Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Поддержка механизма рефакторингов в metaCASE-системе QReal

Курсовая работа студента 345 группы

Кузенковой Анастасии Сергеевны

Научный руководитель: Ю.В.Литвинов,

ст. преп. кафедры СП СПбГУ

Санкт-Петербург

2012

Оглавление

Введение ... 3
1 Постановка задачи ... 5
1.1 Основные задачи рефакторинга моделей ... 5
1.2 Задачи курсовой работы ... 8
2 Существующие решения ... 10
2.1 Eclipse Modeling Framework ... 10
2.2 Generic Modeling Environment ... 13
2.3 Fujaba ... 15
3 Реализация ... 18
3.1 Язык задания рефакторингов ... 19
3.2 Работа с рефакторингами ... 22
3.3 Автоматическое расположение элементов ... 24
3.4 Примеры ... 25
Заключение ... 32
Литература ... 35

Введение

Рефакторинг — широко известный способ повысить качество программного обеспечения. Рефакторинг определяют как изменение внутренней структуры системы с целью сделать её проще для понимания и внесения дальнейших изменений, при этом, не изменяя существующей функциональности. При программировании с помощью текстовых языков рефакторинг активно применяется уже много лет и является хорошо изученной областью знаний [1].

В последнее время становится популярным модельно-ориентированный подход (Model-driven engineering, MDE) к разработке ПО. В соответствии с ним создаваемая система представляется в виде набора моделей, описывающих её с различных точек зрения, и впоследствии по ним генерируется (частично или полностью) исходный код системы. Особенностью данного подхода является то, что разработчик большую часть времени работает не с кодом, а с моделями. В связи с этим возникает потребность в рассмотрении понятия рефакторинга и для визуальных моделей тоже.

Рефакторинг на уровне моделей является довольно молодым направлением исследований. Многие вопросы в

этой области остаются открытыми для экспериментов и дальнейшего изучения.

1 Постановка задачи

1.1 Основные задачи рефакторинга моделей

Рассмотрим общий сценарий рефакторинга модели. Система описана с помощью нескольких моделей, характеризующих её с разных точек зрения, включая её статическую и динамическую структуру. Часть исходного кода системы генерируется из её визуального представления, а часть, возможно, пишется вручную. Таким образом, процесс рефакторинга моделей имеет несколько основных шагов [2]:

- внести необходимые изменения в выбранную нами модель;
- синхронизировать измененную модель с её окружением (например, моделями, связанными с данной);
- заново провести генерацию участков кода, связанных с функциональностью, описываемой измененной моделью;
- согласовать рукописный код с полученным после генерации.

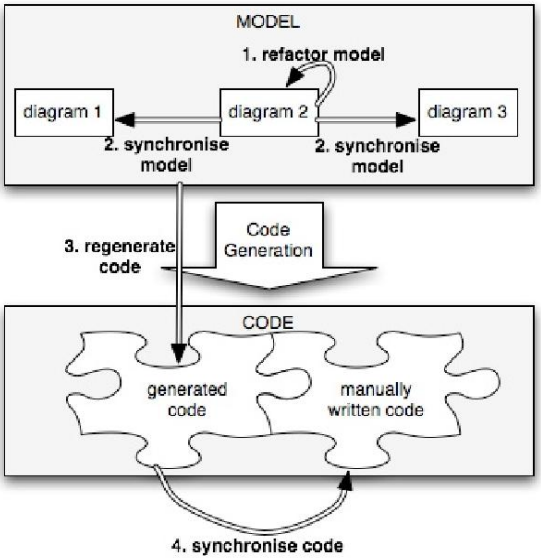

Рис. 1: Процесс рефакторинга модели

Вносить необходимые изменения в модель можно и вручную, однако часто требуется уметь формально описать изменения, которые хочется осуществить, и применить их сразу к нескольким элементам модели или к нескольким моделям в целом. На данный момент не существует универсального способа описания подобных изменений, и поставленный вопрос требует дальнейшего рассмотрения и изучения.

При обеспечении согласованности моделей до и после рефакторинга неизбежно возникает две основные проблемы. Во-первых, при модельно-ориентированном

подходе модели являются ключевыми артефактами разработки, по которым генерируются остальные — исходный код, документация, скрипты сборки и прочее. Если изменяется модель системы, то соответствующим образом должны измениться и все остальные зависящие от неё элементы. Если некоторые из них были изменены вручную, то повторная генерация должна учитывать эти изменения. Данная проблема является отдельной областью исследований и выходит за рамки этой работы.

Во-вторых, согласованность моделей может пониматься как семантическая и синтаксическая корректность в том смысле, что модель, подвергнутая рефакторингу, всё ещё соответствует некоторому языку моделирования и удовлетворяет определенным ограничениям, этим языком накладываемым. Эта проблема является одной из ключевых при осуществлении рефакторинга моделей.

