Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Агапова Татьяна Юрьевна

Ручная спецификация миграции моделей в DSM-платформе QReal

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
ст. преп. Брыксин Т. А.

Рецензент:
ст. преп. Литвинов Ю. В.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Chair of Software Engineering

Tatiana Agapova

Manual specification of model migration strategy in QReal DSM platform

Bachelor's Thesis

Admitted for defence.
Head of the chair:
Professor Andrey Terekhov

Scientific supervisor:
Senior lecturer Timofey Bryksin

Reviewer:
Senior lecturer Yurii Litvinov

Saint-Petersburg
2015

Оглавление

Введение

При использовании модельно-ориентированного подхода [2] к разработке программного обеспечения система описывается с помощью набора моделей на некотором языке моделирования, чаще всего визуальном. Часто при этом используются узко специализированные предметно-ориентированные языки моделирования [8]. Поскольку каждая задача или предметная область требуют создания нового языка, удобно иметь инструмент, позволяющий быстро разрабатывать и использовать визуальные предметно-ориентированные языки моделирования. Такие инструменты называются DSM-платформами (domain specific modelling). При создании языка моделирования с помощью DSM-платформы его синтаксис (набор элементов и связей языка, свойства элементов) описывается также с помощью модели на некотором специальном языке, называемой метамоделью создаваемого языка.

Подобно прочим программным системам языки моделирования со временем развиваются. Причиной необходимости изменений в языке моделирования могут стать неточности или ошибки в первоначальной модели предметной области, а также расширение возможностей языка. В результате эволюции языка ранее созданные с его помощью модели могут стать некорректными в новой версии (к примеру, если в модели предметной области некоторая сущность была заменена на набор более подробно описывающих явление сущностей, из языка пропадёт соответствующий элемент, и с моделью, в которой присутствовал этот элемент, будет невозможно работать в новой версии языка). Таким образом, возникает задача миграции старых моделей в новую версию [20] – произведения таких изменений в модели, чтобы результат был корректен в новой версии языка и максимально сохранял семантику старой модели.

Наиболее простым решением задачи миграции является указание пользователю на несоответствие модели языку и предоставление ему возможности произвести необходимые изменения. Такой подход используется в MetaEdit+ [21] и в работе [28]. Полностью ручная миграция является крайне трудоёмкой, из-за чего повышается вероятность ошибок при преобразовании моделей. В связи с этим необходим способ спецификации миграционной стратегии и её автоматического применения к моделям. В дальнейшем в данной работе рассматриваются только автоматизированные подходы к миграции.

В настоящее время существует множество различных подходов к автоматизации процесса миграции моделей [5, 6, 15]. Весьма привлекательной кажется полностью автоматическая миграция на основе сопоставления старой и новой метамоделей языка, не требующая ни от разработчика, ни от пользователя языка никаких дополнительных действий. К сожалению, как было показано в [16], во многих нетривиальных случаях миграционную стратегию невозможно вывести автоматически. Кроме того, сопоставление метамоделей не приносит никаких результатов, если изменился не син-

таксис языка, а только его семантика: в этом случае метамодели старого и нового языков идентичны, и формально старые модели остаются корректными, но меняется их смысл, что может влиять на зависящие от модели артефакты, к примеру, на генерируемый по модели код. Таким образом, для достижения наиболее точного и корректного результата в дополнение к автоматической миграции полезно иметь средства ручной спецификации преобразований моделей.

Данная работа является продолжением моей курсовой работы [23] по разработке и реализации механизма миграции моделей в DSM-платформе QReal [29], разрабатываемой на кафедре системного программирования Санкт-Петербургского государственного университета. В ходе упомянутой работы был разработан гибридный подход к миграции, учитывающий специфические особенности QReal и сочетающий преимущества ручной и автоматической миграции. Также был реализован механизм автоматической миграции на основе разницы старой и новой метамоделей языка и журнала изменений метамодели между версиями. Таким образом, для полной реализации подхода требуется дополнить его средствами ручной спецификации миграции.

Постановка задачи

Целью данной работы является разработка механизма ручной спецификации миграций в QReal. В ходе работы необходимо решить следующие задачи.

1. Изучение существующих подходов к спецификации преобразований моделей, выбор наилучшего подхода для реализации в QReal с учётом особенностей системы.

2. Реализация средств спецификации и применения миграционных преобразований моделей.

3. Обеспечение целостности моделей при совместном использовании механизмов ручной и автоматической миграции.

4. Апробация решения.

1. Обзор

1.1. Преобразования и миграции моделей

Преобразования моделей лежат в основе модельно-ориентированного подхода к разработке [19], и, как следствие, существует большое количество различных подходов к их описанию [11]. Поскольку миграции моделей являются частным случаем преобразований типа "модель-модель", при их спецификации вполне применимы более общие средства описания трансформаций моделей.

С другой стороны, миграции как преобразования обладают отличительными особенностями [20]: метамодели исходной и целевой моделей хоть и различны, но очень похожи. В связи с этим можно ожидать, что большинство элементов мигрируемых моделей останутся без изменений, и хотелось бы обеспечить их автоматическое сохранение/копирование в целевую модель без необходимости явной спецификации такого поведения. Эту возможность предоставляют специализированные средства описания миграций.

В данном разделе рассматриваются основные подходы к спецификации преобразований моделей с точки зрения их применимости к задаче миграции. Также приведено несколько наиболее значимых средств описания миграции моделей. Кроме того, рассмотрены возможности спецификации преобразований моделей, уже реализованные в QReal.

1.2. Подходы к преобразованию моделей

Среди многообразия подходов к преобразованиям моделей можно выделить следующие основные категории [11]:

- прямое обращение к модели (direct model manipulation) с помощью языка программирования общего назначения;
- использование промежуточного представления (к примеру, XML);
- использование специализированного языка трансформации моделей.

При использовании прямого обращения к модели инструмент моделирования предоставляет пользователю возможность изменять модель с помощью некоторого API[1] на языке программирования общего назначения. Такой подход позволяет пользователю производить любые преобразования в пределах, установленных предоставляемым интерфейсом, и устанавливать наиболее гибкие политики применения преобразований. С другой стороны, использование API требует от него умения программировать

[1] application programming interface

на соответствующем языке общего назначения. Кроме того, этот подход достаточно низкоуровневый, что ведёт к большому объёму и сложности кода, необходимого для описания трансформаций, недостаточной степени переиспользования кода, а также к отсутствию наглядности при спецификации преобразований.

Подходы, основанные на промежуточном представлении, предполагают, что модель переводится в стандартный формат, например, в XML. Это промежуточное представление затем может быть подвергнуто преобразованиям с использованием XSLT[2]. Основанные на XSLT трансформации обладают большой выразительной силой, но данный подход крайне трудоёмок и требует серьёзной подготовки пользователя даже для описания простейших преобразований и так же, как и прямое обращение к модели, страдает от недостаточной наглядности. Ещё одним недостатком преобразований с помощью XSLT является сложность контроля корректности: трансформации могут изменять XML-представление модели произвольным образом, и требуются дополнительные усилия для того, чтобы удостовериться, что результат будет корректной моделью на целевом языке.

