САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Ларчик Евгений Владимирович

Реализация поддержки навигационных сервисов в Eclipse GMF

Магистерская диссертация

Допущена к защите

Зав. кафедрой:

д.ф.-м.н., профессор Терехов А.Н.

Научный руководитель:

к.ф.-м.н., доцент кафедры системного программирования

Кознов Д.В.

Рецензент:

Фесенко Т.А.

Санкт-Петербург
2012

Saint-Petersburg State University

Faculty of Mathematics and Mechanics

Department of Software Engineering

Head of the Department,

Doctor of Mathematics and Physics,

Professor Terekhov A. N.

Master's thesis

Implementation of navigation services concept for Eclipse GMF

Larchyk Yauheni

Supervisor:

Ph.D. in Mathematics and Physics

Koznov D. V.

Reviewer:

Fesenko T. A.

Saint-Petersburg 2012

Оглавление

1. Введение 4

2. Постановка задачи 8

3. Обзор технологий 9

3.1. Eclipse Modeling Project 9

3.1.1. Eclipse Modeling Framework 10

3.1.2. Graphical Modeling Framework 10

3.1.3. ATL 12

3.2. Подходы к реализации навигационных сервисов 12

3.2.1. Автоматические раскладки и проект KIELER 12

3.2.2. Генерация представления по требованию 14

3.3. Подход на основе v2v-трансформаций 16

4. Архитектура 19

4.1. Адаптация подхода на основе v2v-трансформаций для Eclipse GMF ... 19

4.2. Обзор компонент решения 20

4.3. Исполнение v2v-трансформации 24

5. Особенности реализации 27

5.1. Отсутствие достаточной документации 27

5.2. Изменение Notation-метамодели 28

5.3. Шаблон v2v-трансформации 31

5.4. Перезаписывание файла диаграммы 32

6. Апробация решения 33

7. Заключение 38

8. Список использованной литературы 40

1. Введение

Визуальное моделирование (Visual Modeling) – это метод, применяемый в разработке программного обеспечения (ПО), который: (1) использует преимущественно графовые модели визуализации ПО; (2) предлагает рассматривать ПО с различных точек зрения; (3) может применяться в разработке и эволюции ПО, а также в различных видах деятельности по его созданию [1]. Наиболее известным и популярным языком визуального моделирования является UML [12]. Стандарт UML был создан в конце 1990-х годов и с тех пор активно развивается. Именно после его появления визуальное моделирование, ранее недостаточно распространённое из-за отсутствия единой общепринятой нотации, стало неотъемлемой частью цикла разработки ПО. В настоящее время также активно развивается DSM-подход (от англ. Domain Specific Modeling), нацеленный на создание визуальных языков и их программных реализаций для отдельных предметных областей, чаще всего в некоторой компании, отдельном крупном проекте или при разработке линейки продуктов [2]. Этот подход позволяет точнее схватывать особенности предметной области и потому часто оказывается более эффективным, чем UML и стандартные средства визуального моделирования.

Одним из факторов роста популярности DSM-подхода стало развитие фреймворков, автоматизирующих многие аспекты разработки визуальных DSM-языков, как то: разработку метамоделей для новых графических языков, генерацию кода для программного доступа к объектам модели, разработку средств сериализации/десериализации моделей, автоматическую генерацию визуальных редакторов и другие. Eclipse Modeling Project (EMP) является одним из самых успешных проектов в этой области [4]. EMP предоставляет полноценную платформу для разработки средств визуального моделирования. Основными компонентами EMP являются средства для работы с метамоделями Eclipse Modeling Framework (EMF) [13],

инструментарий для создания графических редакторов Eclipse Graphical Framework (GMF) [14], а также средства работы с языком трансформаций моделей ATL [6]. Среди других известных DSM-фреймворков следует отметить Microsoft DSL Tools [15] и MetaEdit+[16].

Разработчики DSM-языка предоставляют пользователю визуальную среду, в которой он работает, создавая, редактируя и просматривая диаграммы. В каждый момент времени диаграмма является определённым представлением модели. Пользователь редактирует, просматривает и изучает модель, работая в графическом редакторе и используя предоставленные редактором диаграммные сервисы. Простейшие диаграммные сервисы позволяют добавить элемент на диаграмму, изменить его свойства, удалить и т.д. Как правило, их применение ведёт к изменению как диаграммы, так и модели. Однако не менее важными являются *навигационные диаграммные сервисы* – частный случай диаграммных сервисов, которые не изменяют модель и предназначаются для обеспечения удобной навигации по большим моделям. Такие возможности, как отображение имеющегося в модели элемента на диаграмме, отображение связанных с ним элементов, применение фильтров к атрибутам всех элементов на диаграмме и другие могут значительно упростить работу с большими и запутанными моделями (пример запутанной модели приведен на рис. 1). Поэтому наличие "умных" и удобных навигационных сервисов становится одним из ключевых факторов удобства использования средств визуального моделирования. Несмотря на это, на данный момент не существует общепринятого подхода к реализации навигационных сервисов, и, в целом, в существующих визуальных редакторах очевиден недостаток таких сервисов. Кроме того, поддержка навигационных сервисов отсутствует в DSM-фреймворках, что заметно усложняет их разработку.

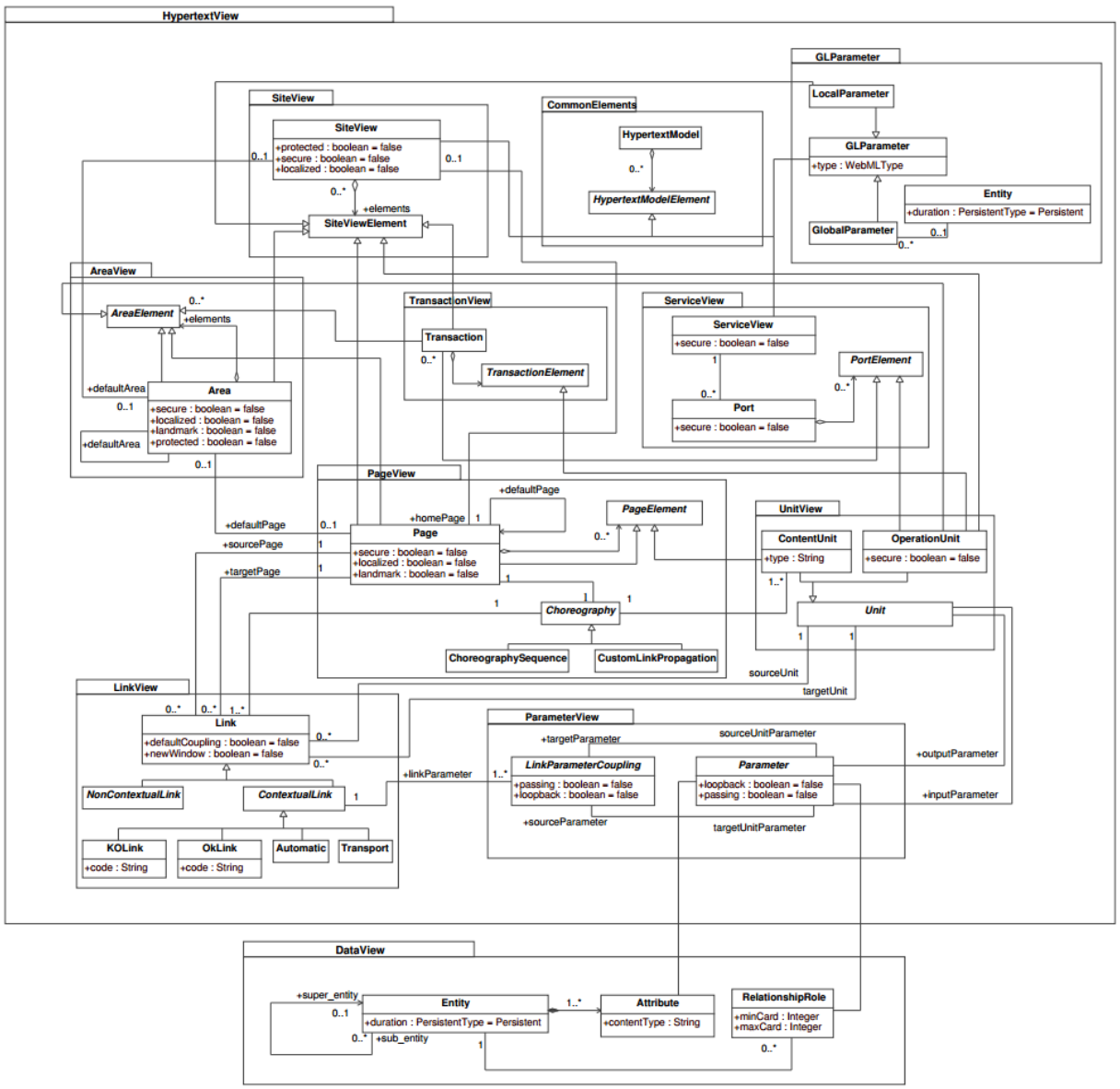

Рис. 1. Фрагмент метамодели языка WebML из работы [11]

В 2011 году в СПбГУ на семинаре по визуальному моделированию под руководством Дмитрия Кознова был разработан подход к реализации навигационных сервисов на основе концепции view-to-view-трансформаций (v2v-трансформаций) [3]. Этот подход наряду с метамоделью предметной области (Domain metamodel) рассматривает метамодель представления (View Metamodel). Модель представления (View Model) является экземпляром этой метамодели, отображающим текущее состояние диаграммы: графические объекты, находящиеся в данный момент на диаграмме, и связи между ними. V2v-трансформации действуют на модель представления, переводя её из одного состояния в другое: модель предметной области при этом не

меняется. Простейшим примером v2v-трансформации является загрузка/выгрузка элемента модели на диаграмму. Более интересный пример приведён на рис. 2: результатом применения трансформации к классу с рис. 2, а) является добавление на диаграмму атрибутов и операций для класса F – рис. 2, б).

