ИНФОРМАТИКА

2004 **апрель-июнь** **№2**

УДК З 973.2 Э949

В.В. Леонович

ЭФФЕКТИВНЫЕ ТРАНЗАКЦИОННЫЕ МОДЕЛИ В СИСТЕМАХ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМИ ПРОЦЕССАМИ

Рассматривается проблема транзакционной обработки данных в системах управления технологическими процессами. Дан обзор подходов и технологий к решению этой проблемы. Приведены общая терминология описания и способ анализа технологических процессов, позволяющие улучшить процесс разработки сложных распределенных программных комплексов.

Введение

Системы управления технологическим процессом (WorkFlow Management System (WFMS)) [1] имеют широкое применение в области автоматизации как разновидность систем массового обслуживания. Это системы электронного бизнеса в целом, банковские системы в частности, медицинские и военные программные комплексы, системы управления предприятием.

Одной из главных задач при конструировании WFMS как многопользовательских систем является обеспечение согласованности данных, их целостности и актуальности, корректности вычислений и высокой производительности при параллельной обработке, а также надежности в сложных, разнотипных и распределенных программных средах. На уровне доступа к данным эта задача решается в системах управления базами данных (СУБД) [2, 3], а эффективные транзакционные модели (Advanced Transaction Model (ATM)) [4-6] рассматривают ее на уровне сложных «долгоживущих» распределенных операций (технологических процессов) в WFMS, в которых могут участвовать несколько СУБД, доступных в разное время.

Обобщая понятие транзакции в СУБД на WFMS, ATM оптимизируют процесс выполнения конкурентных операций в WFMS, сохраняя корректность вычислений в рамках некоторого критерия, зависящего от требований задачи. Таким образом, принципы ATM могут результативно использоваться для построения надежных транзакционных технологических процессов (Transactional WorkFlow (TWF)), что является неотъемлемой частью архитектуры WFMS.

Данная работа систематизирует модели транзакционной обработки информации и представляет общую терминологию описания и способ анализа технологических процессов.

1. Описание транзакционных технологических процессов

Обычно *транзакцию* определяют как единицу работы, обладающую свойствами ACID [2, 3, 7]: Atomicity (атомарность), Consistency (согласованность), Isolation (изоляция), Durability (долговечность), т. е. транзакция или выполнена полностью, или не выполнена совсем – *атомарность*; трансформирует данные из одного целостного состояния в другое – *согласованность*; различные параллельно исполняемые транзакции не оказывают влияния друг на друга – *изоляция*; если транзакция фиксирует результаты своего выполнения, то они перманентны, – *долговечность*.

Аналогичным образом транзакционный технологический процесс TWF можно рассматривать как некоторую бизнес-операцию, требующую наличия свойств ACID в полной мере или в некоторой степени. Технологический процесс (WorkFlow (WF)), состоящий из шагов-транзакций, в общем случае транзакционным в целом не является.

Полагая аксиоматически то, что любые последовательно выполняемые действия системы для одного пользователя корректны, естественным образом в качестве критерия корректности выполнения конкурентных транзакций получим критерий упорядочиваемости:

Сценарий (история) выполнения конкурентных транзакций корректен и упорядочиваем тогда и только тогда, когда конечное состояние данных и результат выполнения этого сценария эквивалентны таковым при выполнении этих транзакций последовательно.

Отметим, что любой сценарий из конкурентных ACID транзакций корректен и упорядочиваем по определению, однако проблема низкой степени параллелизма «долгоживущих» ACID транзакций заставляет искать компромиссные подходы. АТМ, по сути, ослабляют свойства ACID и тем самым повышают производительность, сохраняя при этом корректность полученных результатов. Итак, модели транзакционной обработки информации включают в себя принципы и алгоритмы представления и эффективного выполнения транзакционных технологических процессов в многопользовательском режиме.

Введем следующие обозначения:

A – множество всех действий, определенных в системе;

a_i – некоторое действие (action), $a_i \chi A$;

P – множество параллельных процессов (конкурентных пользовательских сессий), $P = \{ p_m \}$, где процесс p_m может быть как *автономным*, т. е. работающим без участия пользователя, так и *интерактивным*, т. е. содержать в себе диалоги с пользователем;

$a_{i(j)}^{m}$ – j-й экземпляр действия $a_i \chi A$, выполняемый в некотором процессе $p_m \chi P$;

$A(p_m)$ – множество действий процесса p_m, $A(p_m) = \{ a_{i(j)}^{m} \}$, $p_m \chi P$;

$A(P)$ – множество действий всех процессов из P, $A(P) = 4\, A(p_m)$, $p_m \chi P$.

Все действия выполняются в рамках заданных технологических процессов некоторой рассматриваемой прикладной задачи. Обозначим:

$W = \{w_k\}$ – множество всех технологических процессов w_k рассматриваемой задачи;

$W(P)$ – множество выполняемых WF для всех процессов из P;

$A(w_k)$ – множество действий по выполнению w_k, которое в общем случае зависит от представления w_k в той или иной модели. Под *представлением* технологического процесса будем понимать частный способ его реализации.

Технологический процесс WF не обязан быть атомарным. Группы действий в представлении WF, образующие атомарную единицу работы, составляют транзакцию. Эти группы могут включаться друг в друга, но не перекрываться. Обозначим:

$T = \{t_k\}$ – множество всех транзакций t_k рассматриваемой задачи (с учетом представлений $w_k \chi W$ в той или иной модели);

$T(P)$ – множество выполняемых транзакций для всех процессов из P;

$A(t_k)$ – множество действий по выполнению транзакции t_k.

Таким образом, если весь технологический процесс $w \chi W$ должен быть транзакционным, то все его действия должны образовывать транзакцию, т. е. $\exists\, t \chi T$: $A(w) = A(t)$.

По определению процесс p_m может содержать в себе несколько транзакций, равно как и WF w_k может распределяться на несколько процессов. Поэтому такой подход позволяет описывать *бесконечные* (endless), *долгоживущие* (long-lived), *интерактивные* (interactive/collaborative) процессы в контексте расширенных транзакций [0], а также многопоточные процессы в контексте распределенных транзакций [0], причем как с синхронной, так и с асинхронной коммуникациями.

