Статический анализ управления транзакциями в приложениях для платформы Java EE

И.В. Грачев, А.А. Соловьев
Кафедра ИСПИ
Владимирский государственный университет им. А.Г. и Н.Г. Столетовых

***Аннотация* — Ошибки в логике управления транзакциями могут привести к нарушению целостности данных в БД, а избыточное управление транзакциями вносит накладные расходы, влияющие на производительность программной системы. В статье предложена методика статического анализа приложений для платформы Java EE, включающая построение и абстрактную интерпретацию графа вызовов с точки зрения действий сервера приложений Java EE по управлению транзакциями, результатом которых являются выявленные ошибки в управлении транзакциями. Также в статье описан реализованный прототип анализатора, реализующего данную методику, и примененный в нем подход к визуализации управления транзакциями.**

Ключевые слова — управление транзакциями, статический анализ программных систем, платформа Java EE, граф вызовов методов, модель транзакционных контекстов

I. Введение

Программные системы обработки данных (ПС) полагаются на механизм транзакций для обеспечения согласованности данных при многопользовательской работе. Современные платформы для разработки распределенных ПС (CORBA, Java EE, Microsoft COM+), реализующие модель обработки распределенных транзакций X/Open DTP, переносят управление транзакциями с уровня СУБД на уровень промежуточного ПО. Предоставляемые ими возможности декларативного управления транзакциями, с одной стороны, упрощают разработку бизнес-логики с поддержкой транзакций (например, на платформе Java EE по умолчанию все бизнес-методы EJB-компонентов выполняются в контексте транзакции). С другой стороны, неправильная настройка управления транзакциями для компонентов ПС может приводить как к возникновению исключений во время выполнения ПС, так и к фактической утрате свойств атомарности и согласованности транзакций ПС, в первую очередь – распределенных транзакций. Последнее особенно характерно для начинающих разработчиков. Таким образом, реализация инструментального средства для выявления ошибок в управлении транзакциями (анализатора управления транзакциями) является актуальной задачей.

Объектом исследования являются ПС на платформе Java EE (в частности, Java EE 6). Выбор в пользу этой платформы сделан по причинам 1) ее широкого распространения и 2) реализуемым в ней планомерным упрощением управления транзакциями в компонентах разных типов за счет соглашений по умолчанию, что чревато потенциальными ошибками при управлении транзакциями в многослойных конфигурациях компонентов.

Выбор статического анализа для решения данной задачи обусловлен 1) возможностью применения на ранних этапах разработки без необходимости развертывания ПС и 2) сложностью обнаружить все возможные варианты поведения системы при динамическом анализе [1].

Используя разработанное тестовое приложение для платформы Java EE, выполняющее операции с БД в различных режимах переключения транзакционных контекстов при декларативном управлении транзакциями, удалось установить факт влияния накладных расходов по управлению транзакционными контекстами на производительность программной системы [2]. Для транзакций с малым количеством операций с БД излишний запуск и переключение транзакционных контекстов явились причиной более чем двукратного роста времени выполнения тестового сценария, тогда как обработка того же набора операций с БД в контексте одной транзакции привела к росту только на 20% относительно его выполнения в неопределенном транзакционном контексте. Поэтому в анализаторе управления транзакциями необходимо выявлять в исходном коде ПС не только ошибочное, но и неэффективное управление транзакциями.

II. Анализ управления транзакциями в компонентах Java EE-приложений

Результатом анализа управления транзакциями в данной работе являются решения по методам компонентов следующих видов:

1.Неопределенное поведение (решение WARNING) – работа с источниками данных в методах с неопределенным транзакционным контекстом.

2.Ошибки, приводящие к генерации системной исключительной ситуации на этапе выполнения (решение ERROR) вследствие несоблюдения определенных спецификацией [3] требований к программированию на платформе Java EE, например, обращение к методам

интерфейса javax.transaction.UserTransaction в методе компонента с декларативным управлением транзакциями.

3. Избыточное управление транзакциями (решение REDUNDANCY), приводящее к такому переключению транзакционного контекста, при котором в новом (в т.ч. неопределенном) контексте не выполняются операции с базой данных. Как показано выше, это влияет на производительность ПС.

Анализ управления транзакциями выполняется по исходному коду ПС в 3 этапа.

На первом этапе осуществляется построение по абстрактному синтаксическому дереву исходного кода ПС циклического орграфа (далее – графа вызовов), вершины которого представляют собой методы, а ребра – вызовы одного метода из другого.

С вершиной графа вызовов связаны: а) сигнатура метода; б) тип данных, для которого определен данный метод; в) набор метаданных, связанных с методом. В качестве типа данных указывается тип переменной, у которой вызывается метод. Как следствие, в графе вызовов может присутствовать несколько реализаций одного метода.