Еще одной проблемой в этом направлении является то, что на данный момент не существует однозначного общепринятого определения понятия качества модели [2]. Несмотря на то, что основной целью рефакторинга является повышение качества системы, а из моделей происходит генерация исходного кода системы, модель и

код находятся на разных уровнях абстракции, а так как разработка происходит на уровне модели, то качество кода и качество модели не всегда будут иметь взаимно однозначное соответствие. В данном подходе к разработке существенными становятся такие параметры, как удобство визуального восприятия модели, её читабельность, адаптируемость и др. В контексте поставленного вопроса появляется задача выделить класс рефакторингов, которые являются применимыми и на уровне модели, и на уровне кода, и установить между ними соответствие, а также отдельно рассмотреть изменения, которые относятся непосредственно к рефакторингу моделей.

1.2 Задачи курсовой работы

В рамках данной курсовой работы основными задачами стали:

- изучение уже существующих средств рефакторингов визуальных языков;
- разработка собственного способа задания рефакторингов применимого в metaCASE-системе QReal;

- создание на его основе языка рефакторингов, создание удобного пользовательского интерфейса для работы с рефакторингами;
- выделение конкретных примеров часто используемых рефакторингов и их реализация.

2 Существующие решения

В настоящее время существует несколько инструментариев, предоставляющих возможность осуществлять интегрированную поддержку рефакторинга моделей. При этом класс моделей, для которых реализована поддержка рефакторинга в данных средствах разработки, весьма ограничен.

2.1 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) предлагает полноценный инструментарий для разработки метамоделей и поддерживает работу с моделями, соответствующими этим метамоделям [9]. Компонент EMF, отвечающий за рефакторинг моделей, называется EMF Refactor. Он активно использует модуль, отвечающий за преобразование моделей, поэтому сначала более подробно рассмотрим, как устроено EMF-преобразование модели.

EMF-преобразование модели определяется как преобразование исходной модели в целевую на основе некоторых заранее определенных правил [3]. Каждое правило имеет две основные составные части: LHS (Left-hand side) — левая часть правила и RHS (Right-hand

side) — правая часть правила. Соответствия между элементами правой и левой части правила устанавливаются с помощью номеров, которые указываются перед именем класса. LHS описывает предусловие преобразования, а RHS, соответственно, постусловие. Те элементы модели, которые присутствуют в LHS и отображаются в RHS, должны присутствовать в исходной модели и не изменятся в ходе преобразования. Элементы, которые присутствуют в LHS, но отсутствуют в RHS, удаляются в процессе преобразования. А те элементы RHS, с которыми нет соответствия элементов LHS, должны создаваться в целевой модели.

Применимость правил может быть ограничена с помощью описания дополнительных условий — NAC (Negative application condition). NAC описывает условие, при котором правило рефакторинга не применимо. Если задано несколько таких условий, то каждое из них нужно проверить перед проведением преобразования, и если хотя бы одно из них справедливо для данной модели, то преобразование не производится. По сути LHS представляет собой положительное предусловие, а NAC — отрицательное предусловие. Задание NAC является весьма полезным, если преобразование модели зависит от

некоторых свойств элементов, не входящих в область преобразования.

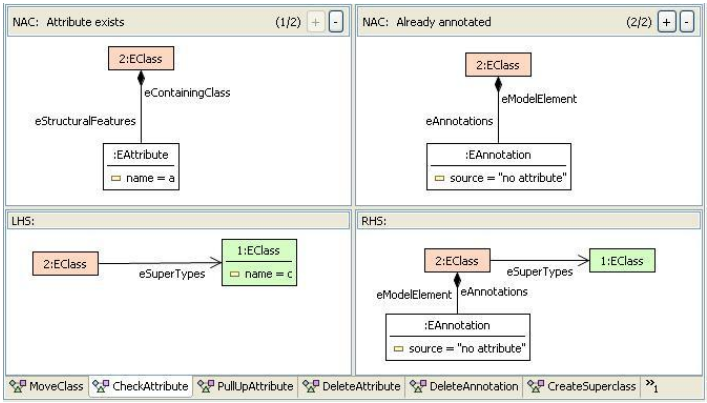

Рис. 2: Правило рефакторинга для создания атрибута

EMF Refactor состоит из двух основных модулей. Модуль генерации рефакторинга интегрирован с функциональностью каждого рефакторинга модели (описанного с помощью EMF model transformation). С помощью этого модуля созданный рефакторинг генерируется в специальный плагин среды Eclipse. Компонент EMF Tiger позволяет графически представить диаграмму трансформации модели. Сгенерированный плагин является расширением компонента EMF Refactor, и новый рефакторинг появляется в соответствующем меню системы. Для него также доступны предварительный просмотр целевой модели и отмена внесенных изменений.