На рис. 1, *а* приведен пример модели технологического процесса.

Обобщая понятия из теории выполнения транзакций [10], определим:

S – некоторый сценарий (расписание) выполнения операций, $S = < A_S, \perp_S >$;

A_S – множество действий, участвующих в сценарии S, $A_S = A(P_S)\ 4\ \tilde{A}(T(P_S))$;

P_S – множество процессов, участвующих в сценарии S;

$A(P_S)$, $T(P_S)$ – множества действий и транзакций, выполняемых процессами из P_S;

$\tilde{A}(T(P_S))$ – управляющие транзакциями из $T(P_S)$ примитивы в той или иной модели;

$\perp_S$ – отношение частичного порядка предшествования действий в сценарии S, $\perp_S \ \overline{(A_S \% A_S)}$.

Как правило, для транзакций определяют следующие основные примитивы (состояния):

begin(t_i) или t_i^{*} – начало транзакции t_i;

commit(t_i) или t_i^{+} – фиксация транзакции t_i;

abort(t_i) или t_i^{-} – отмена транзакции t_i.

Описывая модели с промежуточной фазой фиксации, можно использовать примитив *prepare*(t_i) – подготовка t_i к фиксации результатов.

Помимо вышеуказанных основных примитивов для описания многопоточных процессов можно определить также и некоторые дополнительные [11]:

split(t_i) – разбиение t_i;

join($t_{i1},\ldots,t_{in}$; t_i) – слияние $t_{i1},\ldots, t_{in}$ в t_i;

spawn(t_i) – порождение подпроцессов транзакцией t_i;

delegate(t_i,t_j) – делегирование транзакцией t_i некоторой другой транзакции t_j полномочий по фиксации, отмене, доступу и т. д.

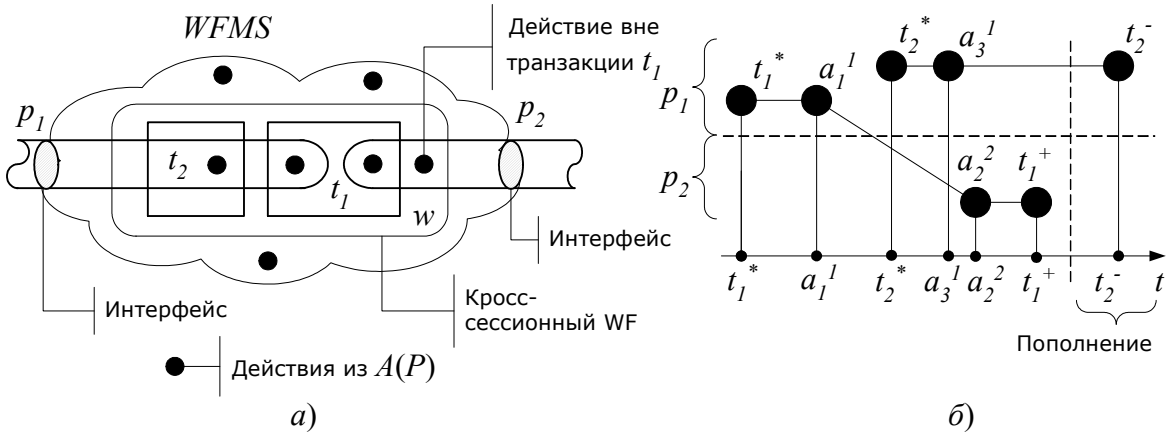

Рис. 1. Модель WFMS и ее возможный сценарий

Порядок действий в том или ином сценарии обусловлен заранее определенным набором зависимостей между операциями, т. е. внутренней логикой технологических процессов, входными данными и способом обработки конкурентных действий в данной модели. Для удобства записи сценария будем использовать последовательность элементарных действий: a_{i1} , a_{i2} , … , a_{in}, где a_{ij} χ A_S , a_{ij} $\perp_S$ a_{ik} для $i_j < i_k$. Для исследования степени перекрытия операций действия a_i в модели могут быть сколь угодно детализированы.

Назовем сценарий *допустимым*, если он в принципе возможен для данной модели. В дальнейшем будем рассматривать только допустимые сценарии.

Транзакция считается *завершенной*, если все действия по ее выполнению завершены, т. е. находятся в одном из конечных состояний.

Сценарий, в котором все транзакции завершены, – *полный*, соответственно в противном случае сценарий *неполный*.

Можно сказать, что *проблема отказоустойчивости* сводится к восстановлению, т. е. пополнению оставшихся после отказа системы неполных сценариев. Аналогичным образом сценарий может пополняться действиями завершения тех операций, для которых предопределен и закончился интервал времени выполнения. Этот механизм используется для разрешения тупиковых ситуаций (взаимоблокировок) и для корректного закрытия сессии по истечении срока ее действия.

Назовем сценарий *упорядоченным*, если для любых двух транзакций справедливо то, что все действия одной транзакции предшествуют всем действиям другой:

$$\forall\ t_k, t_n \text{ либо все } a_i \perp_S a_j, \text{ либо все } a_j \perp_S a_i,\ a_i\ \chi\ A_S(t_k),\ a_j\ \chi\ A_S(t_n).$$

На рис. 1, *б* приведен пример неполного неупорядоченного сценария с перекрытием транзакций и с возможным пополнением.

Под *результатом* сценария S будем понимать данные, возвращаемые действиями сценария пользователям или другим внешним по отношению к WFMS компонентам. Следовательно, результат «внутренних» действий, включая «внутреннее» представление данных, не входит в результат сценария, что позволяет представлять один и тот же WF различным набором действий, сохраняя результат. При этом «внешние» действия в совокупности образуют *интерфейс* WFMS, полностью определяемый технологическими процессами из множества W. Обозначим:

$R(S)$ – результат выполнения сценария S;

B – структура данных, используемых технологическими процессами из множества W;

$B(S)$ – конечное состояние «внутренних» данных после выполнения сценария S; из свойств транзакции имеем, что данные $B(S)$ всегда согласованы для любого полного сценария S;

B^0 – некоторое начальное состояние «внутренних» данных, влияющее на ход сценария S наряду с «внешними» входными данными Q (request).