Приложения на платформе Java EE конфигурируются с использованием метаданных на этапах разработки и развертывания с использованием, соответственно, аннотаций в классах компонентов и установочных дескрипторов (например, ejb-jar.xml для EJB-компонентов). К обработке транзакций относятся элементы метаданных, определяющие способ управления транзакциями и задающие транзакционные атрибуты методов при декларативном управлении транзакциями. Метаданные распространяются по иерархии наследования и по структуре типа данных (от типа к методу), а метаданные этапа развертывания имеют приоритет над метаданными этапа разработки. Таким образом, при построении графа вызовов используется расширенный многоязычный контекст: помимо абстрактного синтаксического дерева анализируемого метода необходимы данные обо всех типах данных ПС (Java) и установочных дескрипторах (XML).

Граф вызовов формируется без учета условных и циклических операторов, то есть для вызова метода всегда создается одно ребро, в том числе, если вызов выполняется при определенном условии либо в цикле. (Эта эвристика не позволяет корректно обработать ситуацию, когда в цикле из метода с программным управлением транзакциями возвращается новый контекст транзакции, который закрывается на последующих итерациях, но она значительно упрощает анализ остальных, более распространенных случаев.) Количество дуг между двумя вершинами-методами соответствует количеству вызовов с учетом указанного ограничения. Ребра, исходящие из одной вершины-метода, упорядочены в порядке вызова методов в исходном коде.

На втором этапе граф вызовов модифицируется с учетом полиморфных вызовов и гибкого связывания компонентов. Результатом является граф вызовов, в котором вызовы методов, полиморфные в рамках анализируемой ПС, заменены на конкретные реализации, которые определяются в результате анализа метаданных и исходного кода, и все вершины-методы без входящих дуг являются точками входа.

Под точкой входа будем понимать метод, с которого Java EE-приложение по запросу клиента начинает выполняться в управляемом контейнером режиме. К подобным точкам входа относятся: метод service() сервлета, обработчики событий и действий JSF в управляемых бинах и EJB-компонентах, конечные точки веб-сервисов и публичные методы удаленного представления EJB-компонентов. Фактические точки входа за пределами исходного кода приложения, такие как метод service() универсального сервлета-контроллера каркаса JSF, не рассматриваются.

Связывание экземпляров компонентов платформы Java EE между собой происходит во время выполнения на основании конфигурации приложения (метаданных). В рамках второго этапа анализируется два варианта связывания компонентов: 1) внедрение зависимостей с помощью аннотаций CDI и EJB; 2) поиск зависимостей в JNDI-каталоге. При этом область поиска типов ограничивается анализируемым Java EE-приложением, то есть предполагается, что все необходимые типы (классы, интерфейсы) определены внутри проекта с исходным кодом ПС или находятся в используемых библиотеках, а также отсутствует динамическая загрузка классов во время выполнения и вызовы методов через механизм рефлексии.

В ситуации, когда на основании метаданных невозможно определить единственную реализацию полиморфного метода, в графе вызовов дуга вызова этого метода заменяется на дуги вызовов для всех найденных реализаций с одинаковым порядковым номером.

После замены методов в графе вызовов помечаются все вершины, достижимые из точек входа. Недостижимые вершины, соответствующие полиморфным реализациям методов, не выбранным при связывании компонентов, удаляются из графа.

На рис. 1 приведен пример модификации графа вызовов, структура которого аналогична типовому Java EE-приложению, в котором выделяются уровни логики представления (класс компонента Ctrl), прикладной логики (интерфейс IModel с двумя альтернативными реализациями – классами компонентов Model и Model2) и логики управления данными (интерфейсы IDao и IDao2, классы компонентов Dao и Dao2). На рис. 1.а) показан исходный граф вызовов, который после связывания компонентов в предположении выбора класса компонента Model в качестве реализации интерфейса IModel принимает вид 1.б). На рис. 1.в) выделены точка входа и достижимые из нее вершины. После удаления недостижимых вершин модифицированный граф вызовов принимает окончательный вид 1.г).

На третьем этапе на основании модифицированного графа вызовов строится модель транзакционных контекстов.