2.2 Generic Modeling Environment

Generic Modeling Environment (GME) — это среда метамоделирования, базирующаяся на языке UML. Данная среда предоставляет возможности создания предметно-ориентированных моделей для широкого класса систем. The Constraint-Specification Aspect Weaver (C-SAW) — плагин GME, который представляет собой механизм для преобразования моделей [4].

C-SAW переводит исходную модель в целевую при помощи спецификаций преобразования. Данные спецификации определяют, что происходит с каждым конкретным элементов в данном преобразовании. Спецификация состоит из нескольких аспектов и стратегий. Аспекты являются отправной точкой преобразования, а стратегия определяет само преобразование. Аспекты и стратегии описываются с помощью специального встроенного языка ограничений — Embedded Constraint Language (ECL). Описанный механизм также применяется в частном случае преобразования моделей — рефакторинге моделей. Браузер рефакторингов моделей (Model refactoring browser) — плагин, встроенный в GME, который в совокупности с C-SAW предоставляет функциональность

для работы с рефакторингами моделей. Основной целью этого браузера является предоставление разработчикам интерактивного инструментария для использования и создания рефакторингов моделей. В браузере имеется некоторый заранее определенный набор рефакторингов для визуальных языков общего назначения, пользователю же предоставляется возможность для создания собственных рефакторингов как для языков общего назначения, так и для предметно-ориентированных языков.

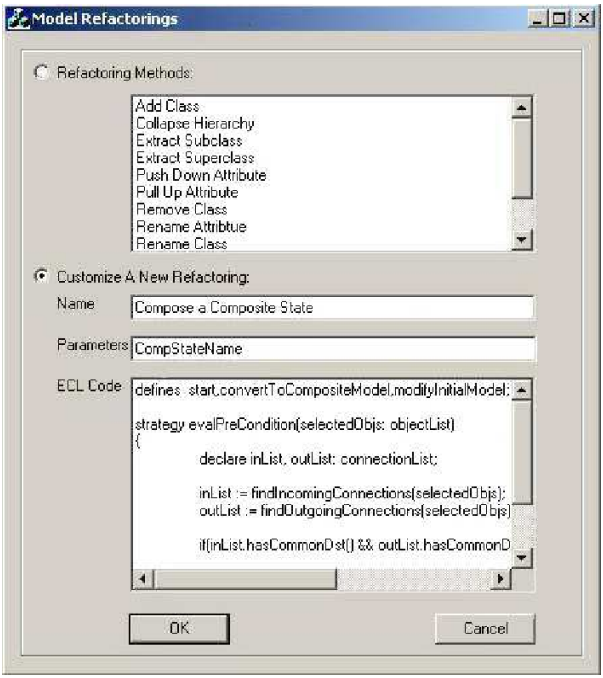

Рис. 3: Браузер для работы с рефакторингами

Стоит отметить, что главным отличием механизма рефакторингов GME от соответствующего аналога в EMF является то, что правила рефакторинга описываются с помощью текстового языка, а не визуального.

2.3 Fujaba

Fujaba — кроссплатформенная CASE-система для модельно-ориентированной разработки программного обеспечения, поддерживающая генерацию в Java [5].

Рефакторинг модели в Fujaba задается в виде кода на языке Java. В частности, это должен быть наследник специального класса AbstractRefactoring, реализующий несколько ключевых методов, таких как checkPrecondition() и perform(). При этом алгоритм рефакторинга можно визуализировать с помощью диаграммы активностей UML, в каждом из элементов-действий которой с помощью диаграммы взаимодействий задаются действия по трансформации графа модели. Для задания LHS и RHS используются стереотипы UML <<create>> и <<destroy>> [6].

Моделирование в Fujaba осуществляется с помощью диаграмм классов и некоторых поведенческих диаграмм UML, поэтому осуществляемые рефакторинги также сильно ориентированы на язык UML (например,

инкапсулирование поля или выделение метода класса в отдельный элемент). При этом и язык описания рефакторинга создан с учетом, что будет преобразовываться модель, по которой генерируется объектно-ориентированный код на Java — например, такие действия, как проверка наличия метода в классе или установка значения видимости поля вынесены в отдельные элементы языка рефакторинга [6].