Использование языков преобразований моделей было признано наиболее эффективным способом трансформации моделей благодаря их специализированности, которая позволяет скрыть сложность алгоритмов преобразования в реализации языка, сделать процесс описания преобразований более простым для пользователя и ослабить требования к его подготовке. В настоящее время существует большое количество различных языков трансформации, в том числе предназначенных для описания миграций моделей. Исчерпывающую классификацию можно найти в [4]. Ниже представлены некоторые типы и свойства языков преобразования, которые необходимо принимать во внимание при разработке языка спецификации миграций моделей.

1.3. Языки описания преобразований моделей

1.3.1. Правила преобразования

В общем случае правило преобразования состоит из двух частей: левой (left hand side) и правой (right hand side). При этом левая часть описывает шаблон, который должен быть найден в исходной модели, а правая представляет собой фрагмент целевой модели, который должен быть создан при нахождении шаблона из левой части. В некоторых подходах части правила описываются явно, в некоторых – неявно следуют из описания преобразования.

Правила трансформации могут быть специфицированы с помощью как визуальных [1, 10], так и текстовых [7] языков. В большинстве случаев визуальные языки предоставляют более наглядное представление трансформаций. Однако в некоторых ситуациях (к примеру, для итерации по смежным элементам или проверки отрица-

[2]Extensible Stylesheet Language Transformations

тельных условий) текстовое описание может быть более удобным и гибким. В связи с этим существуют подходы, сочетающие возможности как графического, так и текстового описания преобразований (например, QVT Relations [10] поддерживают как графический, так и визуальный синтаксис).

При визуальном описании преобразований моделей широко используются графовые грамматики [18]. Правило преобразования графов задаётся парой графовых шаблонов, описывающих его левую и правую части. Спецификация шаблонов может происходить в конкретном синтаксисе исходного и целевого языков либо в абстрактном синтаксисе языка трансформаций. Конкретный синтаксис при этом имеет определённые преимущества в сравнении с абстрактным: описание трансформаций ведётся в терминах исходного и целевого языков, что, во-первых, обеспечивает синтаксическую корректность шаблонов, а во-вторых, делает их описания более компактными и наглядными.

И текстовые, и визуальные языки могут быть императивными и декларативными. Императивные языки, как правило, являются более гибкими и предоставляют возможности для явного описания алгоритма преобразования: средства обхода графа исходной модели, средства проверки условий применимости правил, вызов одних правил другими, композиция правил и др. С другой стороны, при использовании декларативной парадигмы специфицируются только сами правила преобразования, а алгоритм их применения скрыт в реализации. Благодаря этому описания трансформаций на декларативных языках лучше воспринимаются человеком, но это достигается за счёт уменьшения выразительной мощности и точности преобразования. Некоторые подходы стремятся к сочетанию компактности декларативных языков и выразительности императивных, к примеру, подобный гибридный подход используется в Epsilon Flock [15].

Сравнение использования текстовых и визуальных подходов для описания миграции моделей [13] показало, что они одинаково эффективны для решения поставленной задачи.

1.3.2. Управление применением преобразований

Помимо описания правил трансформации, при разработке подхода к преобразованиям моделей важно также рассмотреть вопрос применения правил.

Управление применением преобразований может как происходить явно под контролем пользователя, так и неявно определяться инструментом моделирования. При использовании явной спецификации способа применения правил пользователь может управлять такими аспектами применения трансформаций, как выбор следующего правила, порядок применения правил, определение условий выполнения, композиция правил, вызов одних правил другими и разрешение конфликтов.

В тех случаях, когда политика применения трансформаций определена инструментом неявно, ответственность за выбор лучших алгоритмов лежит на разработчике системы. Оптимальный подход в этом случае будет зависеть от типа предполагаемых трансформаций.

Специфика миграции состоит в том, что преобразования должны применяться в определённом порядке – от версии к версии. Это ограничение должно учитываться при предоставлении пользователю возможностей управления порядком применения правил.

1.3.3. Соотношение исходной и целевой моделей

Важным техническим моментом при реализации применения правил является решение, должна ли в результате создаваться с нуля новая модель, или же преобразование должно происходить "на месте".

Преобразования могут быть эндогенными (в этом случае исходная и целевая модель имеют общую метамодель) либо экзогенными (метамодели исходной и целевой модели различаются) [11]. В общем случае для экзогенных преобразований невозможно произвести трансформацию "на месте".

Миграция моделей является экзогенным преобразованием, так как исходный и целевой языки, хоть и похожи, но всё же отличаются. Несмотря на это, предполагается, что большая часть исходной модели останется нетронутой миграцией. Миграционные преобразования "на месте" в этом случае могут помочь избежать копирования большинства элементов модели.

1.4. Существующие средства миграции моделей

1.4.1. Epsilon Flock

Epsilon Flock [15] – язык описания миграции, принадлежащий семейству языков трансформации моделей Epsilon [9].

В результате миграции с помощью Flock создаётся новая модель, в которую автоматически копируются элементы старой модели, не нарушающие её корректности в новой версии языка. Правила, описанные пользователем, предназначены для того, чтобы скорректировать и направить процесс копирования элементов в новую модель.

Flock имеет текстовый синтаксис, сочетающий декларативные и императивные возможности. Правила бывают трёх видов: замена типа (retype), обновление элемента (migrate) и удаление элемента (delete). Конкретный синтаксис правил представлен на рисунке 1. Все правила содержат тип мигрируемого элемента и условия применимости правила. При обновлении также можно указать набор свойств, которые должны быть отброшены при копировании, и подробно описать миграцию данного элемента в императивном стиле.

```
1  retype <originalType> to <evolvedType>
2  (when (:<eolExpression>)|({<eolStatement>+}))?
3
4  delete <originalType>
5  (when (:<eolExpression>)|({<eolStatement>+}))?
6
7  migrate <originalType>
8  (ignoring <featureList>)?
9  (when (:<eolExpression>)|({<eolStatement>+}))? {
10   <eolStatement>+
11 }
```

Рис. 1: Конкретный синтаксис Epsilon Flock [17].

Применение правил осуществляется следующим образом: для каждого элемента выбирается первое подходящее правило, на его основании создаётся (кроме правил удаления) элемент целевой модели, и свойства старого элемента копируются в новый при совпадении имён. Если для некоторого элемента не было обнаружено применимых правил, выполняется миграция по умолчанию: элемент копируется в целевую модель, если он присутствует в соответствующей метамодели, и игнорируется в ином случае. Таким образом, при применении стандартной миграционной стратегии в результате получится копия исходной модели без тех элементов, которые не соответствуют целевому языку.

Flock ориентирован на описание простых шаблонов миграции, в "левой части" которых находится один элемент. Более сложные шаблоны могут быть заданы с помощью дополнительных проверок и частичного обхода модели в теле правила. Тем не менее, подобным императивным описаниям может недоставать наглядности и простоты восприятия.

Помимо приоритизации правил посредством расположения их в файле Flock не имеет средств управления порядком их применения. Также отсутствуют средства композиции преобразований, которые были бы полезны для миграции модели на несколько версий вперёд.