Сценарии S и S' *эквивалентны* ($S' \sim S$) тогда и только тогда, когда они имеют один и тот же результат и эквивалентные состояния данных после своего выполнения для одних и тех же начального состояния B^0 и входных данных Q в процессах P.

Если под корректностью вычислений понимать упорядочиваемость, то сценарий S *корректен* тогда и только тогда, когда он упорядочен или существует упорядоченный сценарий S' с тем же набором транзакций $T(P_S)$, эквивалентный S. Следствием является то, что все сценарии, эквивалентные некоторому корректному сценарию, сами корректны.

Таким образом, для некоторой поставленной задачи *модель* M строит представление $<A, T>$ технологических процессов из множества W со структурой данных B и описывает способ выполнения конкурентных операций, т. е. способ построения плана (сценария) S в зависимости от этого представления $<A, T>$, состояния данных B^0, пользовательских и системных сессий P и запросов Q:

$$M: W \xi <A, T, B, S(A, T, B^0, P, Q)>.$$

Модель считается *корректной*, если все ее полные допустимые сценарии корректны, а для неполных определен механизм их пополнения до корректного сценария.

Следовательно, анализировать корректность и эффективность АТМ в комплексе можно, используя исследование допустимых сценариев, определив необходимые критерии и признаки корректности, а также функцию эффективности частного сценария $e(S)$: $\Sigma \rightarrow$ P, где Σ – множество допустимых сценариев модели M для задачи, описываемой W. Как правило, эффективность сопряжена со временем реакции системы на запросы – в этом случае получим анализ производительности.

Далее, применяя приведенную терминологию, рассмотрим некоторые характерные модели транзакционной обработки информации.

2. Вложенные транзакции

Данная модель представляет собой набор транзакций, каждая из которых может рекурсивно содержать другие транзакции, тем самым образуя дерево [12]. Индекс i в формальной записи транзакций становится многомерным:

$$A(t_{ij}) \quad \bar{A}(t_i), t_i, t_{ij} \chi T.$$

Дочерняя транзакция t_{ij} может начаться только после начала родительской t_i, и родительская транзакция, в свою очередь, может завершиться только после завершения всех дочерних. Если родительская транзакция отменена, все дочерние тоже должны быть отменены, однако если дочерняя транзакция прервана, то родительская может сама выбирать способ восстановления, например выполнять другую транзакцию, осуществляющую альтернативное действие.

Для сохранения свойств ACID на всех уровнях данная модель обеспечивает полную изоляцию, т. е. используемые данные и результаты дочерней t_{ij} могут быть доступны для других параллельных процессов только после фиксации родительской t_i и так далее до верхнего уровня. Другими словами, дочерние транзакции находятся в «сфере управления» родительской. Различают следующие сферы управления [13]: атомарность процесса, фиксацию, контроль зависимости процессов, распределение ресурсов, восстановление, аудит, согласованность данных.

Поэтому такая модель вложенных транзакций называется *закрытой*. Она увеличивает модульность TWF, повышая его гибкость и гранулированность обработки исключительных

ситуаций. Тем не менее, модель имеет низкую степень параллелизма, что практически неприменимо в случае долгоживущих транзакций.

Критерий упорядочиваемости возможных сценариев для «закрытой» модели очевидным образом выполняется, потому что для любого неупорядоченного сценария S и любых его транзакций параллельных процессов t_k и t_i (t_k завершается раньше t_i) сценарий S' последовательного выполнения t_k и t_i эквивалентен S, так как в силу изоляции t_k не использует результатов t_i. Если при этом t_i также не использует результатов t_k, то они могут выполняться параллельно.

3. Открытые вложенные транзакции

Открытые вложенные транзакции [14] ослабляют требование полной изоляции для закрытой модели, делая результаты выполнения дочерней t_{ij} видимыми для других процессов еще до фиксации родительской t_i, тем самым достигая большей степени параллелизма. При этом результаты t_{ij} могут оказаться неактуальными после дальнейшей модификации или отмены родительской t_i и использоваться несогласованно в некоторой транзакции t_k параллельного процесса. Чтобы разрешить проблему использования несогласованных данных, вводится дополнительное условие, гарантирующее соблюдение критерия упорядочиваемости:

Только перестановочные транзакции разных процессов t_i и t_j могут использовать результаты работы друг друга до фиксации родительских транзакций.

Транзакции t_i и t_j *перестановочны* (t_i (t_j), если результаты их работы и конечное состояние используемых данных не зависят от порядка выполнения этих транзакций, т. е. $t_i\, t_j \sim t_j\, t_i$. Неперестановочные транзакции называются *конфликтующими*. Как правило, понятие перестановочности рассматривается вне контекста выполнения операций, т. е. используются имя операции и входные параметры, хотя определяется также и контекстно-зависимая перестановочность, учитывающая дополнительно результаты работы операций [14].

В общем случае родительская транзакция может ограничивать доступ к результатам дочерних, определяя перестановочность конкурирующих с ними транзакций. Если t_k использует результат t_{ij}, но t_k (t_{ij}, то исходный сценарий S корректен и эквивалентен выполнению одной t_k (рис. 2, *а*): $S = t_{ij}\, t_k\, t_i^- \sim t_k$ ($t_{ij}\, t_i^-$) $\sim t_k$, где t_i^- – отмена транзакции t_i и, следовательно, всех дочерних t_{ij}. В противном случае конфликт может быть разрешен ожиданием транзакцией t_k завершения t_i.