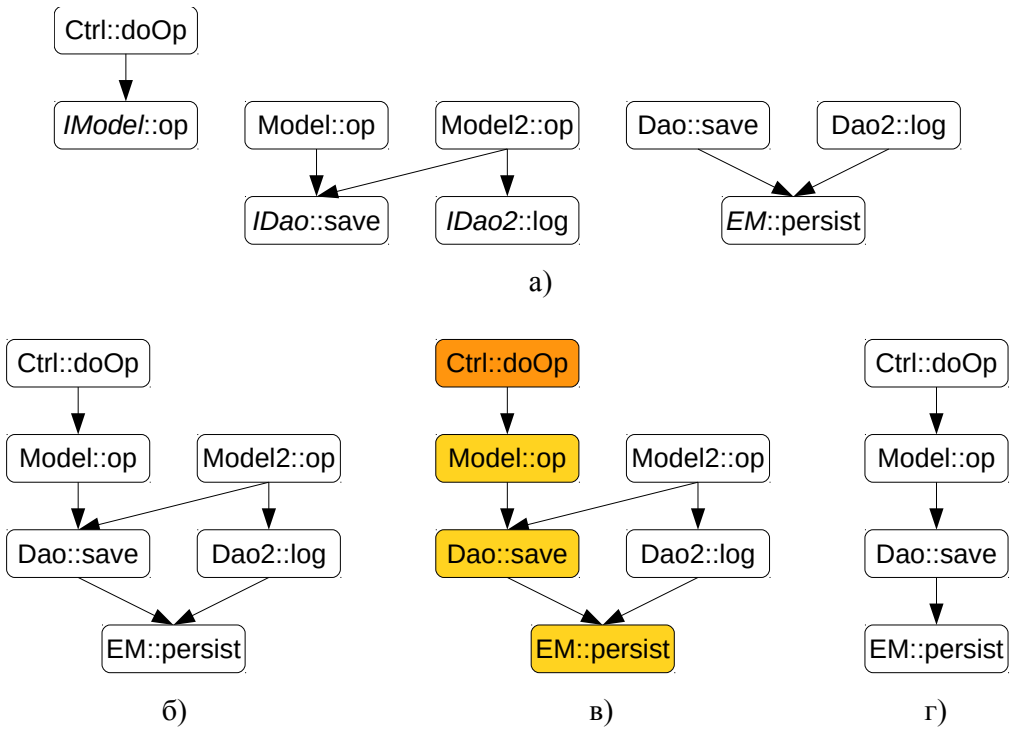

Рис. 1. Пример модификации графа вызовов

Модель транзакционных контекстов состоит из развернутого графа вызовов, множества обнаруженных в нем транзакционных контекстов и множества решений по методам ПС.

Развернутый граф вызовов является формой представления модифицированного графа вызовов, в котором, для удобства последующей визуализации, каждый вызов метода представлен отдельной вершиной с входящей в нее дугой из вызывающего метода.

Транзакционный контекст идентифицируется порядковым номером и содержит тип управления контекстом (программное либо декларативное управление), флаги завершенности и неопределенного поведения, множество вершин-методов, которые выполняются в рамках данного транзакционного контекста, множество решений по контексту.

Построение модели транзакционных контекстов представляет собой рекурсивный обход вершин-методов графа вызовов, начиная с вершин без входящих дуг, так как в модифицированном графе вызовов все подобные вершины по построению являются точками входа в ПС.

Для каждой точки входа вызывается рекурсивная процедура analyzeRecursively (блок-схема которой приведена на рис. 2), принимающая в качестве параметра вершину-метод в модифицированном графе вызовов. Также процедуре доступен текущий транзакционный контекст, причем предполагается, что точка входа начинает свое выполнение в отсутствие активного контекста транзакции, и это представляется фиктивным контекстом NO_TX. Процедура analyzeRecursively формирует развернутый граф вызовов и заполняет множества решений по всем рассматриваемым методам. При обработке каждой исходящей дуги из переданной в качестве параметра вершины-метода: 1) по типу управления транзакциями текущего транзакционного контекста проверяется, не запрещен ли вызываемый метод; 2) определяется поведение контейнера Java EE-компонентов по переключению транзакционного контекста (распространение текущего контекста на вызываемый метод, создание нового контекста, приостановка текущего контекста, приостановка текущего контекста и создание нового, отсутствие каких-либо действий) и транзакционный контекст, который будет установлен при выполнении вызываемого метода. Тем самым выполняется абстрактная интерпретация графа вызовов с точки зрения действий сервера приложений Java EE по управлению транзакциями.

Модифицированный граф вызовов может содержать циклы, что требует особой обработки в рамках поставленной задачи: рекурсивные вызовы метода обрабатываются до тех пор, пока не повторяется поведение по переключению транзакционного контекста, что определяется по истории вызовов. Тем самым максимальное количество анализируемых уровней рекурсии ограничено сверху количеством типов поведения по переключению транзакционного контекста, и алгоритм является конечным.