1.4.2. Ecore2Ecore

Ecore2Ecore [12] – средство спецификации миграций моделей, входящее в состав Eclipse Modelling Framework (EMF)[3].

Описание миграции с помощью Ecore2Ecore происходит в две фазы. На первой стадии пользователь в графическом редакторе производит соответствие элементов двух версий языка. Таким способом могут быть заданы только самые простые миграции, такие как переименование элементов или их свойств. Описание ведётся в конкретном

[3] http://www.eclipse.org/modeling/emf/

синтаксисе, и поэтому для графической спецификации миграции необходимо наличие метамоделей обеих версий языка.

На втором шаге пользователь может описать на Java более сложные преобразования. При этом используется довольно низкоуровневый API, подразумевающий разбор строк и приведение типов вручную пользователем, что усложняет восприятие кода преобразования. На этом этапе доступ к метамоделям языка не требуется.

С помощью указанных механизмов Ecore2Ecore позволяет задать далеко не все возможные миграции [14]. У пользователя есть возможность описывать произвольные преобразования, но для этого ему придётся отказаться от поддержки автоматического копирования элементов в целевую модель.

В Ecore2Ecore нет встроенной поддержки композиции миграций, но пользователь может достичь нужного эффекта при подробном описании миграции на Java.

1.5. QReal

1.5.1. Поиск шаблона на графе модели

Преобразования моделей возникают во многих задачах модельно-ориентированной разработки, и QReal уже обладает необходимыми средствами.

В QReal поддерживается подход к преобразованию моделей, основанный на графовых грамматиках. Важнейшей задачей при его реализации является поиск заданного шаблона на графе модели, и QReal предлагает довольно общие средства для её облегчения. Используемый алгоритм поиска шаблона подробно описан в работе Владимира Полякова [26]. Отдельные шаги алгоритма, такие как сравнение элементов и их атрибутов, могут переопределяться или уточняться для решения конкретной задачи сопоставления с шаблоном.

Алгоритм поиска работает с шаблонами, представленными в виде модели QReal. Других ограничений на используемый формат описания шаблонов нет, благодаря чему данный механизм можно использовать в самых различных задачах преобразования моделей. В настоящее время поиск шаблонов используется в двух модулях QReal: в редакторе семантики визуальных языков [26] и в механизме рефакторингов [24].

1.5.2. Семантика визуальных языков

Средства задания семантики предназначены в основном для разработки визуальных интерпретаторов и визуальных отладчиков, что отразилось на выбранном способе описания правил преобразования. Левая и правая части правила в этом языке объединены, а для обозначения удаляемых и создаваемых элементов используются специальные метки. Язык имеет отдельную сущность для обозначения произвольного элемента модели. На каждом шаге преобразования выбирается применяемое правило,

при этом управлять выбором правил можно с помощью присвоения им приоритетов. Пример описания семантики языка блок-схем приведён на рисунке 2.

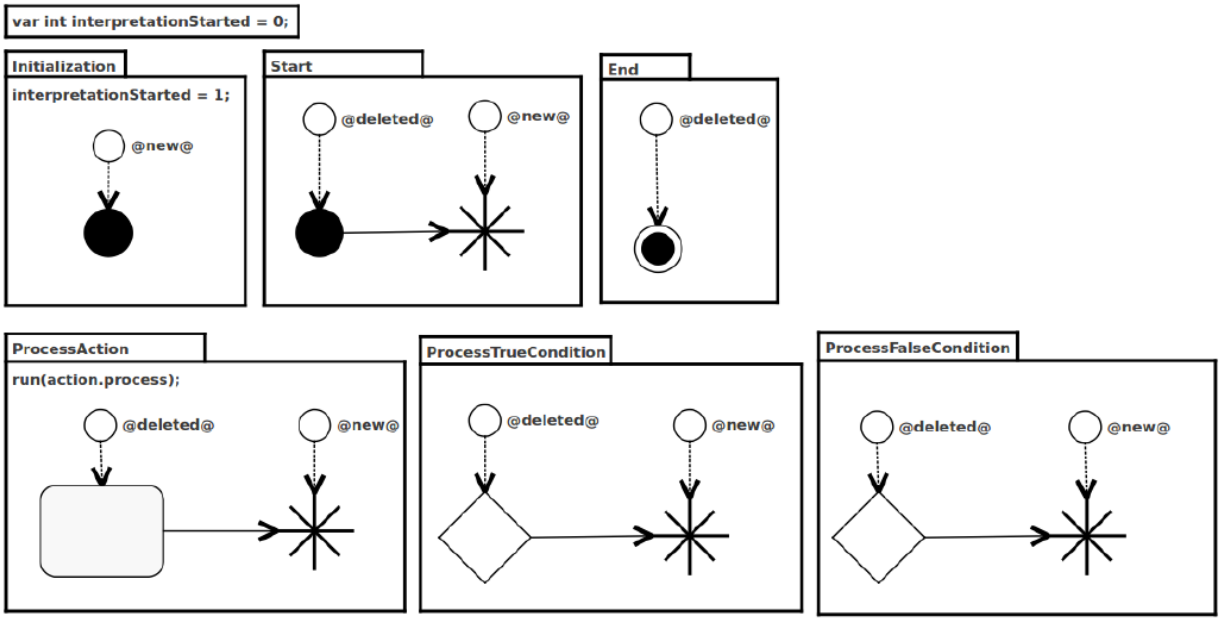

Рис. 2: Семантика языка блок-схем [26].

1.5.3. Рефакторинги

Правила рефакторингов имеют более традиционный для графовых грамматик вид: левая и правая части правила отделены, а для описания соответствия элементов используются специальные идентификаторы. Можно указать, что значение свойства при замене элемента должно остаться прежним. Также присутствуют элементы, обозначающие произвольный элемент или произвольную связь языка. Рефакторинг модели производится в интерактивном режиме: пользователь сам выбирает правило для применения и вхождение шаблона, которое нужно заменить. В связи с этим не требуются дополнительные механизмы определения порядка или приоритета правил. На рисунке 3 изображён пример простого рефакторинга, разворачивающего в другую сторону все связи в модели.

1.5.4. Механизм конвертеров

В данный момент поддержка миграции в QReal осуществляется посредством прямого обращения к модели. Разработчик языка может определить один или несколько так называемых конвертеров – функций, принимающих на вход интерфейс к модели и преобразовывающих её в процессе применения. С помощью данного интерфейса конвертер может изменять модель произвольным образом. Описание конвертеров происходит, как и разработка подключаемых модулей, с использованием C++/Qt.