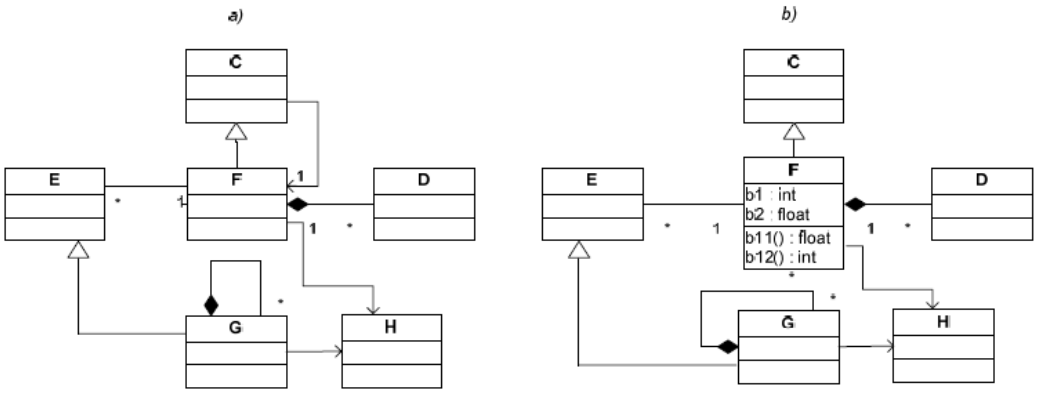

Рис. 2. Пример v2v-трансформации. Рис. взят из работы [3]

Фактически, любому навигационному сервису может быть сопоставлена некоторая v2v-трансформация. Но на примере трансформации из рис. 2 мы видим, что после исполнения v2v-трансформации необходимо выполнить раскладку элементов на диаграмме, так как некоторые ее элементы изменили свой размер, другие – местоположение, и, наконец, на диаграмму могли добавиться новые элементы. Очевидным способом решения данной проблемы является применение алгоритмов автоматической раскладки [7]. Таким образом, в рамках предложенного подхода навигационный сервис есть пара: v2v-трансформация плюс алгоритм раскладки, применяемый к её результату.

2. Постановка задачи

Целью данной магистерской работы является реализация поддержки навигационных сервисов для фреймворка Eclipse GMF. Для её решения были поставлены следующие задачи:

1) интегрировать в Eclipse GMF подход, основанный на v2v-трансформациях;

2) произвести апробацию решения на примере создания визуального редактора.

3. Обзор технологий

3.1. Eclipse Modeling Project

Eclipse Modeling Project (EMP) – один из наиболее мощных и перспективных фреймворков для разработки DSM-решений [4, 5]. Этот фреймворк создан на основе платформы Eclipse; в состав EMP входят несколько десятков проектов с открытым кодом, направленных на реализацию различных идей в области визуального моделирования (рис. 3). Фреймворк включает в себя инструментарий для разработки и визуализации моделей, генерации по модели кода на различных языках программирования и многое другое.

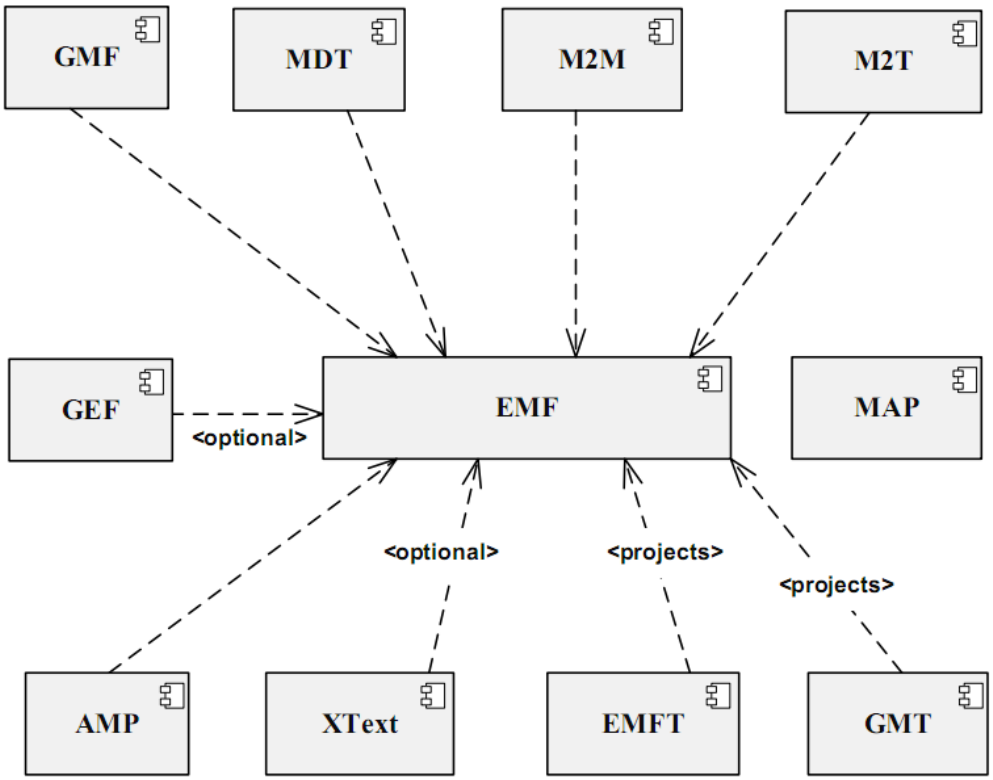

Рис. 3. СхемаEclipse Modeling Project. Рис. взят из работы [5]

Наиболее интересными в рамках данной работы являются проекты Eclipse Modeling Framework (EMF, инструментарий для создания и работы с метамоделями) [13], Graphical Modeling Framework (GMF, инструментарий для создания визуальных языков) [14], а также проект ATL (среда для

исполнения трансформаций моделей на основе одноимённого языка, подпроект проекта M2M) [6].

3.1.1. Eclipse Modeling Framework

EMF является ключевым проектом в EMP, он служит основой для многих других проектов в рамках платформы Eclipse (в том числе для проекта Eclipse GMF). Проект EMF предлагает полноценный инструментарий для разработки метамоделей и поддерживает работу с моделями, соответствующими этим метамоделям. EMF реализует собственный язык задания метамоделей – Ecore. Любому элементу Ecore-метамодели в EMF соответствует Java-класс, реализующий соответствующий интерфейс, задающий методы для установки и чтения свойств элементов. EMF позволяет задавать Ecore-метамодель множеством различных способов, включая XMI-описание, XML-схему, аннотированный Java-код, UML-модели и т.д.

3.1.2. Graphical Modeling Framework

Задачей проекта GMF является предоставление разработчикам высокоуровневых возможностей автоматической генерации графических редакторов на основе заданного множества моделей. GMF предлагает задать следующие модели:

- доменную модель (Domain Model) – описание метамодели нового визуального языка;
- графическую модель (Graphical Model) – описание графических элементов редактора;
- модель инструментария (Tooling Model) – описание вспомогательных средств работы с графическими элементами на диаграмме (в том числе задание панели инструментов);
- модель сопоставления (Mapping Model) – описание связей между тремя первыми тремя моделями;

- модель генератора – задание свойств генерации исходного кода генератора [5].

Процесс создания редактора изображён на рис. 4. В качестве графического бэкенда GMF использует Graphical Editing Framework (GEF).

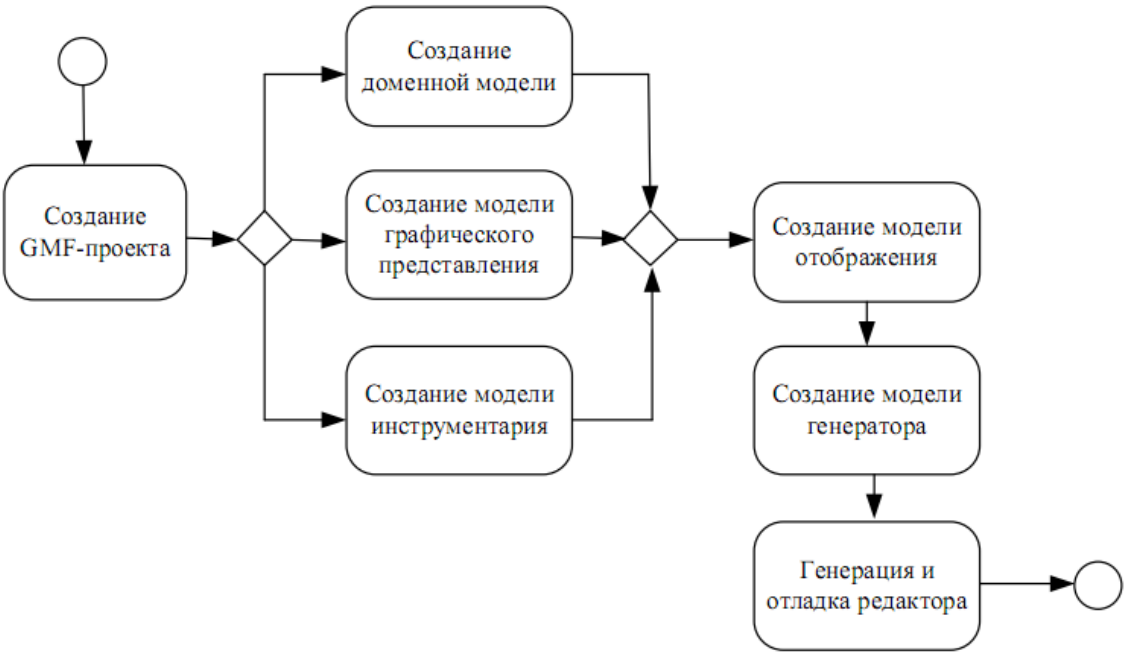

Рис. 4. Процесс создания редактора в Eclipse GMF. Рис. взят из работы [5]

GMF является мощным средством для создания графических редакторов и активно используется в контексте DSM-подхода. Примером промышленного использования данного фреймворка может служить продукт Appollo for Eclipse[1].