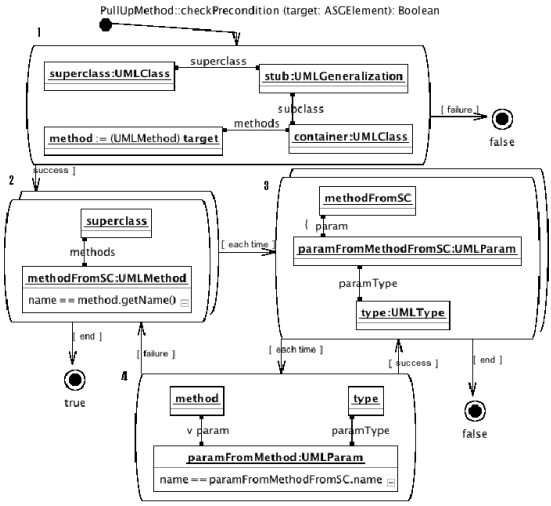

Рис. 4: Визуализация предусловия для PullUp-метода

На наш взгляд, данный подход сложно применить для более общего случая, когда целевым языком генерации не обязательно является Java или даже любой другой объектно-ориентированный язык. К тому же, созданные

диаграммы, описывающие рефакторинг, получаются довольно громоздкими и сложными для восприятия.

3 Реализация

Целью данной работы стало создание прототипа механизма рефакторингов в среде визуального программирования QReal, которая разрабатывается на кафедре системного программирования Санкт-Петербургского государственного университета с 2007 года силами студентов и преподавателей кафедры [7]. QReal имеет средства быстрого создания новых графических языков и инструментальных средств для них, благодаря чему на его основе было создано несколько CASE-систем для различных областей.

В качестве языков визуального программирования, на которых будет осуществляться апробация рефакторингов, были выбраны наиболее зрелые языки среды QReal — язык блок-схем и язык программирования роботов Lego. Последний входит в состав среды QReal:Robots [8], и позволяет задавать логику поведения робота Lego Mindstorms NXT 2.0 в виде последовательности управляющих блоков и исполнять программу на роботе, интерпретируя диаграмму и посылая команды роботу через Bluetooth или USB.

3.1 Язык задания рефакторингов

При рассмотрении существующих аналогов возникает существенный вопрос — должен ли язык описания рефакторингов быть текстовым или визуальным.

Среди преимуществ текстового языка можно выделить:

- широкие возможности задания рефакторингов — текстовый язык представляет более мощный аппарат для задания правил рефакторинга по сравнению с визуальным языком;
- компактность — как правило, для описания правила рефакторинга на текстовом языке требуется несколько строк кода, тогда как для описания аналогичного правила с помощью визуального языка требуется создать большую подробную модель.

Преимуществами визуального языка являются:

- отсутствие требования специальной подготовки — характерной чертой визуального языка является то, что он, как правило, интуитивно понятен пользователю, либо не требует значительного времени для понимания, как его использовать;

- наглядность — визуальное представления правила рефакторинга позволяет лучше понять, как будет происходить преобразование модели.

Поскольку QReal является средой визуального программирования, достижение наглядности и понятности видятся нам главной целью при разработке всех входящих в неё инструментов.

При задании модели рефакторинга может потребоваться использовать элементы того языка, для которого этот рефакторинг создается. В связи с этим одной из задач в контексте данной работы является создание языка рефакторинга, гибким образом интегрирующего в себя элементы исходного языка.

Элементы языка делятся две основные группы:

- Шаблон задания рефакторинга — элементы, описывающие каркас правила рефакторинга:
  - блок “ДО”;
  - блок “ПОСЛЕ”;
  - связь преобразования.
- Основные элементы языка рефакторинга:
  - Элемент;
  - Связь;

- Выделенный сегмент.

Блок “ДО” представляет собой предусловие — в нем находится шаблон, который мы хотим изменить. Блок “ПОСЛЕ” содержит постусловие — результат преобразования после применения данного рефакторинга. Элемент “Связь преобразования” — стрелка в направлении от блока “ДО” к блоку “ПОСЛЕ” используется исключительно для наглядности.

Блок “Элемент” обозначает любой элемент исходного языка, а блок “Связь”, соответственно, любую связь исходного языка. Перед описанием рефакторинга пользователь может выделить несколько элементов на диаграмме, которые он хочет использовать в правиле как единый блок — “Выделенный сегмент”.

При выборе языка, для диаграмм которого хочется создать рефакторинг, описанный выше язык автоматически интегрируется со всем элементами исходного и подключается к системе QReal. При этом у каждого элемента вновь созданного языка появляется числовое поле “ID”, благодаря которому можно устанавливать отношение между элементами правой и левой части правила: все элементы в левой части правила

нумеруются произвольным образом, а в правой части правила элементам (если требуется) указываются соответствующие номера.

При определении правила языка, имеется возможность использовать существующие значения свойств элементов диаграммы — для этого используется ключевое слово “EXIST”. Например, это может быть полезно, если мы хотим заменить элемент, а имя оставить прежним.

3.2 Работа с рефакторингами

Рассмотрим процесс работы с инструментами поддержки рефакторингов в QReal.

Предположим, что мы подключили нужный язык для задания рефакторингов и с помощью этого языка создали правило. Далее нужно сохранить наше правило, после чего оно появится в специальном диалоге, содержащем рефакторинги, доступные на данный момент в системе. В этом диалоговом окне слева расположен список рефакторингов, а справа соответствующее ему изображение с диаграммой выделенного правила рефакторинга.