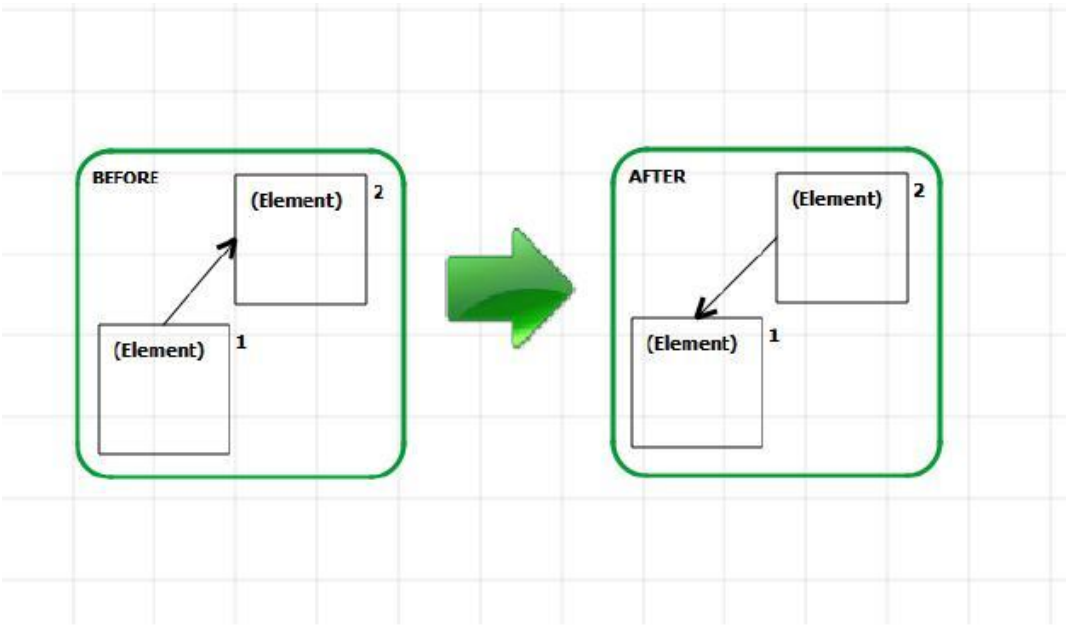

Рис. 3: Пример рефакторинга модели [24].

Для использования данного механизма необходимо при разработке визуального языка указать его версию в метамодели. При этом строка, описывающая версию, должна удовлетворять определённому формату. Перед применением конвертеры фильтруются и сортируются по номеру версии так, чтобы при миграции применялись только нужные конвертеры и в правильном порядке от версии к версии. Разработчик конвертеров при этом обязан обеспечить их непересекаемость (например, не должно быть двух конвертеров с версии “1” в версию “3” и с версии “2” в версию “4”).

Важно, что для успешного применения конвертеров не требуется метамодель ни исходной, ни целевой версий языка. Номер версии языка исходной модели находится в её файле сохранения, что позволяет механизму миграции “знать”, какие конвертеры необходимо применить.

Описанный подход позволяет задавать произвольные преобразования модели, но вместе с тем он обладает некоторыми недостатками:

- спецификация преобразований на C++ – довольно трудоёмкий процесс;
- код на языке общего назначения – не самый наглядный способ описания преобразований модели;
- предоставляется только базовый интерфейс к модели, и если потребуется произвести поиск сложного шаблона, его каждый раз придётся писать заново;
- визуальное метамоделирование позволяет создавать и использовать языки даже людям без подготовки, но миграцию при использовании такого подхода сможет произвести только программист C++.

В связи с обозначенными недостатками механизма конвертеров было бы полезно иметь в дополнение к нему также более простой и наглядный способ описания миграции.

Также стоит отметить возможность расширения интерфейса конвертеров путём реализации внутреннего предметно-ориентированного языка, реализующего распространённые преобразования. Подобный язык уже используется для описания конвертеров в среде программирования роботов TRIK Studio [25], но его поддержки на уровне DSM-платформы в настоящее время нет.

2. Требования к ручной спецификации миграции

Прежде чем определить требования к средству ручного описания правил миграции, стоит обратить внимание на некоторые особенности QReal.

Рассматриваемая система предоставляет возможности создания визуальных языков пользователю, не обладающему навыками программирования: для описания языков используются визуальные средства, не требующие написания кода на языках общего назначения. Механизм описания миграции по возможности также не должен требовать от пользователя специальной подготовки.

Создание и использование визуальных языков в QReal может происходить в двух режимах: генерации редактора и интерпретации метамодели языка. В режиме генерации создаётся подключаемый модуль, реализующий описанный метамоделью язык. При интерпретации, напротив, метамодель напрямую используется системой для получения информации о языке, при этом не происходит генерации никаких промежуточных артефактов. Подсистема миграции должна поддерживать оба режима работы.

С учётом проведённого выше сравнения различных подходов к задаче преобразования моделей и особенностей платформы QReal можно определить следующие требования к механизму ручной спецификации миграций:

- наглядность – описания преобразований должны быть как можно более простыми для восприятия и понимания;
- выразительность – в идеале язык должен позволять описывать любые преобразования моделей;
- точность – результат преобразования должен как можно лучше отражать намерения разработчика миграции;
- эффективность – в процессе преобразования следует избегать ненужных действий, таких как копирование не затронутых миграцией элементов;
- минимальные требования к подготовке пользователя;
- наличие возможности управлять порядком применения правил;
- совместимость с механизмом конвертеров;
- совместимость с механизмом автоматической миграции [23];
- возможность использования в режиме интерпретации метамодели.

Желательными также являются:

- переиспользование существующих в QReal механизмов преобразования моделей;

- наличие средств статической проверки корректности преобразований (например, не должно быть возможности указать в левой части правила элемент, не принадлежащий исходному языку).

3. Описание предложенного подхода

3.1. Спецификация миграций

Требование наглядности и простоты описаний преобразований имеет большое значение в системе QReal, предоставляющей возможности создания визуальных языков пользователям, не являющимся профессиональными программистами. В связи с этим предпочтение было отдано декларативному визуальному языку, основанному на графовых грамматиках. По сравнению с текстовыми и императивными подходами, выбранное решение значительно снижает требования к подготовке пользователя, поскольку графическое представление шаблонов, как правило, интуитивно понятнее и проще, чем эквивалентное преобразование, описанное кодом на императивном текстовом языке.

Выбор графовых шаблонов для описания правил позволяет переиспользовать инструменты поиска на графе модели, уже реализованные в QReal. Тем не менее, использовать существующие механизмы преобразования моделей для решения задачи миграции не представляется возможным по следующим причинам.

1. Как визуальная интерпретация, так и рефакторинги являются примерами эндогенных преобразований, так как языки исходной и целевой моделей совпадают. Это позволяет описать трансформацию с помощью одной модели, а в случае визуальных семантик и вовсе совместить левую и правую части правила. В случае миграции исходный и целевой языки различаются, что требует чёткого разделения частей.

2. Оба рассмотренных инструмента для получения редактора языка преобразований вносят изменения в метамодель исходного языка, а затем генерируют и подключают редактор в виде отдельного модуля. Подобный процесс представляется недопустимо долгим и сложным при спецификации миграций. Кроме того, его побочным продуктом является сгенерированный модуль с редактором языка преобразований, что при использовании для описания миграции приведёт к генерации редакторов для каждой версии языка, участвующей в определении миграций.