Следует отметить, что GMF может быть с лёгкостью расширен, поскольку имеет открытый исходных код и простую модульную архитектуру. Это делает его по-настоящему привлекательным для экспериментирования с новыми технологиями визуального моделирования.

[1] http://www.gentleware.com/apollo.html

3.1.3. ATL

Язык описания трансформаций ATL (Atlas Transformation Language) разрабатывался как альтернатива стандарту QVT[2], разработанному консорциумом OMG[3] [6]. Трансформация на языке ATL состоит из правил, которые определяют, как элементы исходной модели находятся и изменяются для получения элементов целевой модели. Язык является одновременно и декларативным, и императивным: предпочитаемым стилем написания трансформаций является декларативным (если сопоставления элементов входной и выходной модели описываются просто), однако в синтаксисе языка присутствуют также и императивные конструкции (если сопоставление элементов слишком сложно, чтобы быть описанным декларативно).

Проект ATL предоставляет среду для написания и запуска трансформаций на языке ATL. Редактор поддерживает подсветку синтаксиса, имеется встроенный отладчик. Проект активно развивается и используется благодаря тому, что ориентирован на EMF и Ecore.

3.2. Подходы к реализации навигационных сервисов

3.2.1. Автоматические раскладки и проект KIELER

Одним из способов реализации навигационных сервисов является использование автоматических раскладок элементов диаграммы. Под термином «автоматическая раскладка» мы понимаем процесс автоматического перераспределения элементов на диаграмме в соответствии с некоторым алгоритмом после некоторого события (trigger). Варианты автоматической раскладки могут быть самые разные: повторная раскладка элементов может выполняться после любого изменения диаграммы либо по запросу от пользователя; алгоритмы раскладки могут быть как устойчивыми

[2] http://en.wikipedia.org/wiki/QVT
[3] http://www.omg.org/

(небольшие изменения диаграммы приводят к небольшим изменениям после применения алгоритма раскладки), так и неустойчивыми (результат алгоритма не зависит от того, как именно расположены элементы на диаграмме до его запуска). В действительности, устойчивые алгоритмы должны применяться в то время, когда диаграмма активно редактируется, чтобы элементы диаграммы не терялись из вида пользователя, тогда как неустойчивые алгоритмы помогают в работе с уже готовой диаграммой, предоставляя различные варианты расположения элементов. Всё это избавляет пользователя от рутинной работы по перемещению старых элементов диаграммы при добавлении нового, упрощает поиск наилучшего расположения элементов на диаграмме. Однако у данного подхода есть недостаток: он не подразумевает изменения количественного состава элементов на диаграмме. Это приводит к тому, что зачастую при применении автоматической раскладки большая и запутанная диаграмма остаётся такой же большой и становится лишь чуть менее запутанной. Поэтому само по себе наличие автоматических раскладок не решает проблему навигационных сервисов для больших диаграмм, но их использование в более сложном и совершенном подходе может быть вполне оправданным.

Проект KIELER (Kiel Integrated Environment for Layout Eclipse Rich Client), стартовавший в 2004 году в институте информатике города Киль в Германии, является исследовательским проектом, основная цель которого состоит в создании инструментария для применении алгоритмов автоматической раскладки к графическим компонентам диаграмм [7]. Целевая платформа проекта – Eclipse. Фреймворк KIELER представляет собой множество Ecplipse-плагинов с открытым кодом, которые интегрированы с основными проектами Eclipse Modeling Project, в том числе EMF и GMF [8]. Наиболее полезным с точки зрения автоматических раскладок является плагин KIML (Kieler Infratsructure for MetaLayout), который выполняет следующие задачи.

- Извлекает графовую структуру графических объектов диаграммы в специально разработанную структуру KGraph. С этой структурой работают все алгоритмы раскладки.
- Предоставляет возможность применения результатов раскладки обратно к диаграмме, следуя стандартному для GEF шаблону. Соответственно, раскладки KIELER могут быть использованы в любом GMF-редакторе, поскольку они используют GEF в качестве графического бэкенда.

В большинстве случаев при создании GMF-редактора KIELER может быть использован без изменений. Однако он также определяет несколько точек расширения, которые могут быть использованы разработчиками редактора. Точка расширения Eclipse *layoutInfo* может быть использована для задания значения по умолчанию для различных опций, например, различных алгоритмов для различных типов диаграмм. Более интересной является точка расширения *layoutProvider*, которая позволяет интегрировать в KIELER собственные алгоритмы раскладки.

3.2.2. Генерация представления по требованию

Ещё одним подходом к организации работы с большими моделями в средствах визуального моделирования является поддержка множественных диаграмм для одной и той же модели, каждая из которых есть некоторое представление модели, взгляд на неё с некоторой точки зрения. Более точно такая диаграмма представляет собой визуализированный результат применения некоторого запроса-фильтра к модели. Множество доступных фильтров задаётся создателями средства визуального моделирования в процессе разработки. В свою очередь пользователь готового решения может выбрать конкретный фильтр и получить на экране некоторое представление модели. Поэтому такой подход называют генерацией представления по требованию (On Demand View Generation). Преимущества такого подхода очевидны. Каждая из диаграмм содержит лишь некоторое подмножество

элементов исходной модели, что упрощает работу с большими моделями. Кроме того, пользователю не нужно следить за корректностью различных представлений модели и тем, как они синхронизованы между собой, поскольку все представления генерируются «на лету». Однако сгенерированные таким образом диаграммы предназначены только для просмотра и получаются весьма ограниченными в плане редактирования (зачастую сгенерированные представления нельзя редактировать вообще, а для изменения исходной модели пользователю предоставляются отдельные средства). Кроме того, для просмотра каждой отдельной диаграммы пользователю также необходимо предоставить некоторые навигационные сервисы.

В работе [9] представлена концепция реализации навигационных сервисов на основе подхода ODVG, получившая название Orthographic Service Modeling (OSM). Авторы предлагают использовать при построении различных представлений модели метафору ортогонального проектирования в многомерном пространстве. При этом под измерением понимается некоторый конечный набор значений некоторого свойства моделируемой системы. Рассматриваются три типа измерений: статические (с фиксированных числом элементов), динамические (число элементов определяется просматриваемой моделью) и смешанные (имеют как фиксированные, так и зависимые от модели элементы). Конкретное представление получается выбором некоторого элемента для каждого измерения и, фактически, представляется собой единичный куб в многомерном кубе представлений (рис. 5). В качестве примеров возможных измерений авторы приводят следующие: измерение абстракции, в котором содержатся три статических элемента (Platform Independent Model, Platform Specific Model и Code), измерения инкапсуляции с двумя статическими элементами (спецификация и реализация), а также измерение сервисов, в котором находятся динамические элементы, описывающие различные

сервисы моделируемой системы.

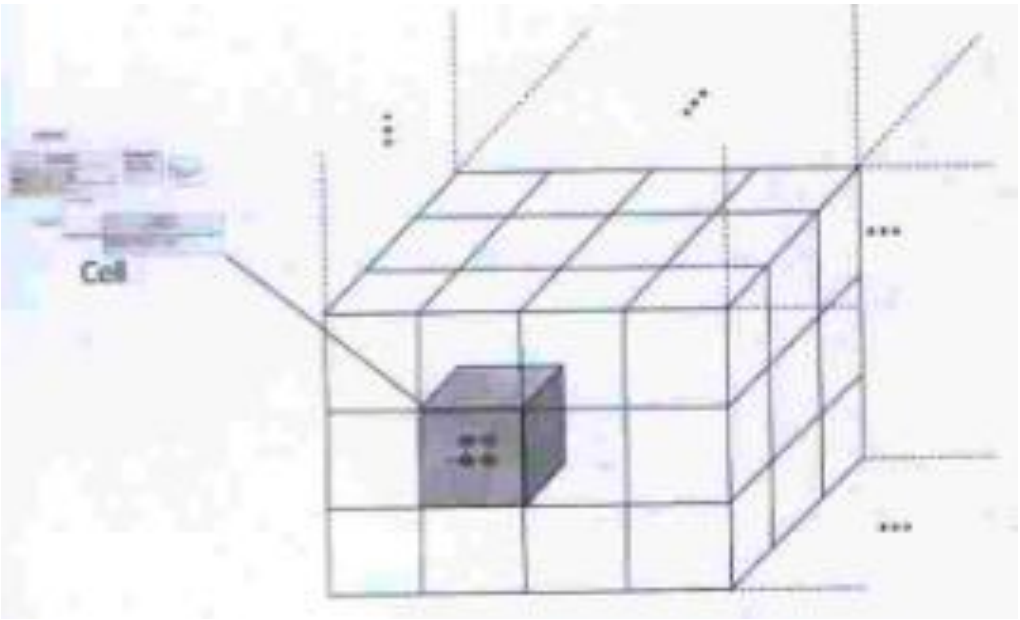

Рис.5. Куб представлений OSM. Рис. взят из работы [9]

Авторами также предлагается интуитивно понятный интерфейс для переключения представлений: слева располагается панель DimensionExplorer, где для каждого измерения находится список его элементов, между которыми можно переключаться, а справа диаграмма, получающаяся как результат выбора значений для каждого измерения. Такой интерфейс упрощает управление различными представлениями модели, поскольку нет необходимости сохранять их как отдельные диаграммы и затем упорядочивать.

