	M2	150	B	1	100	100	已生效
				C	2	50	250	已生效
REQ11001	R3	M3	200	C	1	200	250	已生效
REQ12001	1	M1	100	A2	2	100	100	已生效
REQ12001	2	M4	200	D1	1	50	50	已生效
				D2	1	30	30	拟定
				D3	1	20	20	拟定

到货单实例 *A*1，*A*2，*B*，*C*，*D*1，*D*2，*D*3 按照名称先后次序被 action22 提交则通过算法 1 可以统计获取 *Req* 的关联综合表，如表 5 所示。当到货单实例 *D*1 状态变更为”已提交”，通过对前续单据的关联统计和状态变更判定以后得到，与其关联的订单实例 *ORD*12003 的状态矩阵由“已下达”变为“小部分到货”，计划单实例 *PLAN*11001 仍保持“大部分到货”状态不变，则 *Req* 实例 *REQ*12001 的状态矩阵由 $\{a_{23},a_{32},a_{42}\}$ 变迁为 $\{a_{23},a_{32},a_{43}\}$，即大部分需求已被列入计划并下达了订单，并有一半产品已经到货。

表 5 需求的关联综合表

时刻：到货单实例 X 执行 action22 后			*X* = *A*2	*X* = *B*	*X* = *D*1
需求单号	需求数量	被关联的对象名称	关联总量	→关联总量	→关联总量
REQ11001	450	计划单	450	450	450
		订单	450	450	450
		到货单	100	200	450
REQ12001	300	计划单	250	250	250
		订单	250	250	250
		到货单	100	100	150

因关联关系本身非常复杂,出现多重间接关联关系之后,数据的满足关系仍存在不确定性。例如,当采购需求实例 *REQ*11001 和 *REQ*12001 各有 100 个物料 *M*1 的采购需求,它们被合并为计划 *PLAN*11001 的 *P*1 项和采购订单 *ORD*12001。但在实际到货时只有 100 个 *M*1 被第一批次收到,此时那个 *Req* 被 *A*1 满足的具体关系存在不确定性。可采用"确定化规则"为 *Req* 分配合适的关联数量,例如按照需求先后次序进行满足,采购紧急程度优先满足、最小量优先满足、项目利润最大者优先满足、平均分配等原则来支持自动化的关联关系实现。或采用人工手动调整的方案以灵活满足企业实际需求。实例中采用了按照需求先后次序进行满足的方式,将 *A*1 批次到货的物料分配给 *REQ*11001 的 *R*1 项。

通过实例验证,对比文献[10]和[11]可发现这二者讨论的基础都是小粒度对象之间的直接关联关系,并不适用于 ICEMDA 架构中大粒度业务对象的复杂关联关系管理,尤其欠缺对间接关联的自动统计功能,本文方法更有利于业务流程需求的实现。文献[12]仅给出了一种对象关联模型的笼统介绍,并未明确指出对象关联模型的具体实现方法和解决方案,如对象关联模型被软构件化之后,怎样与参与者对象交互,如何通过关联数据支持状态变迁的具体细节。本文将处理机制、算法以及案例相结合给出一种关联和状态兼顾的模式。这种关联和状态的管理方法可作为一种模式在企业软件与应用设计中被复用,因此可以更明确地指导应用。

5 结 论

本文以模型驱动体系结构为研究背景,分析了 BO 之间关联关系与业务状态依赖的实例,指出这种复杂的关联和状态变迁问题给企业应用软件的高效开发带来了诸多阻碍。提出了业务对象的关联和状态管理模型以及相关处理机制和算法,经过工程中的实践和应用,证明本文方法较传统的关联表和状态日志方法能更为有效地管理大粒度业务对象之间的关联(尤其是间接关联)和状态自动变迁。通过业务对象和关联对象之间相对的低耦合组装满足业务流程中的语义关联性和状态依赖性,对指导基于业务对象的模型驱动软件开发具有较高的应用价值。BO 之间还存在的复杂运算和集成关系等协作关系,将作为下一步工作予以深入研究,以提高业务对象的相对独立性。

参考文献

[1] Caetano A, Silva A, Tribolet J. Using roles and business objects to model and understand business processes. In: Proceedings of the ACM Symposium on Applied Computing, Santa Fe, USA, 2005. 1308-1313

[2] Fingar P. Component-based frameworks for e-commerce. *Communications of the ACM*, 2000, 43(10):61-67

[3] Di Lucca G, Fasolino A, Tramontana P, et al. Recovering a business object model from web applications. In: Proceedings of the 27th Annual International Conference on Computer Software and Applications, Dallas, USA, 2003. 348-353

[4] Sutherland J, Heuvel W. Enterprise application integration encounters complex adaptive systems: a business object perspective. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences, Hawaii, USA, 2002. 3724-3733

[5] Boyd L. Business patterns of association objects. In: Pattern Languages of Program Design. Boston, USA: Addison Wesley Publisher, 1998. 395-408

[6] Kristensen B. Complex associations: abstractions in object-oriented modeling. *ACM SIGPLAN Notices*, 1994, 29(10): 272-286

[7] Saiedian H. An evaluation of extended entity-relationship model. *Information and Software Technology*, 1997, 39(7): 449-462

[8] Krishnamurthy S, Lam H. An object-oriented semantic association model (OSAM*). *Artificial Intelligence: in industrial engineering and manufacturing: theoretical issues and applications*, 1988: 463-494

[9] 战德臣, 冯锦丹, 聂兰顺等. 一种可互操作可配置可执行的模型驱动体系结构. 电子学报, 2008, 36(12A): 120-127

[10] Tsagias M, Kitchenham B. An evaluation of the business object approach to software development. *Journal of Systems and Software*, 2000, 52(2-3):149-156

[11] Jones T, Song II-Y. Analysis of binary/ternary cardinality combinations in entity-relationship modeling. *Data & Knowledge Engineering*, 1996, 19(1):39-64

[12] Gessenharter D. Implementing UML associations in Java: a slim code pattern for a complex modeling concept. In: Proceedings of the workshop on relationships and associations in object-oriented languages, Genova, Italy, 2009. 17-24

The management of associations and states of business objects in ICEMDA

Feng Jindan, Zhan Dechen, Nie Lanshun, Xu Xiaofei
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract

The paper proposes a model and mechanism for management of the associations and states of business objects on the basis of the research on the interoperable configurable enterprise model driven architecture (ICEMDA) to solve the problem of business state dependency aroused by the associations, which is usually ignored in the studies of enterprise software and applications. It is presented through a practical case that the associations affect business correctness. And the architecture and primary concepts are given, including the association object for connecting business objects directly or indirectly, and the state matrix for describing the compositive states of qualitative changes and quantitative changes. The framework based on business object and workflow engine is designed to implement the model according to the relative looser assembly pattern, and the algorithms for association objects and state matrices are also presented and used. A case of purchase management system is finally shown to validate the mechanism.

Key words: business object (BD), association relationship, association object, state dependency, state matrix, workflow, interoperable configurable enterprise model driven architecture (ICEMDA)