В связи с рассмотренными ограничениями появляется необходимость создания отдельного языка для описания миграций. При этом предлагается использование конкретного синтаксиса для спецификации преобразований, так как он более нагляден по сравнению с абстрактным. Кроме того, описание трансформаций в терминах конкретных визуальных языков предоставляет больше возможностей для статического контроля корректности шаблонов. Таким образом, в язык описания миграций должны

входить все элементы исходного визуального языка. С целью достижения компактности и выразительности правил в язык добавлено три метаэлемента, обозначающих любой элемент, любую связь или любое свойство исходного языка. Для того чтобы избежать генерации и подключения отдельного модуля для каждой версии языка, работа с полученными языками ведётся в режиме интерпретации метамодели [27].

Наиболее трудным представляется нахождение компромисса между простотой и наглядностью с одной стороны, и точностью и выразительностью с другой. Данные требования в некоторой степени противоречат друг другу, так как описание точных и подробных условий применения правила в общем случае делает его более громоздким и сложным для восприятия.

Для достижения выразительности языка зачастую приходится жертвовать его простотой, так как расширение возможностей, как правило, требует введения дополнительных конструкций. В предлагаемом подходе имеется возможность в левой части правила с помощью регулярных выражений описывать шаблон, которому должно соответствовать значение свойства элемента, а в правой части при определении нового значения свойства ссылаться на старые значения свойств данного или других элементов и даже на отдельные фрагменты этих значений, захваченные при сопоставлении значения с регулярным выражением. Это предоставляет большую гибкость при преобразовании значений свойств элементов, но для использования данной возможности пользователю необходимо знать язык регулярных выражений, что является довольно серьёзным требованием к его подготовке.

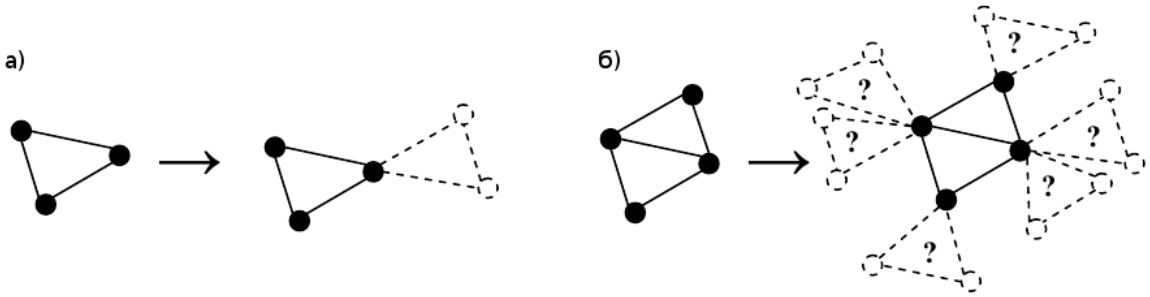

Рис. 4: а) Описание правила преобразования. б) Неоднозначность применения правила [20].

Точность – одна из наиболее сложных задач при описании преобразований моделей. Даже в случае простых правил достижение однозначности результата может оказаться нетривиальным [20]. Рассмотрим, к примеру, шаблон, изображённый на рисунке 4а. В результате его применения к модели с рисунка 4б возможны шесть преобразований. Достичь лучшей точности результата может помочь возможность задания дополнительных условий применения шаблона (например, отрицательных условий, выполнение которых предотвращает применение правила). Несмотря на то, что отрицательные условия не были реализованы в рамках данной работы, их под-

держка является желательным направлением дальнейшего развития предложенного языка трансформаций.

3.2. Применение правил миграции

При определении политики применения правил разработчику механизма преобразований необходимо решить несколько важных вопросов, в числе которых следующие:

- в каком порядке применять правила;
- когда считать применение правила завершённым (наиболее распространённые решения здесь – однократное применение и политика неподвижной точки, когда правило применяется до тех пор, пока это приводит к изменениям в модели);
- как разрешать конфликты при обнаружении пересекающихся вхождений шаблонов;
- какова роль пользователя в решении данных вопросов.

В предложенном подходе у пользователя есть два способа влиять на применение правил. Во-первых, он может определить порядок применения преобразований, относящихся к одинаковым версиям исходного и целевого языков (это ограничение происходит из самой задачи постепенной миграции модели от версии к версии). Во-вторых, для каждого правила пользователь может указать способ завершения, при этом доступны политики однократного применения или неподвижной точки.

Перед применением правила определяется, к какому набору вхождений левой части оно будет применяться. Простейшее решение – выбор случайного максимального набора непересекающихся вхождений шаблона. Подобная политика может привести к неоднозначностям при применении правила. Тем не менее, в большинстве случаев их можно избежать путём спецификации более сложного шаблона с дополнительными ограничениями на применение, чтобы уменьшить количество соответствующих шаблону подграфов, а следовательно, и вероятность их пересечения.

Ещё один вопрос, связанный с применением правил, касается способа преобразования: должна ли в результате создаваться с нуля новая модель, или же изменения производятся “на месте”. Как уже упоминалось, в задаче миграции можно ожидать небольших изменений в языке и в моделях на нём. В связи с этим, процесс миграции можно сделать более эффективным путём произведения преобразований “на месте” вместо копирования не подверженных изменениям элементов в новую модель. Предложенный подход использует это наблюдение: при обнаружении вхождения шаблона оно заменяется правой частью правила миграции, оставляя остальную модель неизменной.

3.3. Место ручных преобразований в подсистеме миграции

Заданные вручную правила преобразования – не единственный в QReal механизм поддержки миграции моделей. Помимо него для решения данной задачи используются также конвертеры, описанные на C++, и модуль автоматической миграции. Для достижения наиболее удовлетворительного результата необходима тесная интеграция указанных подсистем.

Поскольку и конвертеры, и визуальные шаблоны преобразований описываются разработчиком языка, ответственность за их согласование лежит именно на нём. С другой стороны, разработчик языка не имеет возможностей контролировать автоматическую миграцию, и поэтому сам механизм должен обеспечивать совместимость ручных и автоматических преобразований.

Выполнить данное требование помогает разделение процесса преобразования на три стадии: сначала применяются конвертеры, затем графовые шаблоны, и после этого – автоматически выведенные изменения. При этом приоритет в применении принадлежит описанным пользователем преобразованиям, автоматическая миграция же производит минимальные изменения, необходимые для обеспечения согласованности модели с новой версией языка.

Автоматические преобразования должны игнорировать элементы, уже изменённые на предыдущих шагах, чтобы не нарушать специфицированные пользователем изменения. Это достигается с помощью передачи системе автоматической миграции списка изменённых элементов. Разработчик конвертеров должен сформировать этот список самостоятельно в теле конвертера, а при применении правил, описанных визуально, изменённые элементы запоминаются автоматически.

4. Детали реализации

4.1. Процесс спецификации миграций

Переход на новую версию языка начинается с редактирования разработчиком языка его метамодели. При этом все произведённые изменения протоколируются, что позволяет в любой момент при наличии актуальной метамодели получить любую из предыдущих версий.

После проведения желаемых правок, создателю языка требуется зафиксировать новую версию. Для этого нужно отредактировать соответствующий атрибут диаграммы с метамоделью языка, указав номер версии, и нажать кнопку “Зафиксировать новую версию” на панели инструментов метаредактора. Заметим, что механизм графовых преобразований, в отличие от конвертеров, не зависит от формата строки, представляющей номер версии, так как при фиксации используется отдельная внутренняя нумерация версий, в соответствии с которой впоследствии и происходит сортировка графовых шаблонов.