В работе также упоминается прототип решения, реализованный как плагин к Eclipse, однако детали реализации, в том числе спецификация запросов к модели, к сожалению, опущены.

3.3. Подход на основе v2v-трансформаций

Подход к реализации навигационных сервисов на основе v2v-трансформаций был разработан в 2011 году в СПбГУ на семинаре по визуальному моделированию под руководством Д. В. Кознова (помимо автора данной магистерской диссертации, активными участниками этого

семинара были студенты и аспиранты математико-механического факультета СПбГУ Антон Сорокин, Николай Жуков, Елена Храмцова и Кирилл Скородумов) [3]. В основе данного подхода лежат следующие соображения.

- Диаграмма в графическом редакторе в каждый конкретный момент времени представляет собой выборку элементов доменной модели, которые отображены на ней заранее заданным способом, т.е. является некоторым представлением (View) доменной модели. Навигационные сервисы меняют диаграмму, но не меняют доменную модель, а значит, являются исключительно преобразованием текущего представления доменной модели.
- Правила, задающие, каким образом элементы доменной модели будут выглядеть на диаграмме, включаться друг в друга и соединяться друг с другом, определяются разработчиком графического редактора заранее. Эти правила явно или неявно задают так называемую метамодель представления модели (или View-метамодель). Она определяет то, какие графические элементы могут присутствовать на диаграмме и как они связаны с элементами доменной модели. Такая метамодель может быть описана стандартными средствами задания метамоделей, точно так же, как и доменная модель.
- Каждое конкретное представление модели является экземпляром View-метамодели. Соответственно, разработчики графического редактора могут реализовывать навигационные сервисы, задавая их как трансформации над View-метамоделью – v2v-трансформации. Для этого может быть использован один из стандартных языков описания трансформаций моделей.

Однако v2v-трансформация может изменить размеры некоторых элементов, что приводит к необходимости перераспределения элементов на диаграмме. Реализовать такое перераспределение в коде v2v-трансформации

не представляется возможным (вычисление новых координат для элементов – отдельная и трудоёмкая задача). В этой ситуации разумным выглядит применение автоматической раскладки к результату выполнения v2v-трансформации. Для различных навигационных сервисов могут использоваться различные алгоритмы автоматической раскладки, но все эти алгоритмы должны быть устойчивыми, поскольку их задача заключается в том, чтобы лишь слегка отредактировать результат выполнения v2v-трансформации. Таким, образом, в рамках данного подхода разработчик графического редактора задаёт навигационный сервис парой: v2v-трансформация плюс алгоритм автоматической раскладки, применяемый к её результату.

4. Архитектура

4.1. Адаптация подхода на основе v2v-трансформаций для Eclipse GMF

Перед тем, как начать интеграцию подхода к навигационным сервисам на основе v2v-трансформаций в процесс разработки графического редактора, необходимо ответить на следующие вопросы:

- каким способом разработчик редактора будет задавать v2v-трансформации;
- каким образом навигационные сервисы, полученные на основе этих v2v-трансформаций, будут вызываться конечными пользователями редактора;
- как навигационные сервисы будут исполняться.

Очевидным способом задания v2v-трансформация является использование стандартного языка описания модельных трансформаций. В качестве такого языка в проектах, построенных на платформе Eclipse, можно использовать ATL, который так же, как и Eclipse GMF, является частью Eclipse Modeling Project. Описание трансформаций в ATL происходит в терминах метамодели, поэтому для задания v2v-трансформаций разработчику должна быть доступна явно заданная View-метамодель графического редактора. В случае Eclipse GMF View-метамодель действительно доступна и является общей для всех графических редакторов, создаваемых с помощью данного фреймворка. Она описывает все доступные для использования на диаграммах графические элементы, а также их свойства (размеры, положение, границы, шрифты), поэтому эта метамодель получила название GMF Notation (или Notation-метамодель). Описание GMF Notation в терминах Ecore, а также Java-классы, сгенерированные с помощью EMF на основе этого описания, доступны в плагине *org.eclipse.gmf.notation.*

Теперь нужно определить, каким будет пользовательский интерфейс для использования навигационных сервисов на основе заданных v2v-трансформаций. Следует отметить, что такие сервисы могут быть применены как ко всей диаграмме (например, «скрыть все элементы определённого типа»), так и к конкретному элементу диаграммы (например, «скрыть конкретный элемент»). Таким образом, набор доступных пользователю навигационных сервисов на основе v2v-трансформаций в конкретный момент времени должен зависеть от того, выделен ли какой-либо элемент на диаграмме, и также от того, какой именно элемент выделен. В такой ситуации логичным выглядит решение сопоставить v2v-трансформации элементам контекстного меню. Тогда, если пользователь кликает правой кнопкой мыши по какому-либо элементу, то в контекстном меню появляется набор навигационных сервисов, связанных с данным элементом, если же пользователь кликает по свободному месту диаграммы, то строится список общедиаграммных сервисов.

Наконец, нужно определить, как именно навигационные сервисы будут исполняться. Поскольку для задания трансформаций был выбран язык ATL, то v2v-трансформации следует исполнять с помощью специального модуля, в котором будут использованы стандартные средства компиляции и запуска ATL-кода. На вход этому модулю необходимо предоставить код трансформации в виде текстового файла, а также сериализованные представления метамодели (Notation-метамодель) и исходной модели (файл диаграммы GMF-редактора).

4.2. Обзор компонент решения

Жизненный цикл графического GMF-редактора разбивается на два этапа: этап его создания (Design time) и этап его использования (Runtime). В свою очередь на этапе создания редактора можно выделить две фазы:

- спецификация редактора – разработчик описывает его в терминах доменной, графической и других моделей;
- настройки редактора – разработчик работает со сгенерированным Java-кодом, добавляя в него некие специфичные возможности, которые невозможно задать с помощью стандартных GMF-моделей.

Архитектура решения, которое было реализовано для интеграции подхода к навигационным сервисам на основе v2v-трансформаций в процесс разработки GMF-редактора, с учётом разбиения на этапы создания и использования редактора, представлена на рис.6.

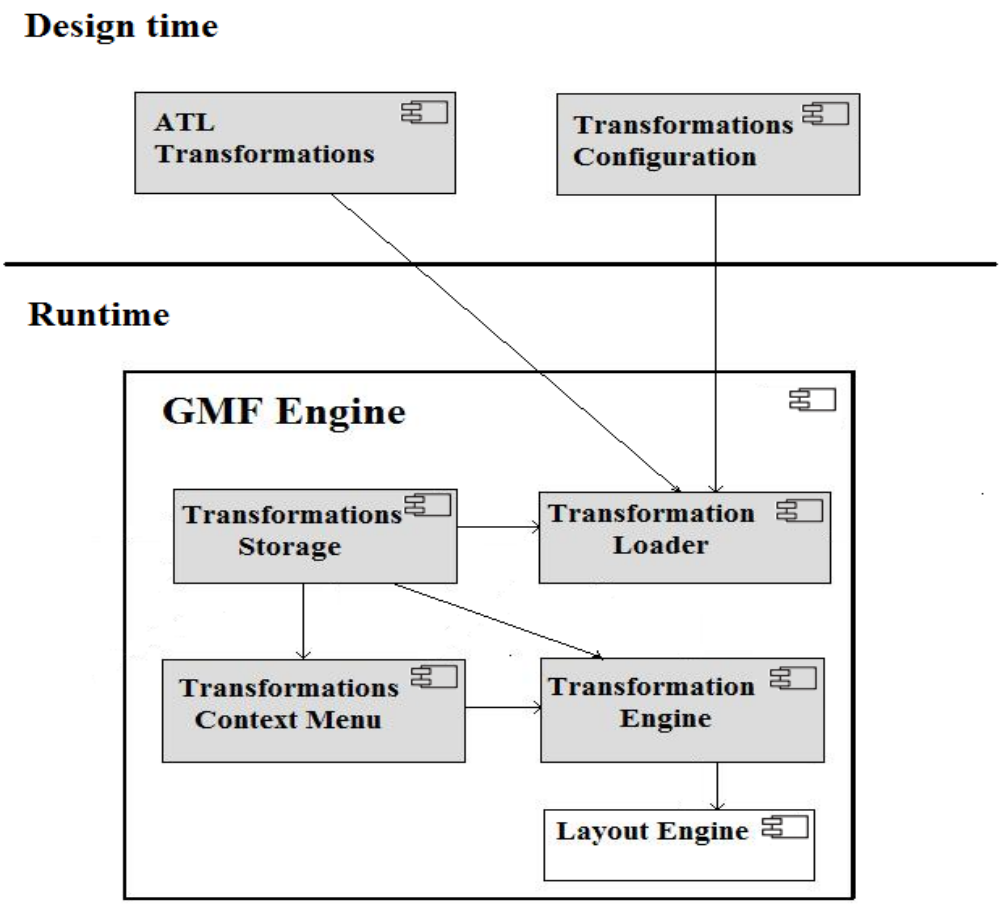

Рис. 6. Компоненты решения (серым цветом выделены компоненты, реализованные в рамках данной работы)

Разработчик GMF-редактора приступает к заданию v2v-трансформаций уже после того, как Java-код редактора был сгенерирован GMF (т.е. на фазе

настройки редактора). Для каждой трансформации он прежде всего должен написать код на языке ATL. При этом разработчик редактора может воспользоваться предоставленным ему шаблоном ATL-трансформации, в котором ему достаточно изменить несколько правил (шаблон трансформации подробно описан в разделе 5.3). Кроме этого, он должен создать определенным образом отформатированный текстовый файл с описанием дополнительной информации о трансформации, содержащий следующую информацию:

- текст, который будет показан для данной трансформации в контекстном меню;
- ссылку на файл с кодом на языке ATL;
- тип элементов доменной модели, к которым может быть применена трансформация (например, тип Class для редактора UML Class-диаграмм).