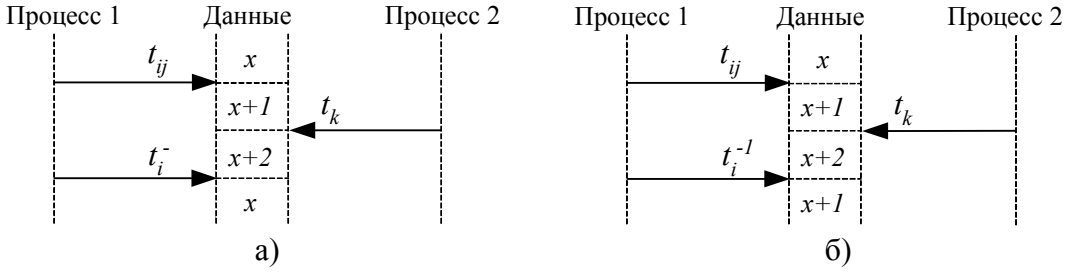

Рис. 2. Параллельное увеличение счетчика

Традиционно перестановочными считаются транзакции только чтения, хотя и некоторые транзакции записи логически также могут быть перестановочными, например операции увеличения счетчика. Однако следует учитывать, что если под отменой транзакции записи t понимать возврат к исходному состоянию на момент ее начала, то под используемыми данными в определении свойства перестановочности также следует понимать и сохраненные исходные значения – буфер отката $b(t)$ для осуществления собственно отката этой транзакции.

При такой схеме рассмотрим, например, случай со счетчиком. Несмотря на то, что независимо от порядка выполнения t_{ij} и t_k (рис. 2, *а*) получается одинаковый результат (x+2), транзакции увеличения t_{ij} и t_k неперестановочны, так как конечные состояния используемых данных в целом не совпадают, а именно значение $b(t_{ij}\, t_k)$ после $t_{ij}\, t_k$, равно (x, $x+1$), а после $t_k\, t_{ij}$ значение b($t_k\, t_{ij}$) равно ($x+1$, x), где x – исходное значение счетчика.

Таким образом, сценарий $(t_{ij}\ t_k)\ t_i^-$ дает результат x, в то время как $(t_k\ t_{ij})\ t_i^- \sim t_k$ даст результат (x+1), что иллюстрирует конфликт $t_{ij}t_k \uparrow t_k t_{ij}$.

4. Компенсация

Компенсирующая модель (Saga) [15] применима к задачам реализации долгоживущих транзакций и состоит как из набора ACID транзакций t_{ij} и операций $t_i = t_{i1}\ t_{i2}\ \ldots\ t_{in}$, собственно определяющих порядок их выполнения, так и из набора предопределенных компенсирующих ACID транзакций t_{ij}^{-1}, семантически отменяющих действие t_{ij}. Другими словами, $t_{ij}\ t_{ij}^{-1}$ не оказывает влияния на сценарий. Обозначим $(t_{ij}\ t_{ij}^{-1})$ как t^0 – *нейтральную* операцию:

$$\forall\ t\ \chi\ T\colon t\ t^0 \sim t^0\ t \sim t.$$

Операция t_i считается выполненной успешно тогда и только тогда, когда все транзакции t_{ij} завершены успешно; иначе, если какая-либо t_{ik} неуспешна, компенсирующие транзакции $t_{i,k\text{-}1}^{-1}\ \ldots\ t_{i1}^{-1}$ должны быть выполнены, чтобы полностью отменить все предыдущие шаги операции t_i.

Компенсационный подход так же, как и открытые вложенные транзакции, ослабляет требование полной изоляции, делая результаты промежуточных шагов t_{ij} видимыми еще до завершения основной операции t_i и увеличивая степень параллелизма. При этом компенсационная модель фиксирует результаты выполнения шагов t_{ij} и, следовательно, сохраняет свойство надежности (перманентности результатов) на всех уровнях – как для t_i, так и для t_{ij}, хотя и затрачивает на это дополнительное время.

Обратная транзакция t_{ij}^{-1} в общем случае семантически может и не базироваться на исходных данных t_{ij} для осуществления компенсации t_{ij}, избавляя от необходимости поддерживать буфер отката, т. е. $b(t_{ij}) = \varnothing$. Возвращаясь к примеру со счетчиком (рис. 2, *а*), обратной транзакцией его увеличения может быть транзакция уменьшения – t_{ij}^{-1} и t_k^{-1}. В этом случае t_{ij} и t_k перестановочны, так как $b(t_{ij}) = b(t_k) = \varnothing$, следовательно, результат и конечное состояние данных одни и те же независимо от порядка, т. е. $t_{ij}\ t_k \sim t_k\ t_{ij}$.

Модель не обязательно возвращает данные в исходное состояние при отмене и, следовательно, ослабляет свойство атомарности, сохраняя корректность путем возврата данных в состояние, логически эквивалентное исходному. Например, компенсируя транзакцию вставки ключа в B-дерево, при которой произошло разбиение некоторой вершины, достаточно удалить ключ без отката разбиения этой вершины [16]. На рис. 2, *б* t_i^{-1} – отмена основной операции, ведущая к компенсации t_{ij}^{-1}. В отличие от ситуации на рис. 2, *а*, сценарий эквивалентен одному увеличению счетчика t_k и дает результат (x+1):

$$(t_{ij}\ t_k)\ t_i^{-1} \sim (t_{ij}\ t_k)\ t_{ij}^{-1} \sim t_k\ (t_{ij}\ t_{ij}^{-1}) \sim t_k.$$

Таким образом, откат t_i^- и компенсация t_i^{-1} – логически разные операции. Отметим, что компенсационная модель требует дополнительных затрат на реализацию семантически обратных транзакций.

Аналогично открытым вложенным транзакциям критерий упорядочиваемости для компенсации выполняется при условии перестановочности конкурентных операций, использующих результаты друг друга. При этом некоторые конфликтующие транзакции в открытой модели могут стать перестановочными при компенсации.

Обобщением компенсационной модели является вложенная компенсационная модель, хорошо структурирующая шаги долгоживущей транзакции в иерархическую схему [17].