Когда версия зафиксирована, можно создавать правила миграции, в которых она участвует. Управление правилами происходит с помощью диалога редактирования миграционных изменений, также доступного из панели инструментов метаредактора. Интерфейс диалога управления миграцией представлен на рисунке 5.

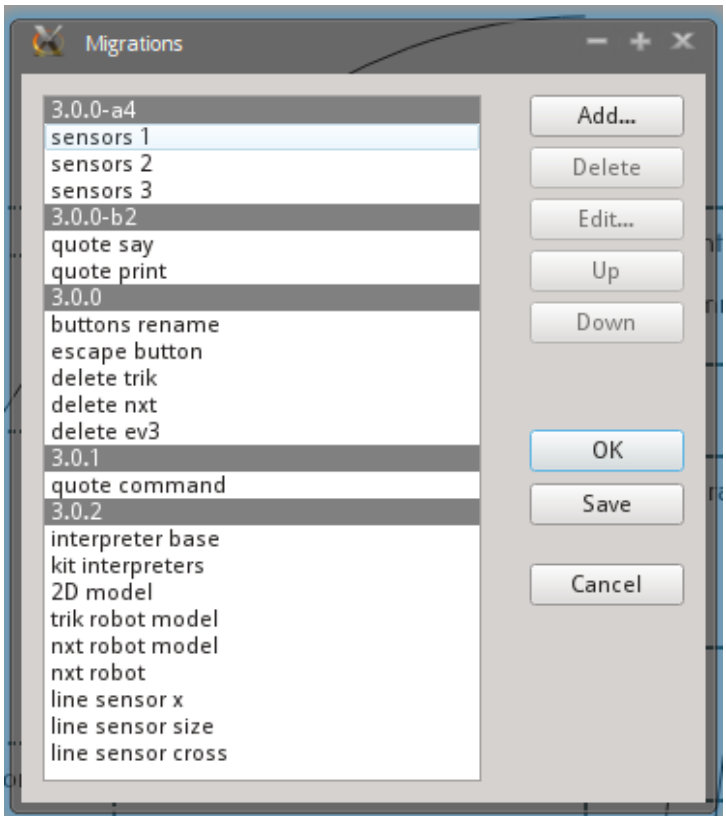

Рис. 5: Диалог управления миграцией.

Диалог редактирования миграций позволяет осуществлять следующие задачи:

- просмотр существующих правил миграции;
- создание новых миграций;
- удаление правил;
- редактирование существующих правил;
- изменение порядка применения правил в пределах, не нарушающих порядка версий;
- изменение дополнительных параметров применения правил, таких как условие завершения.

Отдельного рассмотрения заслуживает процесс спецификации правил преобразования. После нажатия на кнопку “Добавить правило” пользователю предлагается выбрать исходную и целевую версии. Вслед за этим для указанных версий формируются языки описания миграции. Это происходит в два этапа.

1. На первом шаге путём отмены запротоколированных изменений восстанавливаются метамодели обеих версий.
2. Затем полученные языки дополняются специфичными для миграции элементами и атрибутами элементов.

Результирующие метамодели загружаются в интерпретатор метамоделей, что позволяет описывать с их помощью миграции без генерации и подключения к системе дополнительных модулей.

Диалог спецификации преобразований изображён на рисунке 6. Окно диалога разделено на две одинаковые части. Каждая часть работает только с соответствующим языком: попытка использовать элементы левой части на правой диаграмме вызовет ошибку. Это позволяет гарантировать, что обе части шаблона синтаксически корректны и не содержат недопустимых элементов или атрибутов. Язык описания преобразований подробно описан в следующем разделе.

Заданные таким образом миграции сохраняются в файле с метамоделью, и доступны всегда, когда доступна сама метамодель.

4.2. Язык описания миграций

Как уже упоминалось, спецификация миграционных преобразований ведётся в конкретном синтаксисе указанных версий целевого языка, расширенном тремя дополнительными метаэлементами, имеющими специальную семантику при сопоставлении с элементами исходной модели:

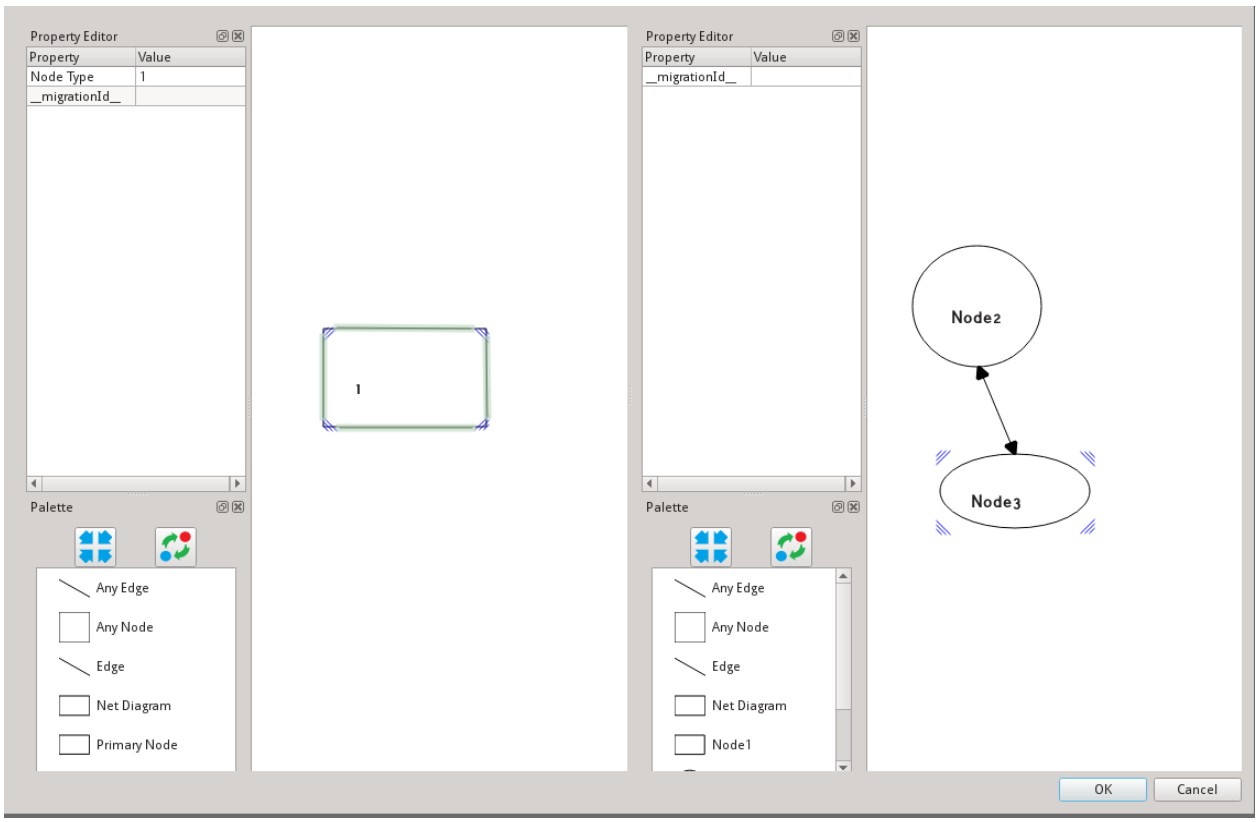

Рис. 6: Диалог спецификации миграций.