Если тип элемента в описании трансформации не указан, то считается, что трансформация применяется ко всей диаграмме. Описания трансформаций для данного редактора (файлы с ATL-кодом и файлы с дополнительной информацией) на рис. 6 представлены как компонента *ATL Transformations*[4].

Поскольку трансформации определены внутри проекта конкретного редактора, а компоненты, работающие с ними, очевидным образом являются частью библиотеки GMF, то необходим способ связывания этих двух сущностей. На рис. 6 соответствующая информация обозначена в виде компоненты *Transformations Configuration.* Данная компонента реализуется с помощью добавления следующего кода в Java-проект редактора - в конструктор класса-наследника библиотечного класса *DiagramDocumentEditor* (такой класс-наследник сгенерирован в каждом

[4] Следуя [1] в качестве компоненты на диаграмме компонент UML, которые инкапсулированы в коде и имеют стандартные интерфейсы, но также и разрозненный код, некоторые данные и пр., важные с точки зрения архитектуры и существующие на логическом уровне как одно целое (не смотря на то, что на физическом уровне они единого целого могут не составлять).

проекте по созданию GMF-редактора и имеет название *[имя проекта] + DiaramEditor*) добавляется вызов метода *initTransformationStorage()*. Параметром данного метода должен быть объект Java-класса, реализующего добавленный нами библиотечный интерфейс *ITransformationConfiguration*. Этот интерфейс определяет методы доступа к различным настройкам, необходимым для загрузки и запуска трансформаций (например, метод *getTransformationPath()* для получения пути к каталогу, в котором находятся их описания).

Остальные компоненты решения задействованы исключительно в процессе использования редактора и определены как часть *GMF Engine* – ядра технологии Eclipse GMF, в котором находится общая для всех графических редакторов функциональность.

Компонента *Transformation Loader* отвечает за загрузку v2v-трансформаций в память редактора в момент его запуска. Для каждой трансформации проверяется наличие её описания, наличие файла с кодом на языке ATL, а также корректность этого кода (с помощью компиляции).

Компонента *Transformation Storage* предназначена для хранения загруженных в память редактора трансформаций. С помощью этой компоненты можно получить информацию о конкретной трансформации, а также отфильтровать трансформации по типу элемента.

Фильтрация по типу элемента нужна для построения контекстного меню активного элемента диаграммы, которое должно содержать список пунктов, соответствующих v2v-трансформациям, применимым к этому элементу. Встраивание v2v-трансформаций в контекстное меню редактора реализовано с помощью компоненты *Transformations Context Menu.*

Компонента *Transformation Engine* является важнейшей компонентой решения. Её задача состоит в исполнении выбранной пользователем трансформации и отображении её результата на экране пользователя.

Компонента *Layout Engine* занимается раскладкой элементов диаграммы после применения v2v-трансформации с помощью технологии

KIELER. Следует отметить, что реализация данной компоненты является темой выпускной квалификационной работы бакалавра Кирилла Скородумова.

4.3. Исполнение v2v-трансформации

Остановимся подробно на том, каким образом исполняются навигационные сервисы, основанные на v2v-трансформациях. Взаимодействие конечного пользователя GMF-редактора с такими навигационными сервисами состоит из двух шагов – см. рис. 7. Сначала пользователь выделяет какой-либо элемент диаграммы и вызывает для него контекстное меню (или вызывает это меню на свободном месте диаграммы), чтобы получить список возможных действий-сервисов. В этот момент внутри редактора создаётся и обрабатывается запрос на построение контекстного меню. Этот запрос перехватывается компонентой *Transformations Context Menu*, которая на основе информации об активном элементе диаграммы запрашивает список подходящих для него трансформаций из хранилища (компонента *Transformation Storage*) и затем для каждой полученной трансформации создаёт пункт в контекстном меню.

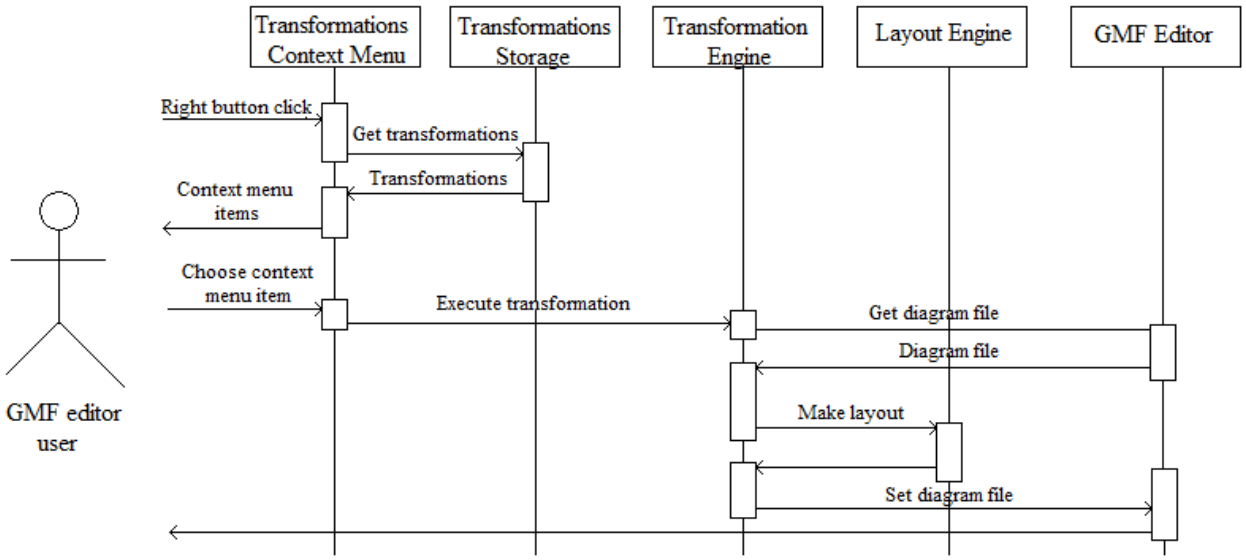

Рис. 7. Диаграмма последовательности, показывающая процесс выполнения навигационного сервиса на основе v2v-трансформации

Когда пользователь выбирает пункт контекстного меню, соответствующий какому-либо навигационному сервису, то компонента *Transformations Context Menu* формирует запрос к компоненте *Transformation Engine* на выполнение соответствующей этому навигационному сервису трансформации.

Для запуска трансформации на языке ATL необходимо скопировать файл, в котором хранится сериализованное представление View-модели, соответствующей текущему состоянию диаграммы. Однако есть сложность: сериализованное представление View-модели считывается GMF-редактором при открытии диаграммы и в дальнейшем меняется только в том случае, если пользователь сохраняет диаграмму. В остальное время меняется лишь runtime-представление диаграммы, которое представляет собой набор Java-объектов в памяти редактора, соответствующих элементам диаграммы. Таким образом, два представления View-модели в момент запуска трансформации могут быть не синхронизированы. Поэтому необходимым первым шагом при выполнении любой v2v-трансформации является синхронизация представлений View-модели, то есть сохранение диаграммы.

После того, как файл с сериализованным представление View-модели получен, компонента *Transformation Engine* приступает к запуску трансформации на языке ATL, состоящему из следующих операций:

- загрузка кода трансформации;
- компиляция кода трансформации;
- загрузка Notation-метамодели и View-модели с помощью средств ATL;
- запуск скомпилированной трансформации.

Результатом выполнения трансформации является файл, в котором находится сериализованное представление изменённой View-модели. Этот файл передаётся компоненте *Layout Engine*, задача которой – применить к

этому файлу соответствующий алгоритм автоматической раскладки.

Наконец, файл, являющийся результатом выполнения трансформации, копируется назад, что приводит к обновлению картинки на диаграмме.

5. Особенности реализации

В этом разделе описаны особенности и трудности реализации представленной выше архитектуры.

5.1. Отсутствие достаточной документации

В проекте Eclipse GMF наблюдается дефицит качественной и развёрнутой документации, и это является общей проблемой многих открытых (open source) проектов. При этом вопросы создания различных GMF-редакторов и их настройки освящены более-менее неплохо – по данной теме EMP-сообществом была накоплена неплохая база знаний в виде различных FAQ и руководств по разработке. Но в случае внутренних библиотек GMF, с которыми как раз взаимодействуют компоненты, работающие с v2v-трансформациями, полезной информации практически нет. Похожая ситуация наблюдается и для другого подпроекта Eclipse Modeling Project – ATL. В частности, автору удалось найти лишь один материал, в котором описывается, каким образом ATL-трансформации могут быть запущены из Java-кода, и который сопровождается лишь простейшим примером [17]. Как результат, понимания принципов внутренней работы Eclipse GMF, таких как сериализация/десериализации моделей и диаграмм, работа с файлами моделей, обработка действий пользователя по редактированию диаграммы, а также различных тонкостей в использовании ATL пришлось достигать лишь с помощью внимательного изучения программного кода проектов, что существенно увеличило время разработки. Однако следует отметить, что программный код Eclipse GMF и ATL, в отличие от многих других open-source проектов, хорошо структурирован, легко читается и использует хорошие мнемонические имена переменных и методов.