В случае наличия в истории вызовов поведения, связанного с вызываемым методом, в развернутый граф вызовов добавляется фиктивная дуга, обозначающая найденный цикл, и проверка вершины – вызываемого

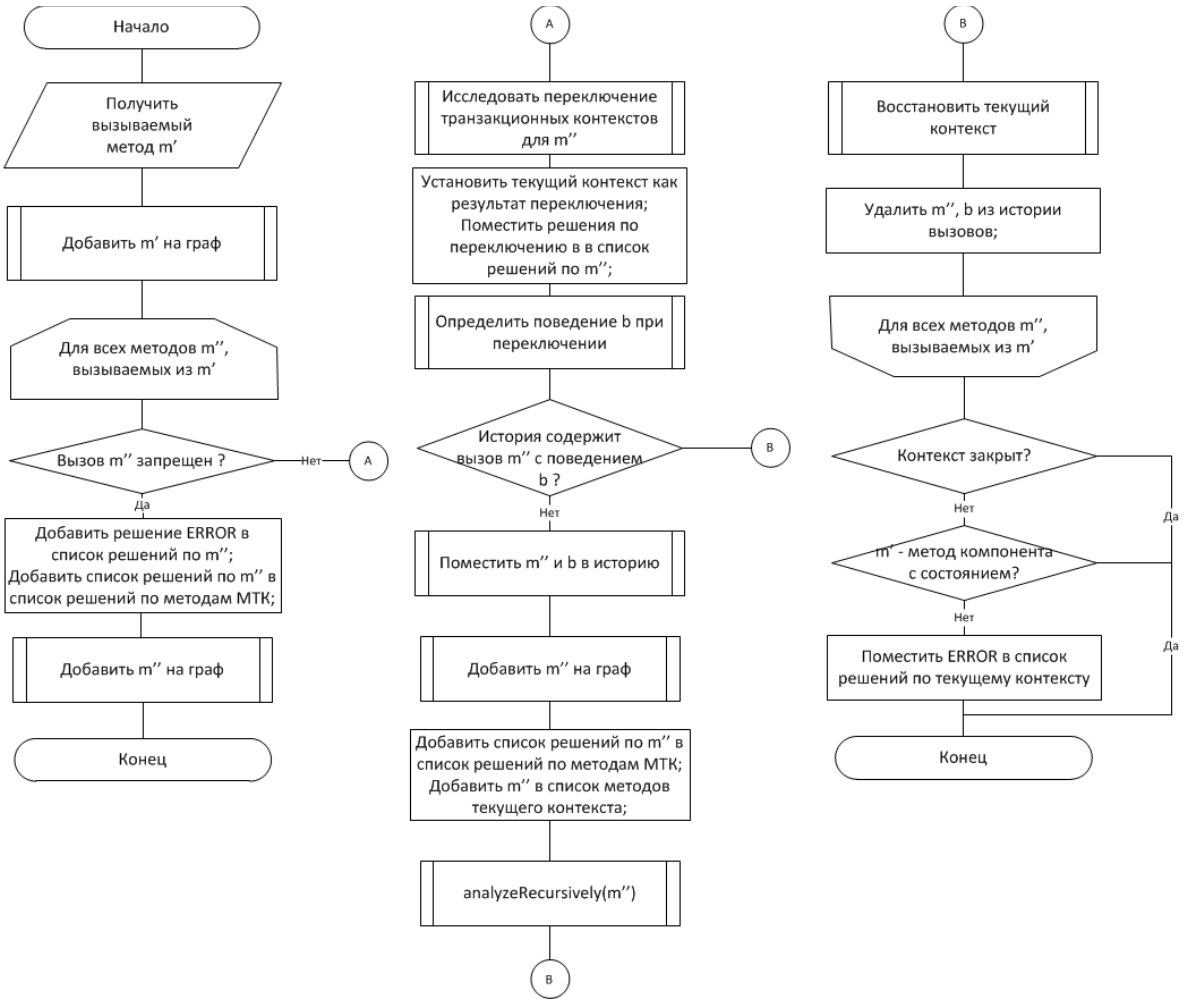

Рис. 2. Алгоритм построения модели транзакционных контекстов (МТК)

метода завершается. В противном случае в историю вызовов добавляется пара <вершина, поведение>, вершина вызываемого метода добавляется в список вершин установленного транзакционного контекста и в развернутый граф вызовов, где соединяется дугой с вершиной – вызывающим методом. Если в числе решений, принятых алгоритмом по вызываемому методу, присутствует хотя бы одно решение типа ERROR, текущая точка входа помечается как завершившаяся аварийно. Затем рекурсивная процедура выполняется для вызываемого метода. Выход из рекурсии происходит при отсутствии у вызываемого метода вызываемых им методов.

При завершении обработки очередного вызываемого метода производится переключение на транзакционный контекст вызывающего метода (для декларативного управления). Проверяется, не осталось ли незакрытой транзакции у метода компонента с программным управлением транзакциями, и вызов метода удаляется из истории вызовов.