5. Многоуровневые транзакции

Многоуровневые транзакции [6, 14] являются разновидностью открытой вложенной модели, где дочерние транзакции t_{ij} могут делать видимыми результаты своего выполнения, а также фиксировать их до завершения родительской t_i. При этом дочерние транзакции t_{ij} могут быть отменены как возвращением в исходное состояние t_{ij}^-, так и компенсацией t_{ij}^{-1}.

Согласно принципам вложенных транзакций в случае отказа t_{ij} родительская t_i не обязательно должна быть отменена. Выполнив некоторые альтернативные действия, она может успешно завершиться. «Открытые» транзакции (включая компенсируемые) должны контролировать использование результатов работы дочерних, например, путем блокировки конфликтующих транзакций, тем самым порождая новую «сферу управления» [13, 14], в то время как «закрытые» транзакции лишь расширяют сферу управления родительских, т. е. их изменения остаются изолированными от внешних процессов.

На рис. 3, *а* можно видеть пример многоуровневой транзакции, состоящей из следующих транзакций:

закрытых – t_{13}, t_{111}, t_{112}, t_{121};

открытых – t_{11}, t_{131};

компенсируемых (и, следовательно, также открытых) – t_{12}, t_{122}, предполагающих наличие логически обратных транзакций t_{12}^{-1} и t_{122}^{-1} соответственно;

основной – t_1, результаты которой окончательны.

Придавая прикладной смысл вышеуказанным операциям, определим:

t_1 - операция (технологический процесс) покупки (*shopping*);

t_{11} - выбор продукции/сервиса (*choice*), при этом в момент выбора некоторые компоненты могут оказаться выбранными другими пользователями, тогда предлагаются альтернативы; результаты выбора видны остальным пользователям после завершения *choice*;

t_{111}, t_{112} – выбор одного продукта/комплектующего (*selection*);

t_{12} – оплата выбранной продукции (*payment*), которая может быть отменена в случае неудачи основного технологического процесса покупки операцией t_{12}^{-1};

t_{121} – проверка возможности осуществления данной покупки (*validation*);

t_{122} – денежный перевод (*transfer*);

t_{122}^{-1} – обратный перевод;

t_{13} – подтверждение получения продукта (*acknowledgement*), только после которого технологический процесс может завершиться успешно;

t_{131} – доставка продукта либо его возврат (*delivery*), при возврате продукции весь процесс покупки должен быть отменен.

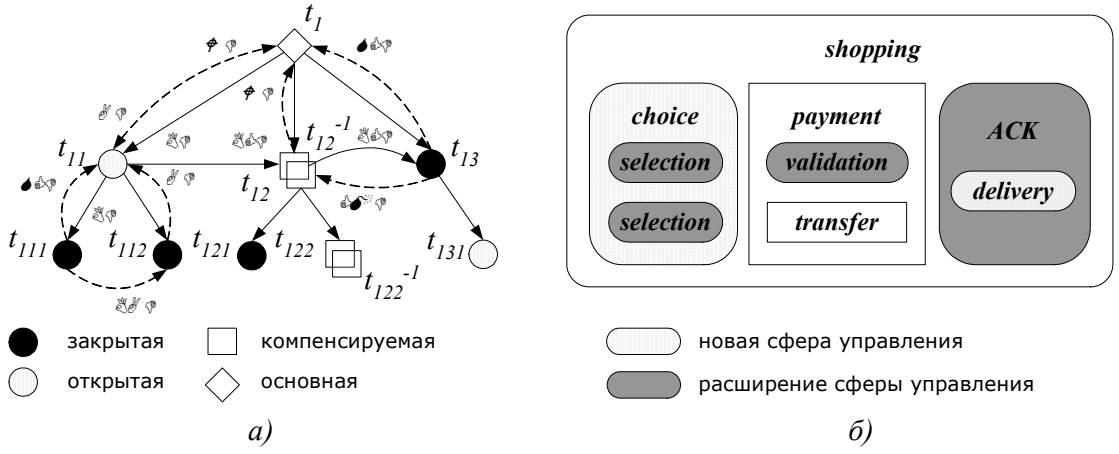

Рис. 3. Многоуровневые транзакции и сферы их управления

На рис. 3, *б* отражены границы сфер управления вышеуказанных операций. Транзакция *transfer* может принадлежать банковской системе, в то время как *payment* – порталу. В данной бизнес-логике несколько операций *shopping* могут перекрываться, не нарушая корректности. При отказе в t_{111} (*selection*) и t_{13} (*acknowledgement*) сценарий выполнения транзакции t_1 (*shopping*) будет нейтральным, соблюдая атомарность, при этом транзакция *shopping* окажется не выполненной совсем, а используемые ресурсы полностью разблокированы и освобождены:

$$shopping^{*}\ (choice^{*}\ selection_1^{-}\ selection_2^{+}\ \mathrm{prepare}(choice))\ (payment^{*}\ validation^{+}\ transfer^{+}$$

$payment^{+})\ ack^{-}\ (transfer^{-1}\ payment^{-1}\ choice^{-}\ shopping^{-}) \sim t^{0}$.

После отмены *ack* операции $transfer^{-1}$ $payment^{-1}$ $choice^{-}$ $shopping^{-}$ составляют процедуру отмены, которая должна быть гарантированно выполнена. В случае сбоя системы эта процедура будет частью процедуры восстановления, т. е. пополнения сценария.

Многоуровневая модель, в которой для любой транзакции все поддеревья имеют одинаковую высоту, называется *многослойной*. В такой модели функционал разбивается на уровни, и листья представляют собой набор базовых транзакций.

6. K-упорядочиваемая модель

Рассмотрим операцию выбора продукции/сервиса (*choice*) в многослойной модели, состоящую из транзакций выбора компонентов определенных типов (*select)*. *Select*(X_i) выбирает наиболее предпочтительный доступный компонент типа X_i и помечает его как «выбранный». После фиксации выбора (*choice*) выбранные компоненты удаляются из списка компонентов соответствующих типов. Обратная операция *deselect*(x) возвращает выбранный элемент x в статус «доступный».