5.2. Изменение Notation-метамодели

Модулю ATL, отвечающему за выполнение трансформаций, помимо скомпилированного кода трансформации необходимо передать входную метамодель, входную модель, выходную метамодель и, наконец, пустую выходную модель, которая заполнится в процессе выполнения трансформации. В случае v2v-трансформаций входная и выходная метамодели совпадают, являясь Notation-метамоделью GMF, а входная модель загружается из файла, в котором хранится сериализованная диаграмма. Оказалось, что для того, чтобы такая конфигурация стала работающей, в Notation-метамодель необходимо внести изменения, описанные ниже. Важно отметить, что изменённая Notation-метамодель используется только внутри компоненты *Transformation Engine* для выполнения v2v-трансформаций на языке ATL. Notation-метамодель, используемая внутренними библиотеками GMF, остаётся прежней.

Структура файла, содержащего сохранённую диаграмму, не вполне соответствует Notation-метамодели. Это несоответствие обусловлено тем, что GMF использует собственные средства сериализации/десериализации диаграмм. А при загрузке модели из диаграммного файла средствами ATL это несоответствие приводит к ошибкам. Данная проблема может быть легко устранена, т. к. она касается всего лишь двух xml-тегов: *edges* и *children*. Тегом *edges* начинается описание любой линии, соединяющей элементы диаграммы. Таким линиям в Notation-метамодели соответствует класс *Edge*. В файле диаграммы родительским тегу *edges* является тег *notation:Diagram*, поэтому загрузчик моделей ATL предполагает, что *edges*– есть некоторый список объектов класса *Edge*, который хранится в объекте класса *Diagram*. Однако в Notation-метамодели такие списки хранятся в объектах класса *View* (описывает графические элементы диаграммы) и делятся на списки входящих линий (*targetEdges*) и исходящих (*sourceEdges*). Эти списки заполняются при загрузке модели внутренними методами GMF. Добавления свойства *edges* в

класс *Diagram* оказывается достаточным для того, чтобы ATL загрузил информацию о соединительных линиях корректно. Похожие проблемы возникают с тегом *children*, с помощью которого описывается древовидное вложение различных графических элементов диаграммы друг в друга. Загрузчик ATL предполагает, что в объекте класса *View* должен быть определён список *children*, состоящий из объектов того же класса *View*, в то время как в Notation-метамодели в классе *View* этот список имеет название *transientChildren*. Соответственно, для того, чтобы загрузчик ATL корректно загружал модель из файла диаграммы, в класс *View* необходимо также добавить свойство *children*. Автор выражает благодарность аспиранту математико-механического факультета Николаю Жукову за помощь в решении этой проблемы.

Для v2v-трансформации разработчик GMF-редактора может определить тип элементов, к которым данная трансформация может быть применена. Таким образом, некоторые v2v-трансформации запускаются не для всей диаграммы, а для некоторого её элемента. Соответственно, для исполнения таких трансформаций необходимо задать некоторый атрибут, который однозначно определял бы элемент диаграммы, и затем передать конкретное значение этого атрибута в трансформацию, как параметр. В сериализованном представлении диаграммы у каждого элемента есть уникальный атрибут *xmi:id*. Значение данного атрибута может быть также получено в runtime-представлении диаграммы для любого элемента, в том числе активного (т.е. значение этого атрибута может быть передано в ATL-трансформацию, как параметр). Однако этот атрибут является внутренним для GMF и не определён в Notation-метамодели, что делает бессмысленным его использование в ATL-коде трансформации. Чтобы решить эту проблему, в изменённую Notation-модель для класса View был добавлен атрибут *xmi_id*, значение которого всегда равно значению атрибута xmi:id и который может быть использован в ATL-трансформации. Изначально в файле диаграммы

такого атрибута, очевидно, нет, как не должно его быть и в файле-результате выполнения трансформации (иначе GMF не сможет загрузить диаграмму из этого файла). Поэтому в процесс выполнения ATL-трансформации пришлось добавить ещё два шага: изменение файла диаграммы до выполнения трансформации (для каждого элемента значение атрибута xmi:id нужно скопировать во временный атрибут xm_id) и изменения файла-результата после выполнения трансформации (для каждого элемента атрибут xmi_id должен быть удалён).

Одной из особенностей языка ATL является поддержка наследования правил. Это означает, что, если во входной метамодели определён класс A и расширяющий его класс B, то в ATL-трансформации можно определить правило для класса A и расширяющее его правило для класса B, в котором достаточно только определить, как меняются атрибуты класса B, не являющиеся атрибутами суперкласса A. При этом если правило для класса A вызвать для объекта класса B (что возможно, т.к. A суперкласс для B), то в действительности применится правило для класса B. Наследование правил является необходимым инструментом для задания трансформаций над метамоделями, в которых присутствует наследование классов, в том числе над Notation-метамоделью. Однако в Notation-метамодели присутствует также и множественное наследование (у некоторых классов есть несколько суперклассов), поддержки которого в ATL пока нет. Если быть точным, то такая функциональность реализована в виртуальной машине для запуска ATL-трансформаций EMFTVM[5], однако на данный момент данная виртуальная машина является экспериментальной и будет включена лишь в следующий релиз ATL. Как результат, текущая версия ATL не может корректно обрабатывать множественное наследование в Notation-метамодели. Таким образом, возникает необходимость избавиться от множественного наследования в Notation-метамодели. Это может быть

[5]http://wiki.eclipse.org/ATL/EMFTVM

сделано с помощью дублирования атрибутов. Например, элемент *Bounds* изначально был унаследован от элементов *Location* (с атрибутами *x* и *y*) и *Size* (с атрибутами *height* и *width*) и не имел собственных атрибутов. В изменённой метамодели элемент *Bounds* наследуется только от элемента Location, но при этом имеет атрибуты *height* и *width*.

5.3. Шаблон v2v-трансформации

Важной особенностью языка ATL является то, что трансформация описывает правила построения новой модели на основе исходной, а не правила изменения модели. Таким образом, элементы, для которых трансформация не определяет никаких подходящих правил, просто не будут перенесены в модель-результат трансформации. Это приводит к тому, что даже самая простая трансформация, которая никак не меняет исходную модель, должна содержать тождественные правила для всех типов элементов исходной модели (т.е. переводящими элементы в самих себя), которых в случае Notation-метамодели может быть несколько десятков. При этом практически любая разумная v2v-трансформация состоит из всего лишь нескольких значимых правил, которые описывают реальные изменения модели. Таким образом, большая часть правил задаваемой разработчиком редактора v2v-трансформации на языке ATL будет тождественными правилами. Очевидным желанием в такой ситуации становится желание оградить разработчика от рутинной работы по заданию таких правил. На данном этапе эта проблема решена предоставлением разработчику шаблона ATL-трансформации (файл Example.atl), где для каждого типа элемента из изменённой Notation-метамодели задано тождественное правило. Для того, чтобы задать полноценную v2v-трансформацию, пользователю остаётся только лишь изменить реализацию отдельных правил. Кроме того, такой шаблон сам по себе является хорошим примером реализации ATL-трансформации.

5.4. Перезаписывание файла диаграммы

Последним шагом выполнения v2v-трансформации является копирование файла-результата трансформации «поверх» файла, в котором хранилась исходная диаграмма. Однако недостаточно лишь перезаписать файл. Во-первых, как только GMF получает информацию о том, что файл диаграммы изменился, пользователю показывается диалоговое окно с оповещением об изменении файла и предложением перезагрузить диаграмму. В этот момент срабатывает внутренняя проверка GMF, которая для каждого файлового ресурса диаграммы (кроме непосредственно файла диаграммы, сюда включается файл, в который сериализуется доменная модель) сравнивает дату его последнего изменения с датой последнего изменения, зарегистрированного для этого ресурса внутри GMF. Таким образом, чтобы избавиться от назойливого диалогового окна, необходимо после перезаписывания файла принудительно обновить информацию обо всех файловых ресурсах, связанных с диаграммой. При этом нужно убедиться, что все потоки записи в файл диаграммы, открытые ранее, к этому моменту закрыты. Иначе в тот момент, когда эти потоки будут принудительно закрыты сборщиком мусора, дата последнего изменения файла диаграммы обновится ещё раз. Во-вторых, поскольку мы заставляем GMF думать, что в сериализованном представлении диаграммы ничего не изменилось, а на самом деле это не так, то нам приходится обновлять картинку с диаграммой самостоятельно. Для этого может быть использован метод *synchronize()*, определённый в интерфейсе *IDocumentProvider*. Класс, реализующий этот интерфейс, определён внутри Java-проекта конкретного редактора, но наличие интерфейса позволяет иметь доступ к нему изнутри библиотеки GMF.

6. Апробация решения

В рамках данной магистерской диссертации была также произведена апробация реализованной технологии. Для этого с помощью фреймворка Eclipse GMF, расширенного средствами поддержки навигационных сервисов на основеv2v-трансформаций, был создан графический редактор диаграмм классов UML . Этот редактор использует стандартную доменную модель – см. рис. 8: диаграмма (Diagram) представляет собой набор классов (Class); классы связаны между собой отношениями агрегирования (aggregations), композиции (compositions) и обобщения (generalizations); в каждом классе определено некоторое количество атрибутов (Attribute) и операций (Operations). Графическая нотация редактора также является стандартной: классы представляются прямоугольниками, разделёнными на три части (верхняя – название класса, средняя – список атрибутов, нижняя – список операций). На рис. 9 изображено, как выглядит окно редактора, а также диаграмма, созданная с его помощью. Пример, на котором происходит демонстрация, взят из книги [10].