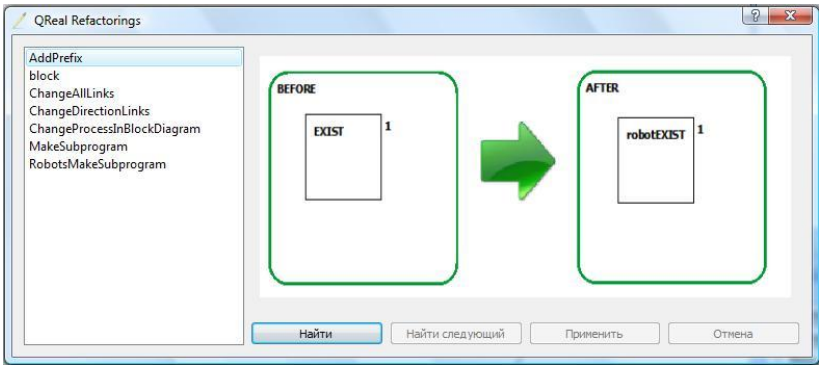

Рис. 5: Диалоговое окно для работы с рефакторингами

В этом же диалоге предоставляются следующие возможности. В начале, можно найти в текущей диаграмме место, где можно применить рефакторинг, дальше же присутствуют варианты: либо применить в найденном месте выбранное правило рефакторинга, либо найти другое место, где это правило можно применить, либо прекратить поиск. В случае если для текущей диаграммы выбранное правило неприменимо, то пользователю выдается сообщение об этом.

Для поиска на диаграмме шаблонов, содержащихся в блоке “ДО” выбранного правила рефакторинга, используется модуль преобразования графов, разрабатываемый совместно с Владимиром Поляковым и подробно описанным в его работе [10].

3.3 Автоматическое расположение элементов

Описанный выше механизм задания рефакторингов можно рассматривать как перенесение понятия рефакторинга с уровня кода на уровень модели. Но, как отмечалось выше, иногда можно выделить рефакторинги, которые имеют отношение только к моделям. На наш взгляд, к таким рефакторингам можно отнести автоматическое расположение элементов на диаграмме.

В процессе создания диаграммы пользователь может располагать элементы произвольно, в результате чего диаграммы становятся плохо читаемыми, в последствие этого сложно понять изначальную задумку автора. Поэтому поддержка средой возможности автоматически располагать элементы в соответствии с некоторыми формализованными правилами размещения элементов является существенным увеличением удобства использования данной среды. В связи с этим в системе QReal был реализован механизм автоматического расположения элементов на диаграмме “слева направо”, “справа налево”, “сверху вниз” и “снизу вверх”. Для решения этой задачи была использована программа dot

пакета утилит по автоматической визуализации графов Graphviz[1].

3.4 Примеры

Рассмотрим примеры нескольких правил рефакторинга, созданных с помощью разработанного механизма.

1. Изменение имени всех элементов диаграммы. Данное правило означает, что любому найденному элементу на диаграмме к существующему имени элемента добавится префикс «robot».

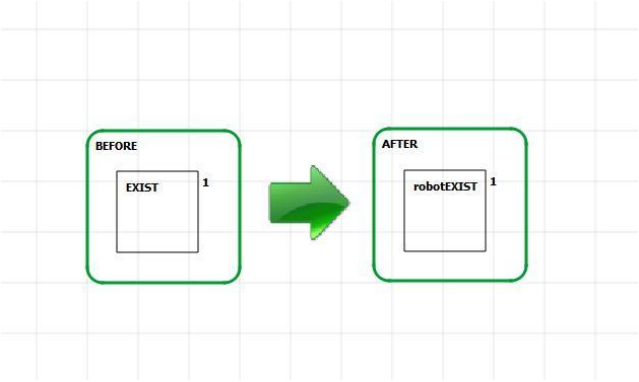

Рис. 6: Добавление префикса

2. Изменение направления всех связей на диаграмме. Согласно этому правилу осуществляется поиск двух

[1] Graph Visualization Software, http://www.graphviz.org/

любых элементов на диаграмме, связанных между собой и происходит замена направления связи между ними.

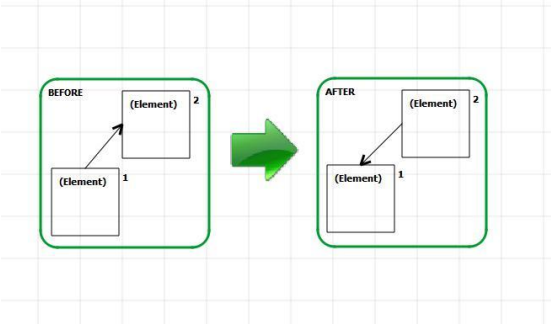

Рис. 7: Смена направления связей

3. Вынесение части диаграммы в подпрограмму. Ниже приведен пример правила, который осуществляет вынесение выделенного сегмента на отдельную диаграмму с заменой этого сегмента на указанный в правиле блок. Переход на выделенный сегмент происходит при двойном клике на заменивший его блок.