Конкурентные операции выбора (*choice*) конфликтуют (рис. 4). При их последовательном выполнении – $choice^{1}$, $choice^{2}$ – обе транзакции будут работать с наиболее предпочтительным элементом a типа X. В случае перекрытия этих транзакций $choice^{2}$ будет работать с элементом b – вторым по предпочтению.

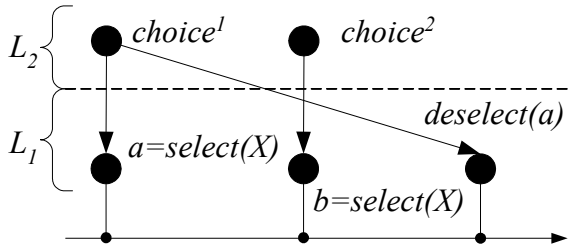

Рис. 4. *K*-упорядочиваемость

В общем случае будем говорить, что уровень (слой) L_i *k-упорядочиваем* относительно уровня L_{i+1}, если операции уровня L_i могут быть упорядочены относительно уровня L_{i+1} с помощью не более чем k перестановок конфликтующих операций [18]. Для всего сценария n-слойной модели в целом определяется вектор $K = (k_1, ..., k_{n-1})$, где k_i – показатель перестановочности уровня L_i относительно L_{i+1}, и сценарий называется *K*-упорядочиваемым.

$K=0^{n-1}$ равносильно полной упорядочиваемости сценария, т. е. в нашем примере выбор (*select*) будет всегда оптимальным. На рис. 4 сценарий (1)-упорядочиваемый. Это значит, что и вторые по оптимальности элементы тоже допустимы, что ведет к увеличению степени параллелизма, и т. д.

Ослабляя критерий упорядочиваемости, родительские транзакции могут контролировать допустимый показатель перестановочности дочерних, тем самым добиваясь компромисса между оптимальностью результата и производительностью.

1. Семантическая блокировка

Наряду с традиционной блокировкой данных для управления параллельным выполнением операций чтения/записи можно также использовать более высокоуровневую семантическую блокировку [19]. В этом случае транзакции разбиваются на шаги, и каждый шаг a_i перед использованием некоторого элемента данных x устанавливает семантическую блокировку $L(a_i,x)$ после установления традиционной блокировки x. После завершения шага a_i традиционная блокировка снимается, а семантическая конвертируется в $L(a_{i+1},x)$, т. е. в блокировку элемента x, следующим шагом транзакции a_{i+1}.

Конфликты шагов конкурентных транзакций можно декларировать матрицей конфликтов для всех объектов данных, *ij*-е элементы которой отражают конфликт шага a_i некоторой транзакции t и a_j некоторой транзакции t'. В случае конфликта с существующими блокировками шаг транзакции должен ждать разрешения этого конфликта путем конвертации блокировки конкурентными транзакциями.

Корректность данной модели гарантирована при условии правильного определения матрицы конфликтов с точки зрения бизнес-логики технологических процессов. Декомпозицией технологических процессов из W на действия A и транзакции T можно влиять на степень перекрытия операций в допустимых сценариях.

7. Модель ACTA

Модель ACTA (по латыни «действия») предоставляет транзакциям работу с данными через методы соответствующих объектов данных. Каждый объект имеет тип, определяющий набор применимых к его данным методов [20]. Транзакция связана с событиями вызовов методов (object events) и событиями управления транзакцией (significant events). Управляющие события содержат в себе инициирующие (такие, как *begin*) и завершающие (такие, как *commit*, *abort*) события. В многопоточных моделях [11] инициирующими событиями могут быть *begin*, *split*, а завершающими – *commit*, *abort*, *join*. Транзакция координирует вызов и завершение используемых методов, однако некоторые задачи координации она может делегировать другой транзакции.

Таким образом, A, $\tilde{A}(T)$ – это object events и significant events соответственно. Для исследования поведения конкурентных транзакций ACTA определяет следующие виды зависимостей D между ними [20]:

Commit Dependency (t_j XΔ t_i) – если t_i и t_j фиксируют свои результаты, то фиксация t_i предшествует фиксации t_j;

Strong-Commit Dependency (t_j ΣXΔ t_i) – если t_i зафиксирована, то и t_j должна зафиксироваться;

Abort Dependency (t_j AΔ t_i) – если t_i отменена, то и t_j должна отмениться;

Weak-Abort Dependency (t_j ΩΔ t_i) – если t_i отменена, а t_j еще не зафиксирована, то t_j также должна быть отменена;

Termination Dependency (t_j TΔ t_i) – t_j не может завершиться до завершения t_i;

Exclusion Dependency (t_j EΔ t_i) – обе t_j и t_i не могут быть зафиксированными;

Force-Commit-on-Abort (t_j XMΔ t_i) – если t_i отменена, то t_j должна быть зафиксирована;

Begin Dependency (t_j BΔ t_i) – t_j не может начаться до начала t_i;

Serial Dependency (t_j ΣΔ t_i) – t_j не может начаться до завершения t_i;

Begin-on-Commit Dependency (t_j BXΔ t_i) – t_j не может начаться до фиксации t_i;

Begin-on-Abort Dependency (t_j BAΔ t_i) – t_j не может начаться до отмены t_i;

Weak-begin-on-Commit (t_j ΩXΔ t_i) – если t_i зафиксирована, то t_j может начаться после фиксации.

Продолжая пример многоуровневой транзакции, диаграмма на рис. 3, *а* частично отражает «внутренние» (intra-transactional) зависимости. Согласно этим зависимостям D и текущему сценарию S можно определить предикаты, а следовательно, и множества видимости и конфликтов транзакции $t \chi T$ с результатами методов $a_i \chi A(T_S)$:

$$V(t, S, D) = \{ a_i \chi A(T_S) : t \text{ видит результат } a_i \};$$
$$C(t, S, D) = \{ a_i \chi A(T_S) : t \text{ конфликтует с } a_i \}.$$

Таким образом, транзакция t может вызывать метод a некоторого объекта только в случае, когда все методы, вызванные другими незавершенными конкурентными транзакциями над этим объектом, видимы для t и не конфликтуют с ней [20]. В этом случае выполнение t будет корректным. Данный подход реализован в системе поддержки расширенных транзакций ASSET [8].