III. Прототип анализатора управления транзакциями

На основании предложенной методики анализа управления транзакциями был разработан прототип анализатора исходных кодов ПС для платформы Java EE 6.

Исследование средств статического анализа для Java-проектов показало их разделение на 2 группы: 1) готовые анализаторы, предоставляющие на выходе некоторую модель исходного кода, и 2) API для анализа, позволяющие разрабатывать собственные анализаторы, но со значительной трудоемкостью. По совокупности факторов в качестве основы для анализатора был выбран относящийся к первой группе каркас модельно-ориентированного реверс-инжиниринга MoDisco [4] как универсальное решение, позволяющее анализировать исходный и байт-код, аннотации и конфигурационные файлы.

Разрабатываемый анализатор представляет собой набор подключаемых модулей для платформы сред разработки Eclipse, что обеспечивает совместимость с каркасом MoDisco (тоже поставляемым как набор подключаемых модулей для Eclipse), а также комфортную работу пользователя. Модель исходного кода ПС, формируемая

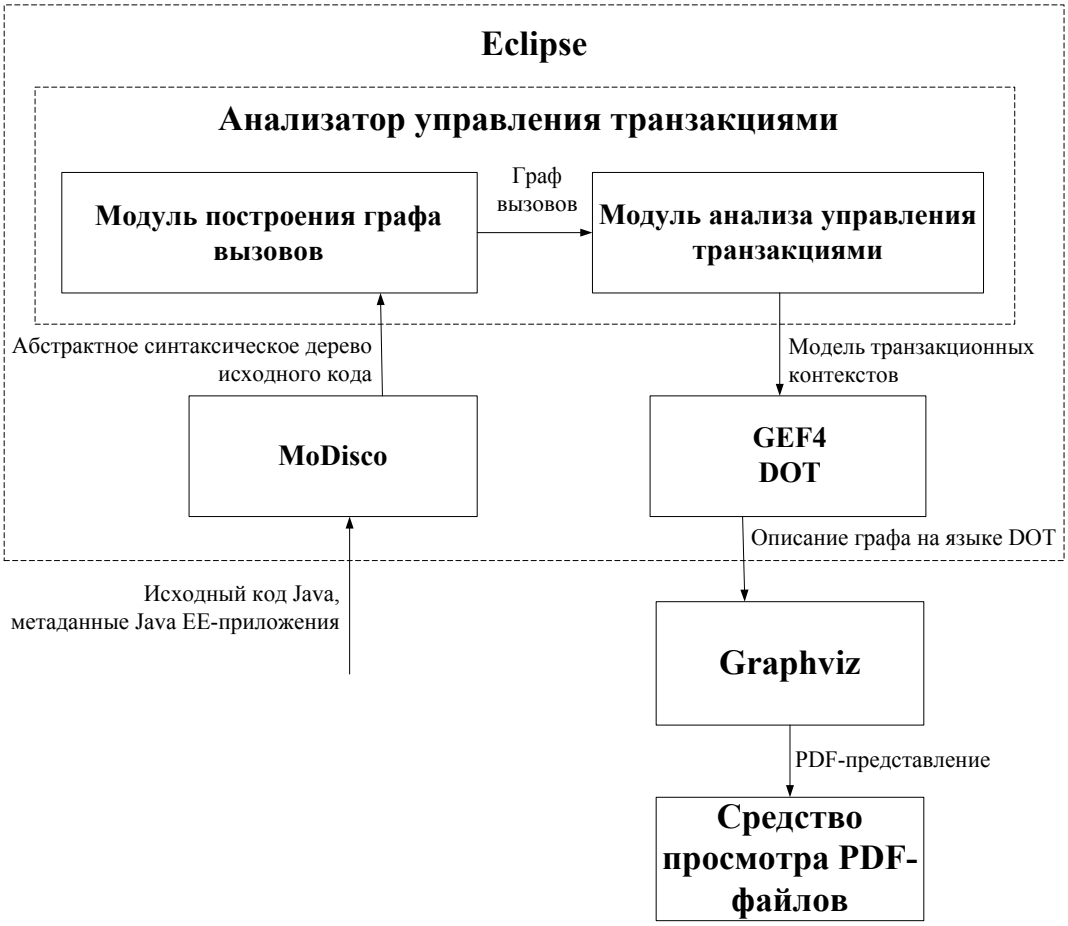

Рис. 3. Взаимодействие анализатора и внешних средств

MoDisco, поступает на вход модуля построения графа вызовов, выполняющего первый и второй этапы предложенной методики. Полученный из него модифицированный граф вызовов преобразуется в модель транзакционных контекстов модулем анализа управления транзакциями в соответствии с третьим этапом методики.