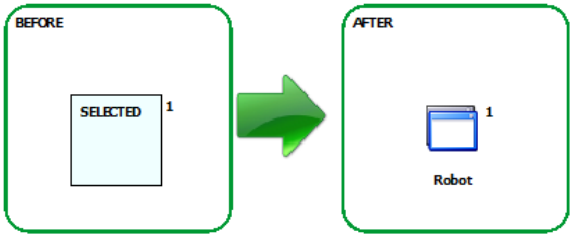

Рис. 8: Вынесение выделенного сегмента в подпрограмму

Ниже приведен пример применения правила: рассматривается программа для балансирования робота на двух колесах. Часть диаграммы, отвечающая за инициализацию программы, была выделена и вынесена на отдельную диаграмму.

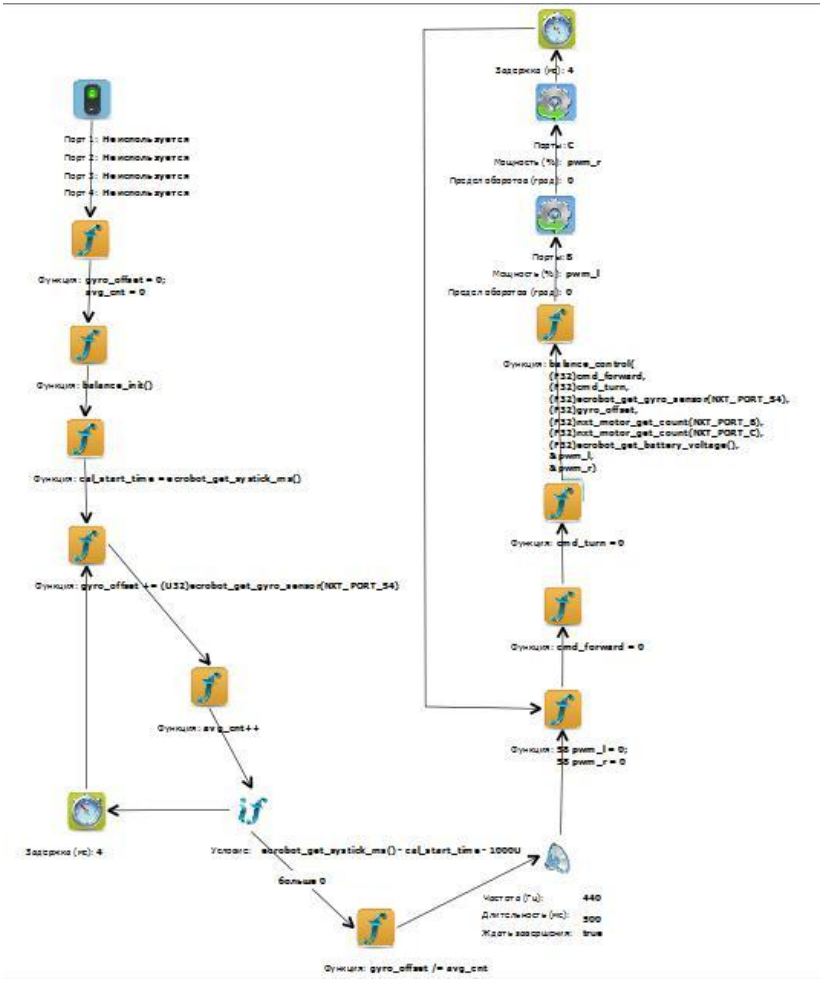