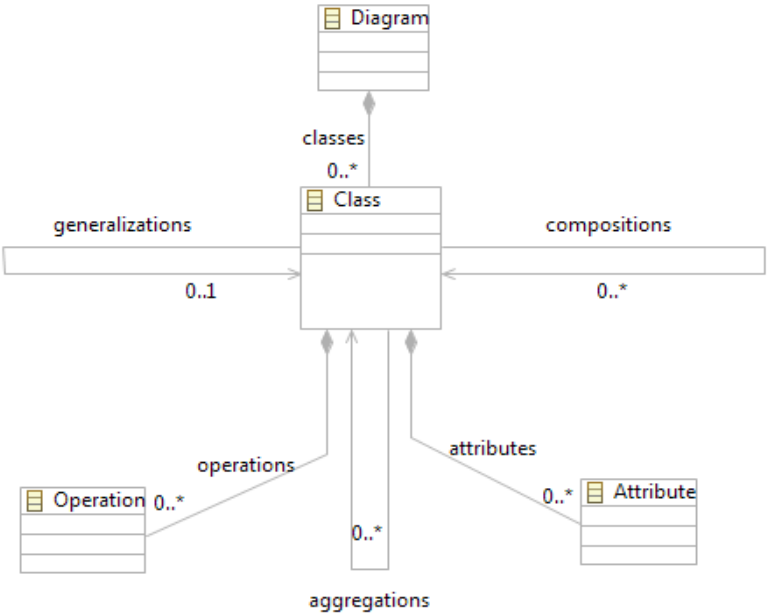

Рис. 8. Доменная модель для редактора классов UML.

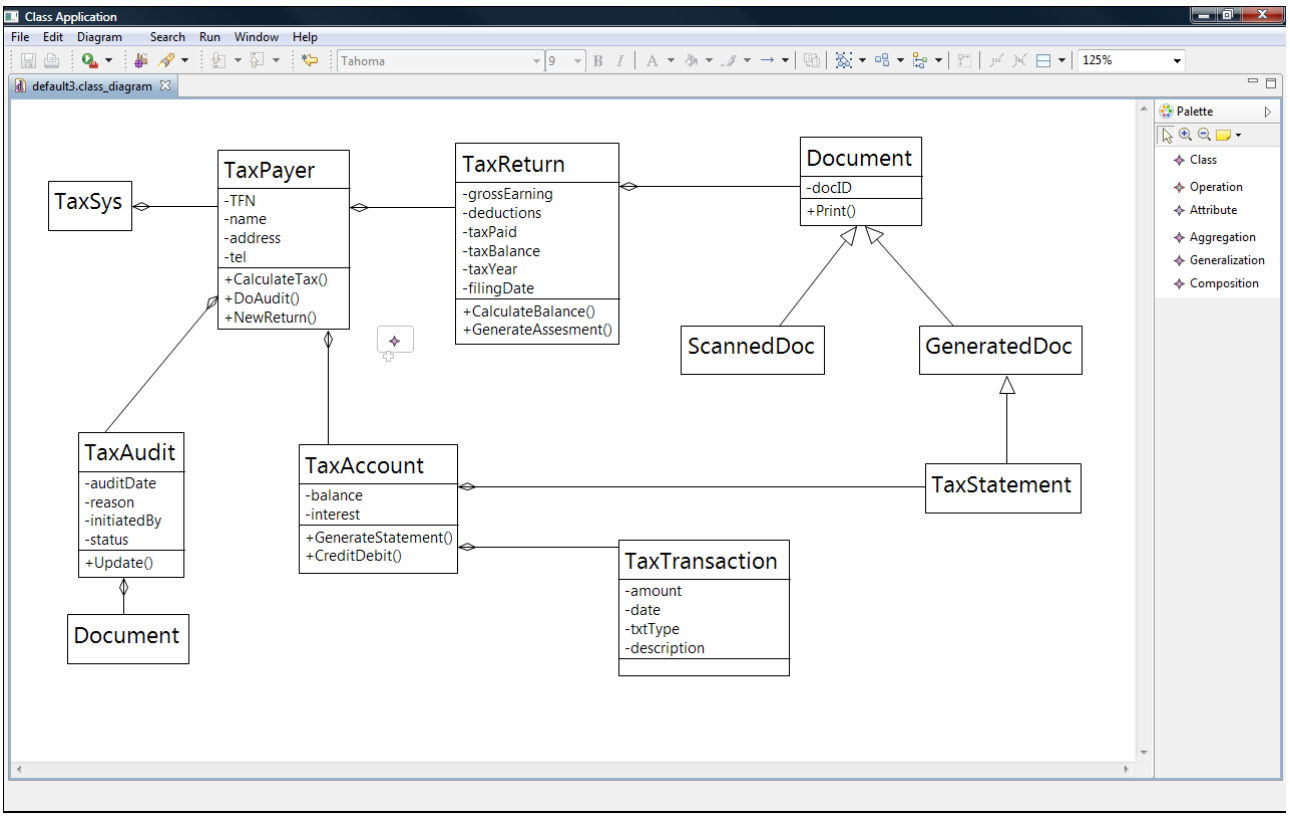

Рис. 9. Пример UML Class-диаграммы, созданной в нашем редакторе

Мы дополнили данный GMF-редактор v2v-трансформациями, реализующими следующие навигационные сервисы:

- скрыть конкретный класс на диаграмме (Hide Class);
- скрыть атрибуты и операции конкретного класса (Hide all properties);
- показать атрибуты и операции конкретного класса (Show all properties);
- скрыть все классы, связанные с некоторым классом (Hide all connected);
- показать все классы, связанные с некоторым классом (Show all connected);
- добавить на диаграмму все элементы, которые были скрыты ранее (Show all elements).

Первые пять трансформаций применяются к элементам типа Class, последняя – ко всей диаграмме. В результате в контекстное меню редактора для каждого класса появилось пять новых пунктов, соответствующих конкретным v2v-трансформациям (рис. 10).

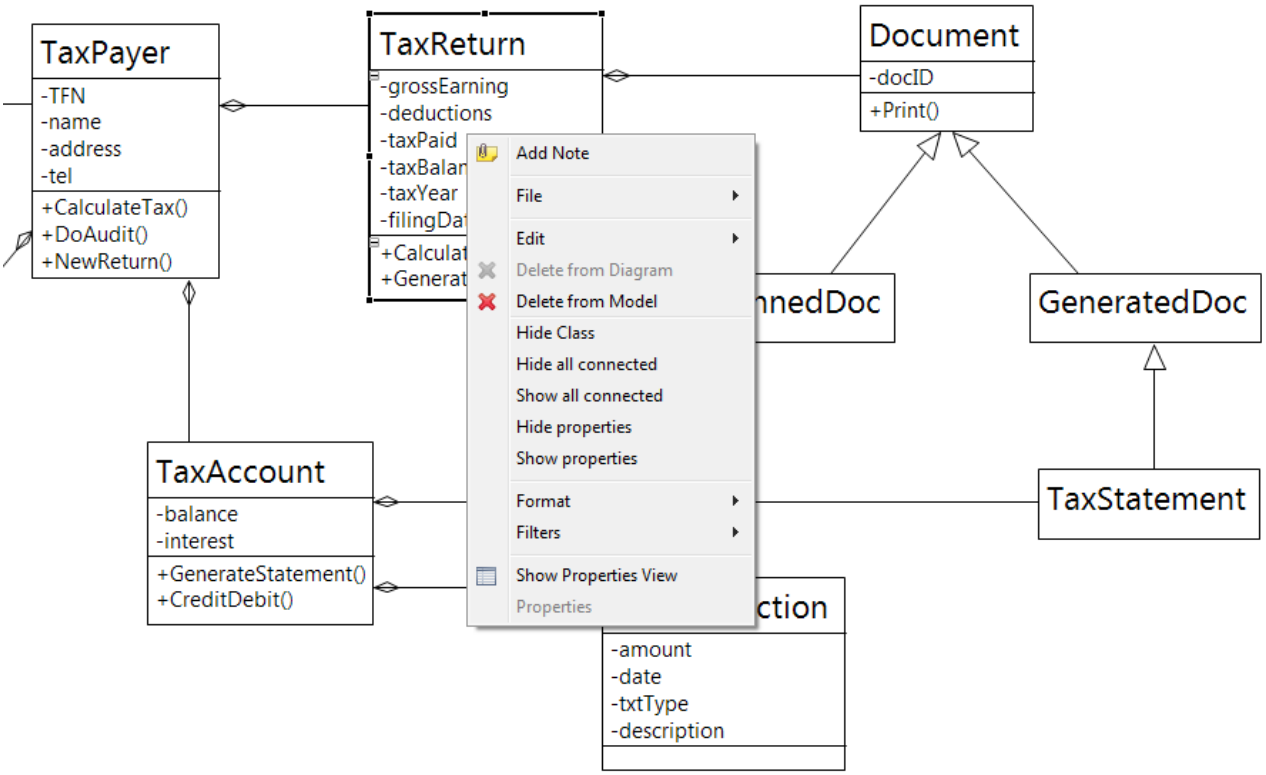

Рис. 10. Контекстное меню элемента TaxReturn

Следующие несколько рисунков иллюстрируют результаты исполнения некоторых из получившихся навигационных сервисов. Если пользователь выберет пункт контекстного меню Hide properties для класса TaxReturn, то диаграмма станет выглядеть так, как она изображена на рис. 11. Заметим, что применение обратной трансформации (Show properties) вновь вернёт пользователя к диаграмме на рис. 10.