8. Гибкая транзакционная модель

Гибкая модель описывается набором субтранзакций t_{ij}, множеством их состояний $\tilde{A}\{t_{ij}\}$, множеством зависимостей между этими состояниями D (как в ACTA) и множеством приемлемых состояний t_{ij}, описывающих успешное выполнение транзакции t_i.

Например, для t_i множество приемлемых состояний может быть определено как $\{<commit(t_{i1})>,<abort(t_{i1}),commit(t_{i2})>\}$. Выполнение транзакции t_i начинается с выполнения субтранзакций t_{ij}, удовлетворяющих условиям старта согласно зависимостям D. Как только изменяется статус какой-либо из t_{ij}, возможны следующие ситуации [21]:

- достигнуто одно из приемлемых состояний для t_i, в этом случае переход к завершающей фазе фиксации;
- ни одна из t_{ij} не выполняется, ни одна из t_{ij} не удовлетворяет условиям старта, ни одно из приемлемых состояний не достигнуто, в этом случае переход к завершающей фазе отмены;
- во всех остальных случаях фаза выполнения продолжается.

Завершающая фаза фиксации t_i останавливает все еще выполняемые t_{ij}, фиксирует, отменяет или компенсирует необходимые t_{ij} согласно требуемому состоянию. Завершающая фаза отмены t_i отменяет или компенсирует все t_{ij}. Корректность должна быть обеспечена контролем видимости данных.

9. Гранулированные технологические процессы

В данной модели транзакционный технологический процесс представляется в виде набора задач t и следующих управляющих этими задачами конструкций [22]:

порядок (ordering) – диктуемый требованиями технологического процесса основной порядок выполнения задач;

случай сбоя (contingency) – запланированные задачи на случай отказа основных;

альтернатива (alternative) – набор альтернативных задач, где система имеет свободу выбора любой из альтернатив;

условие (conditional) – выполнение задачи только при некотором условии;

итерация (iteration) – повторение задачи при некотором условии.

Задачей также может быть и вложенный технологический процесс. Задача имеет набор состояний $\tilde{A}(t)$ и переходов между ними. Обязательными являются начальное состояние (initial), множества состояний фиксации и отмены (для описания альтернативных путей выполнения задачи). Состояние всего TWF в целом определяется состояниями входящих в него задач. Таким образом, выполняя задачи согласно управляющим конструкциям, TWF сам переходит из одного состояния в другое.

Конструкции гранулированного технологического процесса могут быть сколь угодно разнообразными, при этом для исследования данной модели важны лишь зависимости между состояниями $\tilde{A}(T)$, что сводит данную модель к ACTA.

10. Асинхронные модели

В архитектуре систем управления технологическими процессами (WFMS) часто встречается асинхронная передача сообщений, т. е. действия $A=\{a_i\}$ могут содержать в себе отправку/получение сообщений. Такой подход позволяет иметь временно доступные ресурсы (рис. 5) и инициировать задания в фоновом режиме. К тому же координация распределенных технологических процессов в этом случае может осуществляться без единого для всех систем центра путем обмена персистентными сообщениями для перехода от одного шага транзакции к другому [23].

Операции отправки/получения сообщений могут образовывать транзакцию t вместе с другими ее действиями $a_i \chi A(t)$, причем действия $A(t)$ не обязательно должны выполняться в одном процессе p_m.

Если под атомарностью долгоживущей транзакции t понимать тот факт, что либо все сообщения внутри t прочитаны и обработаны, либо ни одно из них не обработано, то в случае отмены t транзакция отзывает непрочитанные сообщения и посылает соответствующие

сообщения для отмены реакций на уже обработанные сообщения. Другими словами, функциональные вызовы и обмен сообщениями образуют единую сферу управления [24].

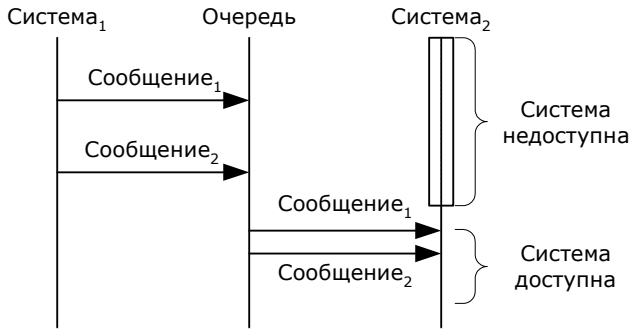

Рис. 5. Асинхронная обработка

Заключение

Представленный подход в совокупности с исследованными технологиями может быть результативно применен при разработке систем управления технологическими процессами, а также при анализе их корректности и эффективности. Отметим, что транзакции внутри технологических процессов могут быть вложенными, образовывая многоуровневую систему с контролем видимости данных и компенсацией. Видимость и согласованность данных можно контролировать семантическими блокировками, причем для увеличения производительности допускается перекрытие операций, при котором результат будет неоптимальным, если это приемлемо с точки зрения условий задачи.

Для короткоживущих транзакций, как правило, применяется закрытая модель – она наименее сложная по трудоемкости, и блокировки обеспечивают упорядочиваемость допустимых сценариев, что гарантирует корректность. Долгоживущие транзакции нуждаются в компенсации, так как долговременные блокировки данных практически неприемлемы с точки зрения степени параллелизма on-line транзакций. В этом случае логика работы системы должна учитывать возможную компенсацию.

Фоновые процедуры (off-line транзакции), для которых требования по времени реакции системы не столь строгие, могут работать в открытой модели, оказывая как можно меньшее влияние на время ответа интерактивных задач. Фоновые процедуры между собой могут использовать оптимистический подход, при котором данные не блокируются, а транзакция может выполниться успешно тогда и только тогда, когда все транзакции, результаты работы которых были использованы, также завершились успешно. Оптимистический подход эффективен в случае маленького процента отменяемых транзакций, т. е. почти все операции должны завершаться успешно.