Рис.9: Программа балансирования робота на двух колесах

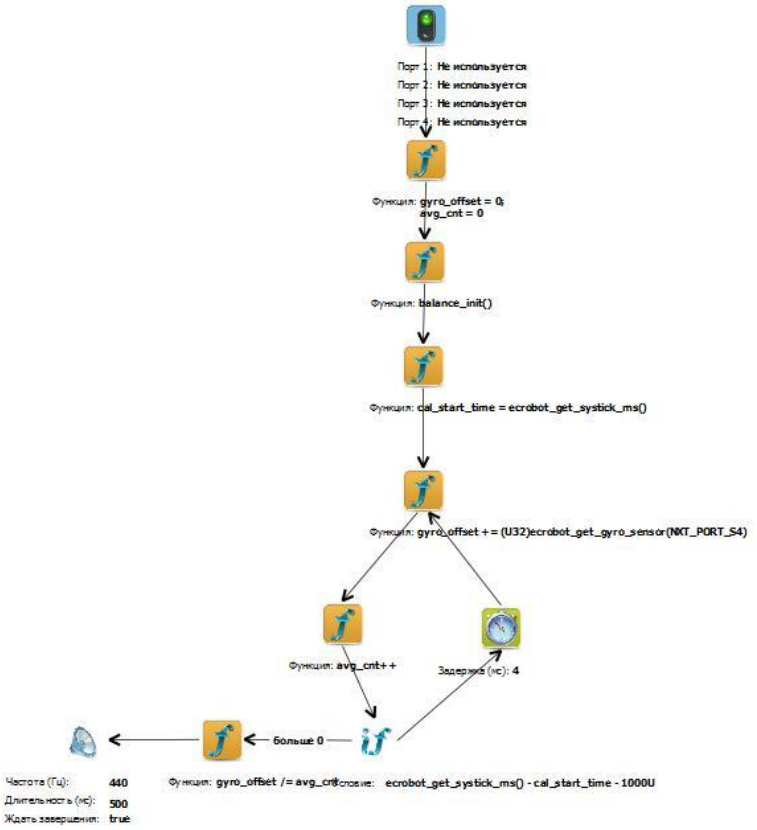

Рис. 10: Инициализация для программы балансирования робота на двух колесах

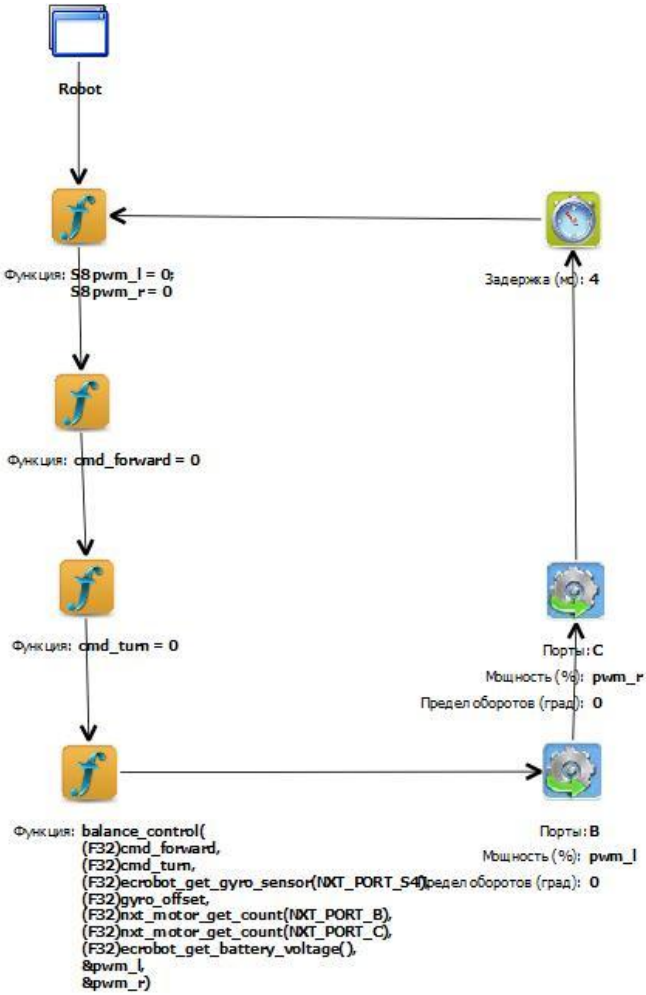

Рис. 11: Основная программа для балансирования робота на двух колесах

4. Автоматическое расположение элементов. Демонстрация применения автоматического расположения элементов слева направо.