Наглядное представление модели транзакционных контекстов обеспечивает средство визуализации графов Graphviz [5]. Модуль анализа управления транзакциями сохраняет граф модели транзакционных контекстов в виде текста на языке DOT и обеспечивает его преобразование в формат PDF посредством подключаемого модуля GEF4 DOT. Общая схема взаимодействия модулей анализатора и внешних средств представлена на рис. 3.

Модель транзакционных контекстов визуализируется в виде графа (развернутого графа вызовов), при этом вершины, входящие в один транзакционный контекст, размещаются в подграфе типа «кластер» и, как следствие, входят в общий ограничивающий прямоугольник (рис. 4). В вершинах, соответствующих методам EJB-компонентов, выводятся данные о типе управления транзакциями и действующем транзакционном атрибуте.

На рис. 5 показан обнаруженный цикл на графе: вызов метода postProcess помечен как циклический после того, как повторно было обнаружено вхождение с поведением «распространение текущего контекста на вызываемый метод». Также на рис. 5 представлен избыточный контекст транзакции, не содержащий операций с БД.

Случай обнаружения ошибки (вызов метода javax.transaction.UserTransaction::commit() вне контекста транзакции) представлен на рис. 6. Точка входа помечена как завершившаяся аварийно, методы, которым вследствие возникновения системного исключения не будет передано управление, помечены как недостижимые.

IV. Обзор существующих работ

Вопросы анализа управления транзакциями в основном упоминаются в рамках задачи мониторинга производительности сложных ПС.

Статический анализ является основой предложенного в [6] подхода к повышению производительности Java EE-сервера приложений путем исключения лишних обращений из представлений к EJB-контейнеру в части управления транзакциями и обеспечения безопасности. Эти оптимизации выполняются во время генерации представления компонента при развертывании ПС на основании статического анализа байт-кода. Анализ ведется на частном графе вызовов (для одного конкретного метода), причем порядок вызова методов не важен, а для полиморфных методов выполняется связывание с