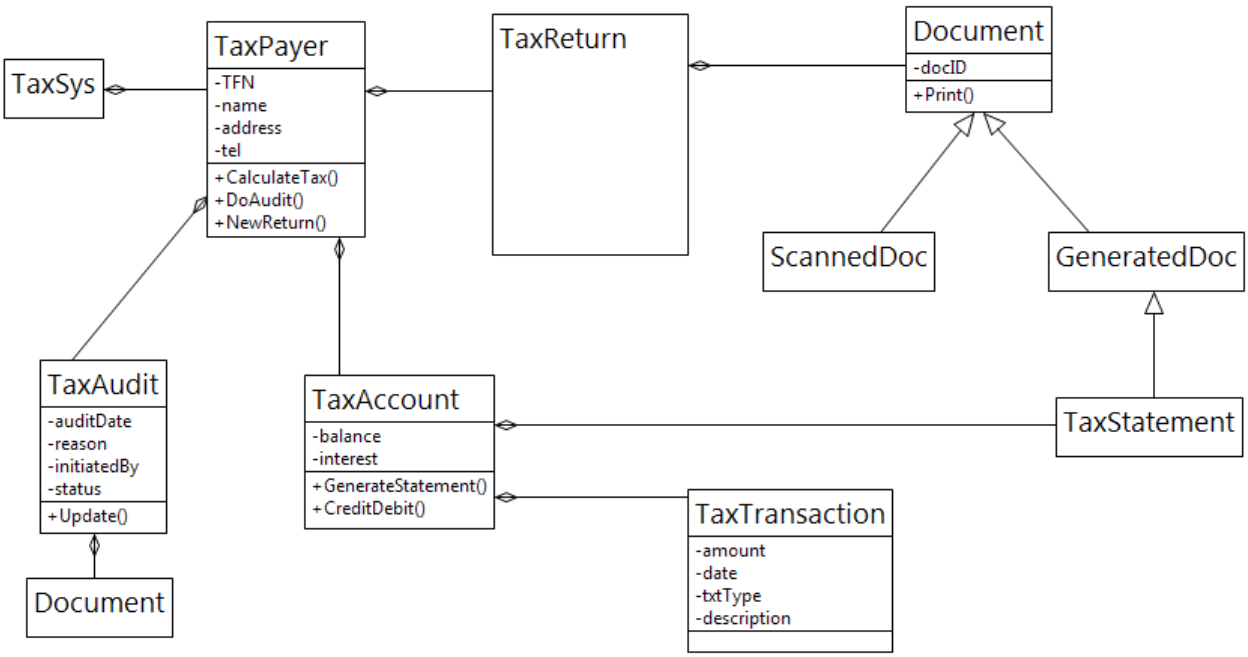

Рис. 11. Результат применения трансформации HideProperties для элемента TaxReturn

Затем, если пользователь решит последовательно скрыть элементы TaxTransaction и TaxStatement (используя пункт контекстного меню Hide Class), то диаграмма примет вид, изображённый на рис. 12. Применение обратной трансформации (Show Class) очевидным образом вернёт диаграмму к состоянию на рис. 11. А если пользователь решит вернуть все скрытые элементы на диаграмму (Show all elements), то он увидит диаграмму, изображённую на рис. 9.

Реализация GMF-редактора диаграмм классов UML с поддержкой v2v-трансформаций доказывает применимость предлагаемой технологии. Однако в процессе апробация были выявлены также некоторые недостатки решения, в частности, недостаток средств отладки v2v-трансформаций, а также необходимость для разработчика на хорошем уровне понимать принципы работы ATL и Notation-метамодель.

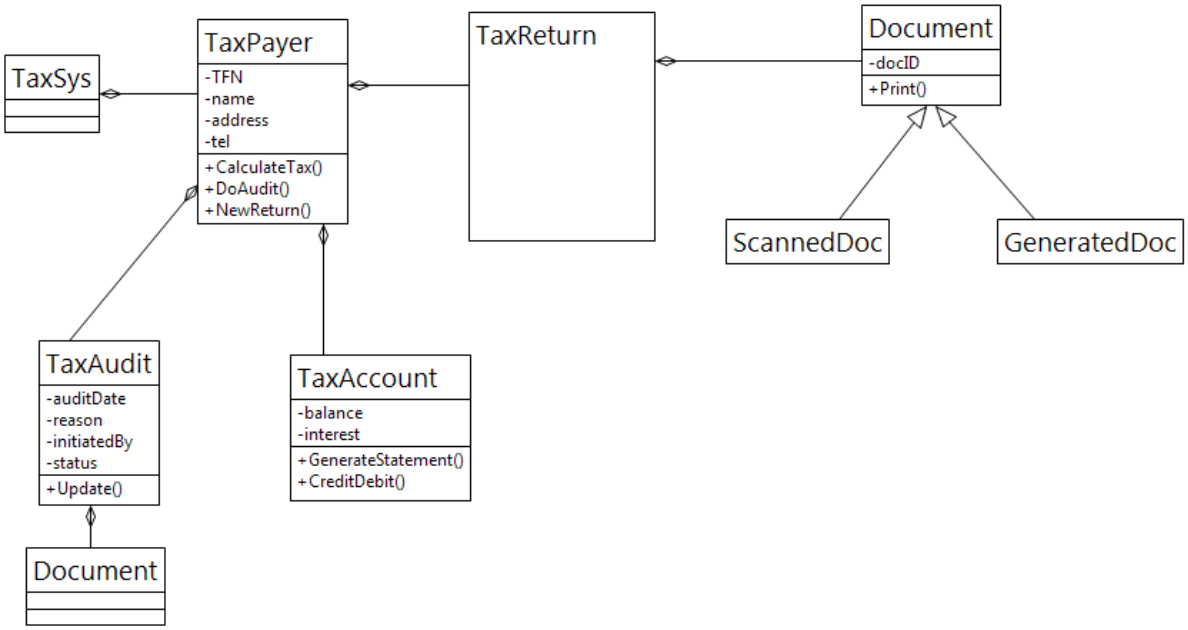

Рис. 12. Результат применения трансформации Hide Class к элементам TaxStatement и TaxTransaction

7. Заключение

В рамках данной дипломной работы были получены следующие результаты:

- разработан плагин для Eclipse GMF, реализующий поддержку навигационных сервисов, основанных на v2v-трансформациях;
- в качестве апробации решения был разработан GMF-редактор для UML Class-диаграмм, в котором с помощью трансформаций на языке ATL были реализованы следующие навигационные сервисы:
  - скрыть отдельный класс;
  - скрыть атрибуты и операции у отдельного класса и обратный к нему;
  - скрыть все классы, соединённые с данным, и обратный к нему;
  - показать все скрытые классы.

Данная разработка является первым проектом по применению на практике подхода к реализации навигационных сервисов на основе v2v-трансформаций. По его результатам можно сформулировать следующие направления дальнейшей деятельности.

- Разработать метаязык задания трансформаций, который позволит разработчику GMF-редактора задавать трансформации на основе доменной и графической моделей и который будет транслироваться в код на языке ATL, описывающий трансформацию над Notation-метамоделью. Наличие такого метаязыка позволит задавать трансформации ещё до генерации Java-кода редактора, а также избавит разработчиков от необходимости знать нюансы Notation-метамодели и уметь разрабатывать трансформации на языке ATL.
- Добавить средства генерации исполняемых трансформаций по декларативным спецификациям, включая автоматическую генерацию

всего дополнительного кода, который в данный момент вставляется в редактор GMF «вручную».

- Добавить средства отладки трансформаций.

8. Список использованной литературы

[1] Кознов Д. В. Основы визуального моделирования. М: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. 248 с.

[2] Kelly S., Tolvanen J. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press, 2008. 448 p.

[3] Кознов Д. В. О спецификации диаграммных преобразований в графических редакторах. Вестник Санкт-Петербургского университета. Сер. 10. Вып. 3. С. 100-111.

[4] Eclipse Modeling Project, http://www.eclipse.org/modeling/

[5] А.Сорокин, Д.Кознов. Обзор проекта Eclipse Modeling Project // Сб. Системное программирование./ Вып. 5, под ред. А.Н.Терехова и Д.Ю.Булычева. СПб.: Изд. СПбГУ, 2010. С. 6-31.

[6] ATLAS group LINA& INRIA. ATL: Atlas Transformation Language. User Manual. January 2005. 17 p.

[7] KIELER Eclipse project, http://www.informatik.uni-kiel.de/rtsys/kieler/

[8] Hauke Fuhrmann, Reinhard von Hanxleden. Taming graphical modeling, Proceedings of the 13th international conference on Model driven engineering languages and systems: Part I, October 03-08, 2010, Oslo, Norway. Pp. 196 - 210.

[9] Colin Atkinson, Dietmar Stoll and Christian Tunjic. IEEE, 2011 Orthographic Service Modeling. Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International. Pp. 67-70

[10] Sharam Hekmat. UML Process. Pragsoft.com, 2005. 60 p.

[11] N. Moreno, P. Fraternali and A. Vallecillo. WebML modeling in UML. IET Software 1(3), 2007. P_67-80.

[12] UML Resource Page, http://www.uml.org

[13] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/

[14] Graphical Modeling Framework, http://www.eclipse.org/modeling/gmp/

[15] Microsoft DSL Tools, http://www.microsoft.com/en-us/download/details.aspx?id=2379

[16] MetaEdit+, http://www.metacase.com/mep/

[17] ATL Howtos, http://wiki.eclipse.org/ATL_Howtos