- “любой элемент” (Any Node) по умолчанию считается совпадающим с любым элементом модели и имеет атрибут “тип”, в котором пользователь может указать шаблон, которому должно удовлетворять название типа элемента модели;
- “любое свойство” (Any Property) – вкладывается в Any Node и имеет два атрибута: имя, в котором задаётся условие для названия свойства; значение, с которым будет сравниваться значение подходящего свойства элемента модели при сопоставлении;
- “любая связь” (Any Edge) – аналогично “любому элементу”, за исключением невозможности вложения в неё свойств.

Эти элементы также могут использоваться и в правой части правила:

- Any Node и Any Edge обязательно должны иметь сопоставленные им элементы из левой части; при замене шаблона вместо них будет создаваться элемент либо того же типа, как и тип найденного элемента модели, либо типа, указанного в поле “тип”, если оно не пусто;
- Any Property должен иметь либо сопоставленный ему элемент Any Property в левой части правила, либо непустое поле “имя”; это необходимо для определения, какое свойство описывает этот элемент.

Помимо введения дополнительных сущностей, язык миграций также добавляет во все элементы исходного языка поле “миграционный идентификатор” (migrationId), который используется для установления соответствия между элементами левой и правой частей. В случае, если для элемента правой части существует сопоставленный ему элемент из левой части, при применении преобразования этот элемент получит все связи старого элемента, а также его дочерние элементы и значения всех свойств, которые присутствуют в обоих элементах (если новое значение свойства не указано отдельно в шаблоне).

Условия, которым должны удовлетворять значения атрибутов элементов модели, указываются в атрибутах соответствующих элементов шаблона. При этом пустое значение означает, что данный атрибут не участвует в проверке соответствия шаблону. Непустое значение интерпретируется как регулярное выражение в синтаксисе Perl, и значение свойства мигрируемого элемента проверяется на соответствие этому выражению. Возможен также “захват” в регулярном выражении частей значений атрибутов и дальнейшее их использование при формировании новых значений свойств.

Похожим образом значения атрибутов элементов из правой части правила описывают свойства элементов, которые будут созданы в модели после применения этого правила. Пустое значение атрибута оставит соответствующее свойство без изменений. В свойствах элементов правой части можно ссылаться на старые значения свойств любых элементов левой части, имеющих установленный миграционный идентификатор, а также на части этих значений, захваченные при сопоставлении с шаблоном. Для этого требуется указать идентификатор элемента из левой части, имя свойства и номер части значения.

Заметим, что метаэлементы предоставляют возможность описывать преобразования с использованием исключительно абстрактного синтаксиса. Рекомендуется, однако, по возможности задавать миграции в терминах целевого языка, поскольку это позволяет избежать большинства синтаксических ошибок, таких как описание правил для элементов несуществующего типа.

4.3. Процесс миграции модели

При открытии файла с моделью проверяется возможность её миграции и загрузки с помощью имеющейся версии соответствующего языка. Для этого используются метаданные модели, содержащие информацию о версии языка, с помощью которой была создана данная модель. Эта версия сравнивается с текущей доступной в системе версией соответствующего языка, после чего делается вывод о необходимости и возможности миграции:

- если версия модели старше, чем текущая версия языка, использующаяся в системе, миграция необходима;

- если версия модели младше, это свидетельствует об использовании устаревшей версии языка; в этом случае пользователю будет предложено обновить модуль с реализацией языка, и загрузка модели в систему не произойдёт;
- если версия модели соответствует имеющейся версии редактора языка, миграция не требуется и модель загружается в систему без каких-либо преобразований.

В случае, если была выявлена необходимость миграции, модель последовательно проходит через три стадии преобразования:

- применение конвертеров;
- применение графовых преобразований;
- автоматическая миграция элементов, нарушающих корректность модели и не исправленных на предыдущих двух шагах.

Механизм конвертеров подробно рассмотрен в обзорной части данной работы, а детальное описание подсистемы автоматической миграции можно найти в [23]. Здесь мы остановимся на процессе применения правил, заданных с помощью графовых шаблонов.

Сначала из всех имеющихся правил отбрасываются те, которые описывают миграцию с версий старших, чем версия модели. Остальные правила сортируются с учётом последовательности версий и приоритетов, указанных разработчиком миграционной стратегии.

Далее правила по очереди применяются, при этом процесс преобразования можно разделить на несколько фаз.

1. Алгоритм поиска в графе находит в модели набор непересекающихся вхождений шаблона из левой части правила. Сравнение элементов шаблона и модели производится в соответствии с семантикой языка миграций, описанной в предыдущем разделе.
2. Каждое найденное вхождение заменяется на правую часть правила:
    - вначале создаются элементы из правой части; при этом каждый созданный элемент отмечается в списке, который впоследствии будет передан механизму автоматической миграции для предотвращения дальнейших преобразований уже изменённых элементов;
    - затем атрибуты и связи новых элементов инициализируются в соответствии с шаблоном либо копируются из соответствующих элементов из левой части правила;

- наконец, удаляются старые элементы.

3. В случае, если для данного правила разработчиком миграции была указана политика неподвижной точки, эти два шага будут повторяться до тех пор, пока поиск находит хотя бы одно вхождение шаблона в модель.

5. Апробация

Для апробации реализованного подхода к миграции было выбрано два языка. Первый из них используется в TRIK Studio [25] – среде визуального программирования роботов, разработанной на основе QReal. Второй – язык описания сетей Петри – является традиционным примером для демонстрации возможностей средств миграции и удобен для сравнения с другими инструментами.

5.1. TRIK Studio

TRIK Studio – наиболее зрелый продукт, созданный с помощью QReal. В течение своего развития данная среда претерпела значительные изменения, требующие миграции старых моделей на более новые версии. Это делает TRIK Studio идеальным кандидатом для апробации механизма миграции.