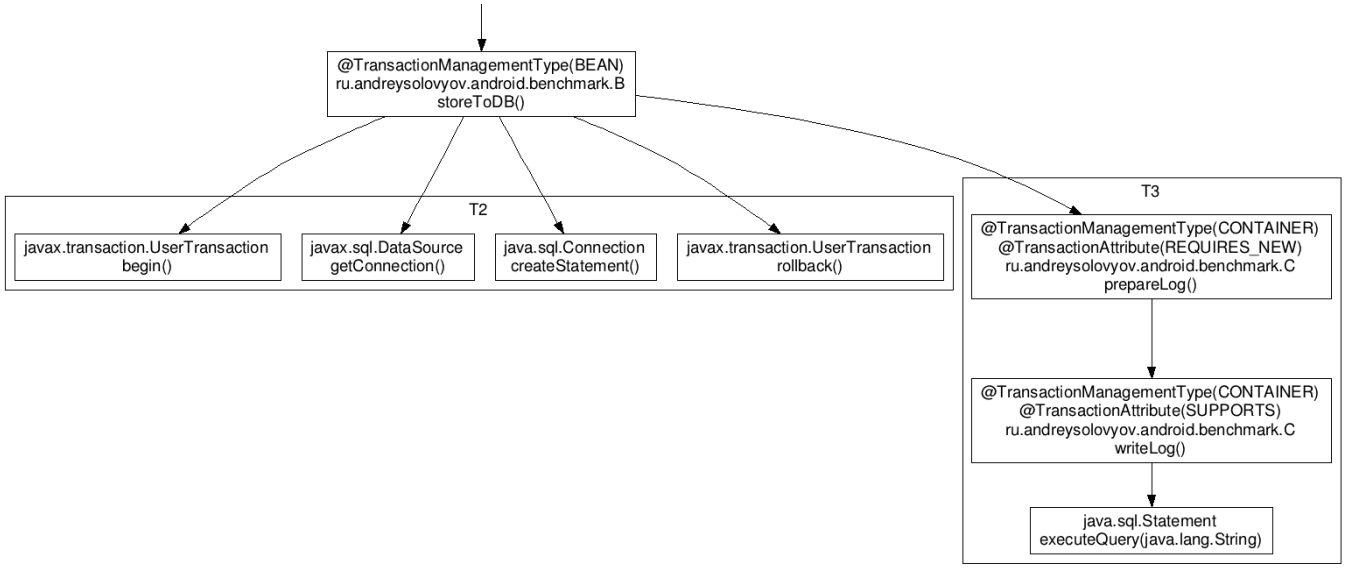

Рис. 4. Корректные транзакционные контексты

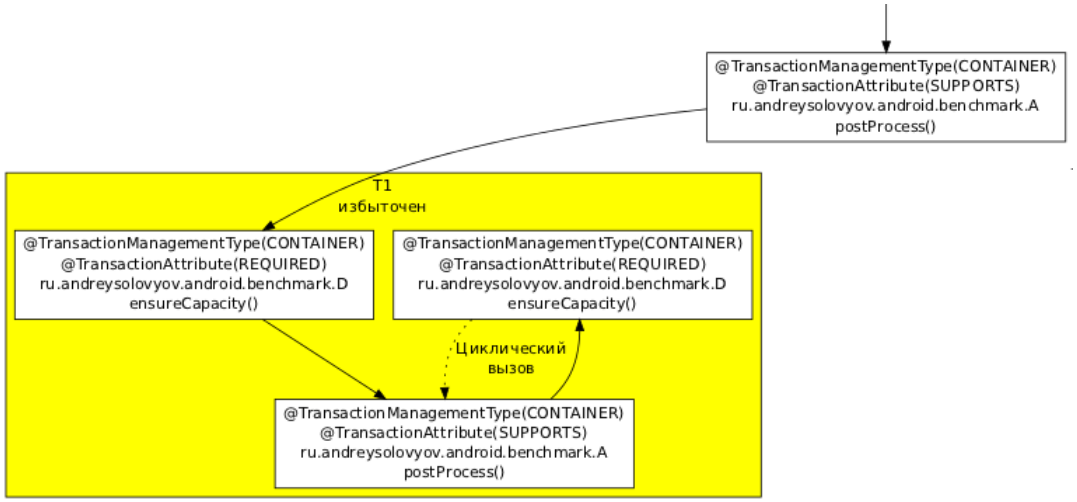

Рис. 5. Цикл и избыточный контекст

конкретной реализацией компонента на основании настроек приложения. Для принятия решения об оптимизации используется логический вывод. Предложенный в [6] самооптимизирующийся EJB-контейнер требует использования особой среды выполнения ПС, тогда как описанная в настоящей работе методика анализа не предъявляет подобных требований и применима к произвольным ПС для платформы Java EE. Другим важным отличием является позиция по «открытости мира»: в [6] предполагается, что заранее неизвестно, откуда вызывается метод компонента, и происходит это в контексте транзакции либо нет. Вследствие этого принимаемые решения являются вероятностными. В настоящей работе, наоборот, все взаимодействия, возможные с компонентами JavaEE-приложения, считаются известными и локализованными в том же Java EE-приложении. Таким образом, модель транзакционных контекстов является детерминированной.

Динамический анализ производительности Java EE-приложений вообще и управления транзакциями в частности входит в перечень функций многих коммерческих инструментальных средств, таких как HP OpenView Transaction Analyzer, dynaTrace и других. Используемые при этом методики не раскрываются.

Также упоминается комбинированный подход к анализу ПС как в коммерческих продуктах (eoSense, Yonita), так и в исследовательских проектах [7]. Так, решение eoSense основано на методике Derived Model Analysis [8]: инструменты статического анализа формируют абстрактную модель компонентов ПС, в которую во время выполнения ПС добавляются службы