Для спецификации логики транзакций можно устанавливать зависимости между состояниями этих транзакций. Определяющие конфликт операций предикаты позволяют отслеживать корректность. Используя эти зависимости, можно конструировать технологические процессы с помощью управляющих элементов и признаков успешного выполнения транзакций – приемлемых состояний. Эта модель достаточно гибкая, что упрощает и ускоряет процесс разработки и отслеживания «узких» мест системы.

Без ограничения общности вышеуказанные зависимости также распространяются и на распределенные системы, в которых переход от одной фазы (шага) транзакции к другой может осуществляться путем передачи персистентных сообщений в асинхронном режиме. Комбинируя эти подходы, в рамках одного технологического процесса могут выполняться и on-line запросы, и фоновые задачи. При любом подходе отказоустойчивость гарантируется процедурами пополнения отмененных транзакций и незаконченных сценариев. Понятие «законченного» сценария в архитектуре системы делает минимальным риск появления ситуации, требующей вмешательства оператора, что особенно актуально в финансовых системах.

Список литературы

1. Coalition. Workflow Reference Model Specification. – The Workflow Management Coalition, 1995. – 55 p.

2. Date C. An introduction to database systems. V. 6. – Addison-Wesley Longman Inc, 1995. – 846 p.

3. Connolly T. M., Begg C. E. Database Systems. A Practical Approach to Design, Implementation and Management. Third Edition. – Addison-Wesley, 2003. – 1440 p.

4. Worah D., Sheth A. Transactions in Transactional Workflows. In Advanced Transactional Models and Architectures. – Kluwer Academic Publishers, 1997. – 400 p.

5. Advanced Transaction Models in Workflow Contexts / G. Alonso, D. Agrawal, A. Abbadi et al. // In proc. of the 12th International Conference on Data Engineering. – New Orleans, 1996

6. Elmagarmid A. Database Transaction Models for Advanced Applications. – Morgan Kaufmann Publishers, 1992. – 610 p.

7. Gray J., Reuter A. Transaction Processing: Concepts and Techniques. – Morgan Kaufmann Publishers, 1993. – 1070 p.

8. ASSET: A System for Supporting Extended Transactions / A. Birilis., S. Dar., N. Gehani et al. // In proc. of SIGMOD International Conference on Management of Data. – 1994. – P. 44-54.

9. The Open Group. Distributed Transaction Processing: The XA Specification. X/Open Document C193. – X/Open Company Ltd., UK, 1991. – 80 p.

10. Vingralek R., Ye H., Breitbart Y., Schek H. J. Unified Transaction Model for Semantically Rich Operations // ICDT (International Conference on Database Theory). – 1995. – 25 p.

11. Pu C., Kaiser G., Hutchinson N. Split-Transactions for Open-Ended activities // In proc. of the 14th International Conference on VLDB. – 1988. – P. 26-37.

12. Moss J. Nested Transactions and Reliable Distributed Computing // In proc. of the 2nd symposium on Reliability in Distributed Software and Database Systems. – IEEE CS Press., 1982.

13. Davies C. Data Processing Spheres of Control // IBM Systems Journal. – 1978. –V.17.– № 2.

14. Weikum G., Shek H. Concepts and applications of multilevel and open-nested transactions. In Elmagarmid A. Database Transaction models for Advanced Applications. – Morgan Kaufmann Publishers, 1992. – 39 p.

15. Garcia-Molina H., Salem K. Sagas // In proc. of ACM SIGMOD Conference on Management of Data. – San Francisco, CA, 1987.

16. ARIES (Algorithm for Recovery and Isolation Exploiting Semantics): A Transaction Recovery Method Supporting Fine-Granulatity Locking and Partial Rollback Using Write-Ahead Logging. IBM Research Report RJ6649 / C. Mohan, D. Haderle, B. Lindsay et al. // ACM Transactions on Database Systems. – 1992. – V. 17. – № 1.

17. Modeling Long-Running Activities as Nested Sagas / H. Garcia-Molina, K Salem, D. Gawlick et al.// IEEE Data Engineering Bulletin. – № 14(1). – 1991.

18. Krychniak P., Rusinkiewicz M., Cichocki A., et al. Bounding the Effects of Compensation under Relaxed Multi-level Serializability // Distributed and Parallel Databases. – 1996. – V. 4. – № 4.

19. Bernstein A., Gerstl D., Lewis P. A concurrency control for step-decomposed transactions // Information Systems. – 1999. – 35 p.

20. Chrysanthis P. K., Ramamritham K. ACTA: The SAGA Continues // Database Transaction models for Advanced Applications. – Morgan Kaufmann Publishers, 1992. – P. 435-470.

21. Ansari M., Ness L., Rusinkiewicz M., Sheth A. Using Flexible Transactions to Support Multi-System Telecom Applications // In proc. of the 18th International Conference on VLDB. – Canada. – 1992. – 12 p.

22. General model for nested transactional workflows. Technical report. DSTC. / D. Kuo, M.Lawley, C.Liu, M. Orlowska A – University of Queensland, 1996. – 18 p.

23. Exotica: A Project on Advanced Transaction Management and Workflow Systems / C. Mohan, A. Abbadi, D. Agrawal et al. // ACM SIGOIS. – 1995. – V. 16. – № 1.

24. Tai S., Mikalsen T., Rouvellou I., Sutton S. Dependency spheres: A global transaction context for distributed objects and messages // In 5th International Enterprise Distributed Object Computing Conference (EDOC). – September, 2001. – P. 10-22.

Поступила 19.03.04

Белорусский государственный университет,
Минск, пр. Ф. Скорины, 4
e-mail: vl@us.ibm.com

V.V. Leonovich

EFFECTIVE TRANSACTION MODELS IN WORKFLOW MANAGEMENT SYSTEMS

A problem of transactional workflow execution in workflow management systems is considered. A common terminology and an analysis method for the workflows which allows to improve the development process of complex distributed software systems are presented.