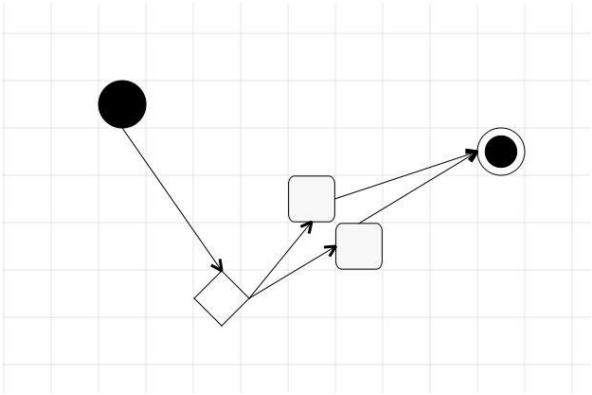

Рис. 12: Диаграмма до применения автоматического расположения

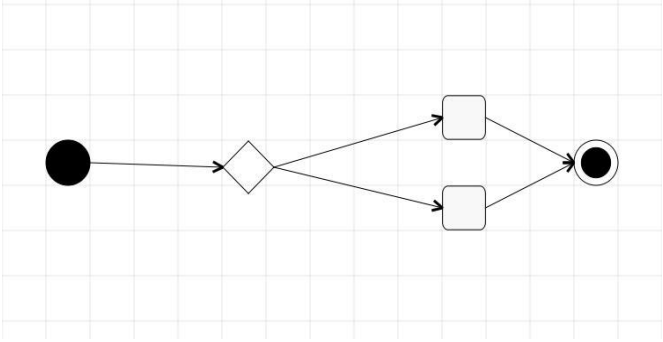

Рис. 13: Диаграмма после применения автоматического расположения

Заключение

Данная работа направлена на исследование вопросов применимости рефакторингов в области модельно-ориентированной разработки ПО. В настоящее время не существует однозначных рекомендаций, которым нужно следовать при решении основных проблем, возникающих в данной области.

В результате проведенного анализа существующих решений было выявлено, что инструментариев, имеющих возможность задавать рефакторинг моделей, немного, и их возможности весьма ограничены. Стоит отметить, что, как правило, это связано с проблемой сохранения целостности модели в результате применения рефакторинга. Естественным является желание иметь универсальный способ задания рефакторингов моделей в рамках MDE-подхода, но для обеспечения целостности модели требуется знание семантики языка. Однако, формальное описание семантики используемых языков не является типичной практикой при модельно-ориентированной разработке ПО, поэтому каждый разработчик средств поддержки рефакторингов моделей вынужден искать пути решения этой проблемы в контексте своей инструментальной среды.

На данный момент основными решениями проблемы обеспечения целостности модели являются либо сильное ограничение на возможности задания рефакторинга, либо перенесение полной ответственности за сохранение целостности на пользователя. В нашем подходе в настоящее время используется последний вариант, но в дальнейших исследованиях планируется предложить другие способы обеспечения целостности модели и реализовать их в среде QReal.

Стоит отметить, что особенностью языков, выбранных для апробации разработанного механизма рефакторингов, является отсутствие вложенных элементов. В связи с этим функциональность, связанная с поиском шаблонов для рефакторинга ограничена рассмотрением верхнего слоя элементов (т.е. элементы, содержащиеся в элементах-контейнерах, игнорируются).

В процессе работы был реализован механизм задания рефакторингов в metaCASE-системе QReal, позволяющий создавать несложные правила рефакторингов и применять им к диаграммам, созданным на основе некоторых визуальных языков среды. Данный механизм дает возможность пользователю быстро вносить большое

количество однотипных изменений в диаграмму и тем самым ускорить процесс разработки.

Литература

[1] Martin Fowler, Refactoring. Improving the Design of Existing Code. // Addison-Wesley, 1999

[2] Tom Mens, Gabriele Taentzer, Dirk Müller, Challenges in Model Refactoring. // Proc. 1st Workshop on Refactoring Tools University of Berlin (2007), pp.75-81

[3] Enrico Biermann, Karsten Ehrig, etc., EMF Model Refactoring based on Graph Transformation Concepts. // Electronic Communications of the EASST. Volume 3. 2006, pp. 3-19

[4] Jing Zhang , Yuehua Lin , Jeff Gray, Generic and Domain-Specific Model Refactoring using a Model Transformation Engine. // Volume II of Research and Practice in Software Engineering. 2005, pp. 199-218

[5] Pieter Van Gorp, Niels Van Eetvelde, Dirk Janssens, Implementing refactorings as graph rewrite rules on a platform independent metamodel. // Proceedings of the 1st International Fujaba Days, University of Kassel, Germany, October 13-14, 2003, pp. 17-24

[6] Tom Mens, On the Use of Graph Transformations for Model Refactoring. // Lecture Notes in Computer Science, Volume 4143, Generative and Transformational Techniques in Software Engineering, 2006, pp. 219-257

[7] Кузенкова А.С., Дерипаска А.О., Таран К.С., Подкопаев А.В., Литвинов Ю.В., Брыксин Т.А., Средства быстрой разработки предметно-ориентированных решений в metaCASE-средстве QReal // Научно-технические ведомости СПбГПУ, Информатика, телекоммуникации, управление. Вып. 4 (128). СПб.: Изд-во Политехнического Университета. 2011, С. 142-145.

[8] Брыксин Т.А., Литвинов Ю.В., Среда визуального программирования роботов QReal:Robots // Материалы международной конференции "Информационные технологии в образовании и науке". Самара. 2011. С. 332-334.

[9] Сорокин А.В., Кознов Д.В., Обзор Eclipse Modeling Project. // Системное программирование. Вып. 5. СПб.: Изд-во СПбГУ. 2010, с. 6-31

[10] Поляков В.А., Средства создания визуальных интерпретаторов диаграмм в системе QReal. // Курсовая работа, 2012.