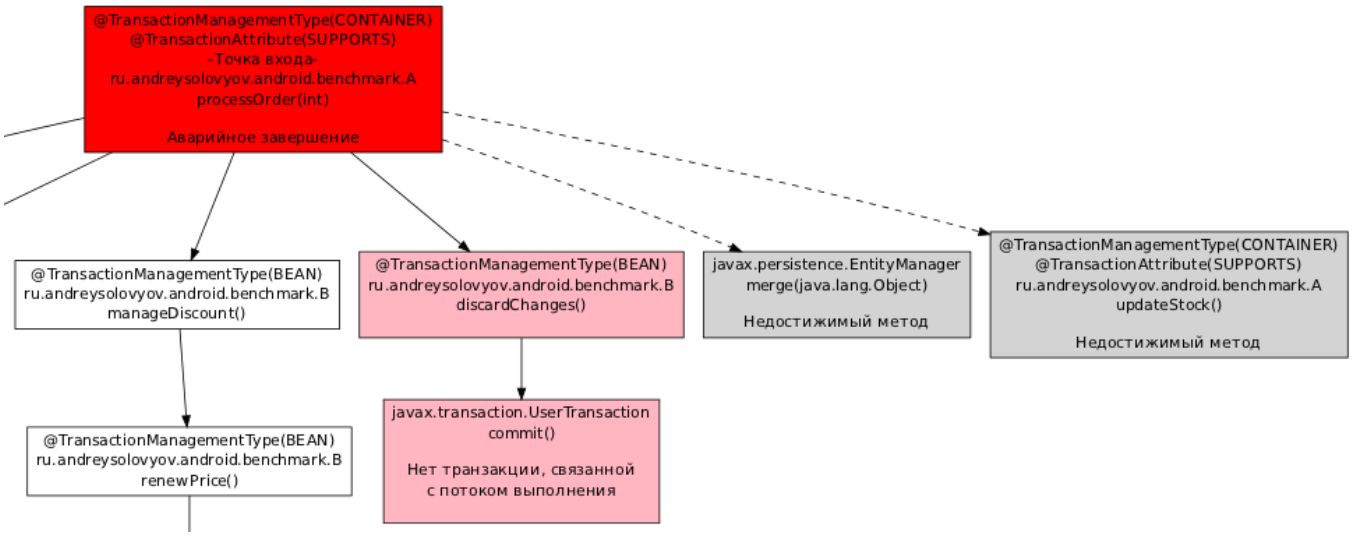

Рис. 6. Ошибка при управлении транзакциями

платформы Java EE (координатор распределенных транзакций, EJB-контейнер, подсистема обмена сообщениями и др.) и конкретные ресурсы. На последовательности событий, приводящих к изменению состояния модели, проверяется выполнение условий триггеров, что позволяет контролировать корректность функционирования ПС. В [8] особо выделяются проблемные ситуации, связанные с управлением транзакциями, и приводятся примеры триггеров для их выявления.

V. Заключение

В статье предложена методика статического анализа приложений для платформы Java EE, применение которой позволяет выявить ошибки и избыточность в управлении транзакциями на этапе разработки ПС. Данная методика реализована в прототипе анализатора управления транзакциями на базе платформы сред разработки Eclipse и каркаса реверс-инжиниринга MoDisco. Результаты анализа представляются в виде развернутого графа вызовов с привязкой вызовов к транзакционным контекстам.

Развитие анализатора управления транзакциями возможно в следующих направлениях:

1) увеличение перечня поддерживаемых возможностей платформы Java EE 7 для расширения класса анализируемых ПС, например, использование управляемых сообщениями компонентов как точек входа, учет транзакционной природы обработки сообщений в JMS-совместимых системах обмена сообщениями, поддержка инъекции зависимостей с помощью фабричных методов, анализ декларативного управления транзакциями для CDI-компонентов;

2) переход от имитации действий контейнера компонентов Java EE к построению системы логических уравнений и определению ее разрешимости, что позволит гибко настраивать правила выявления ошибок и получать решения для вызова точек входа как в неопределенном, так и в активном транзакционном контексте;

3) вывод утверждений о совместной либо раздельной обработке в транзакциях сущностей ПС и сопоставление с заранее заданными инвариантами для верификации свойств атомарности и согласованности транзакций ПС.

Список использованных источников

[1] Collecting and Analyzing Execution Time Data // Java Enterprise Performance Book (dynaTrace) URL: http://javabook.compuware.com/content/ intro/collecting-execution-time-data.aspx (дата обращения 25.04.2014).

[2] Соловьев, А.А. К вопросу декларативного управления транзакциями на платформе JAVA EE 6 / А. А. Соловьев // Исследования и наблюдения российских студентов: традиции и инновации: Материалы Всероссийской студенческой научно-практической конференции. 17 апреля 2014 г. – Мичуринск: Изд-во Мичуринского филиала Российского университета кооперации, 2013. – с. 82-84.

[3] Saks, K. Specifications of the Transaction Attributes for a Bean's Method / Kenneth Saks et al. // JSR 318: Enterprise JavaBeans, Version 3.1 EJB Core Contracts and Requirements. URL: http://download.oracle.com/otndocs/jcp /ejb-3.1-fr-eval-oth-JSpec/ (дата обращения 05.04.2014).

[4] MoDisco // MoDisco. URL: http://www.eclipse.org/MoDisco/ (дата обращения 25.05.2014).

[5] Graphviz - Graph Visualization Software // Graphviz. URL: http://www.graphviz.org/ (дата обращения 25.05.2014).

[6] Trofin, M. Call Graph-Directed Boundary Condition Analysis in Contextual Composition Frameworks / Mircea Trofin – College of Engineering, Mathematical and Physical Sciences, University College Dublin, 2007 – 119 c.

[7] Brosig, F. Automated Extraction of Palladio Component Models from Running Enterprise Java Applications / Fabian Brosig, Samuel Kounev, Klaus Krogmann. – Karlsruhe Institute of Technology (KIT), 2009.

[8] West, A. Derived Model Analysis: Detecting J2EE Problems Before They Happen / Alan West, Gordon Cruickshank // URL: http://www.oracle.com/technetwork/articles/entarch/derived-model-analysis-095992.html (дата обращения 05.09.2014).