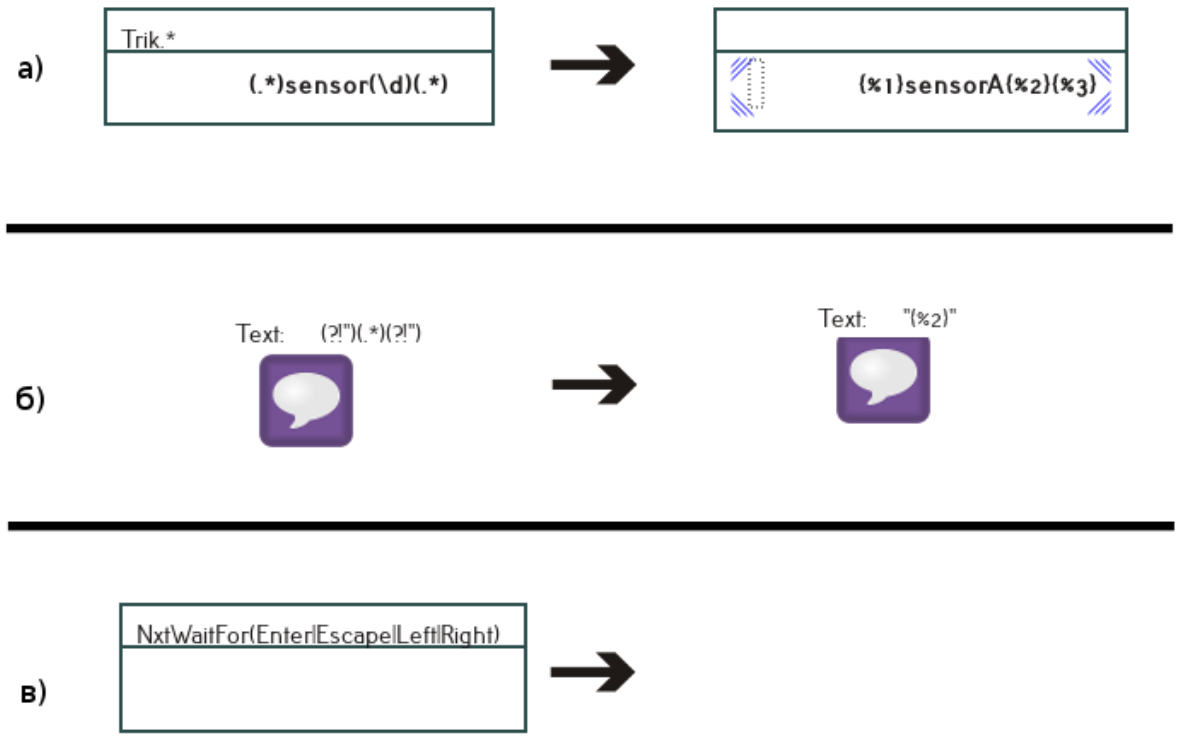

Рис. 7: Описания правил миграции TRIK Studio: а) замена подстроки в свойстве; б) заключение свойства в кавычки; в) удаление элемента.

Процесс миграции модели TRIK Studio можно описать с помощью 72 преобразований, каждое из которых принадлежит к одному из трёх типов:

- замена подстроки в значении свойства; это наиболее многочисленная группа преобразований, состоящая из 53 изменений;
- заключение значения свойства в кавычки (3 преобразования);
- удаление элемента (16 преобразований).

Использование метаэлементов и регулярных выражений позволило сократить количество правил с 72 до 20: 14 правил потребовалось для замены подстрок, 3 для заключения свойства в кавычки и 3 для удаления элементов. Визуальные описания правил изображены на рисунке 7.

Описание соответствующих конвертеров занимает 293 строки кода на C++. Таким образом, по сравнению со старым механизмом визуальный язык позволяет сделать описания правил более компактными. Кроме того, графическое представление значительно нагляднее, чем императивный код на языке общего назначения.

Заметим, что внутри выделенных групп правила очень похожи. В связи с этим желательным представляется наличие средств переиспользования и параметризации правил, что может оказаться перспективным направлением дальнейшего развития реализованного инструмента.

5.2. Сети Петри

Эволюция сетей Петри – пример, часто используемый [3, 5, 15, 22] для описания и сравнения подходов к миграции.

Исходная метамодель сетей Петри содержит позиции и переходы. При эволюции языка в него добавляются элементы PTArc и TPArc, описывающие явные взвешенные связи от позиций к переходам и от переходов к позициям соответственно. Таким образом, при миграции модели для каждой пары соединённых позиции и перехода между ними должен быть вставлен PTArc или TPArc в зависимости от направления связи. Пример подобного преобразования изображён на рисунке 8.

Реализация описанной миграции с помощью предложенного в данной работе языка состоит из двух правил, представленных на рисунке 9. Согласно [14], для аналогичных преобразований Epsilon Flock требуется 16 строк кода, а Ecore2Ecore – 57 сгенерированных по графическим описаниям строк в формате XMI[4] и 78 строк довольно низкоуровневого кода на Java.

Таким образом, реализованный механизм не уступает существующим средствам миграции, а в сравнении с Ecore2Ecore можно говорить о явном превосходстве предложенного подхода.

[4]XML Metadata Interchange

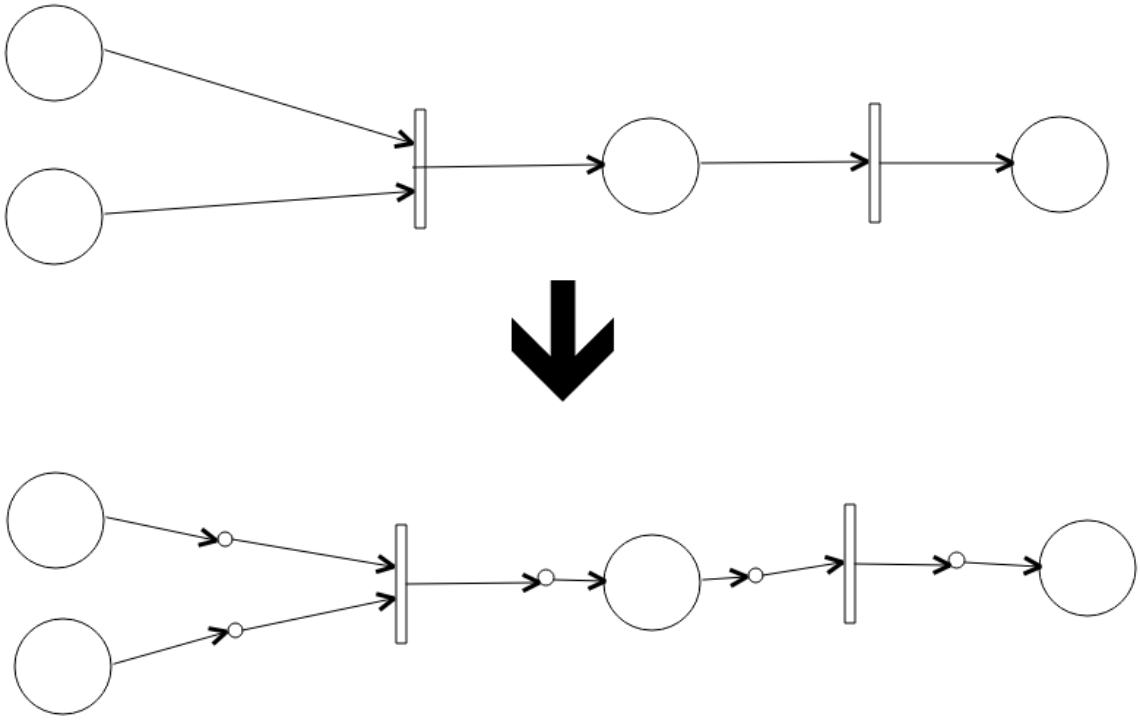

Рис. 8: Преобразование сети Петри.

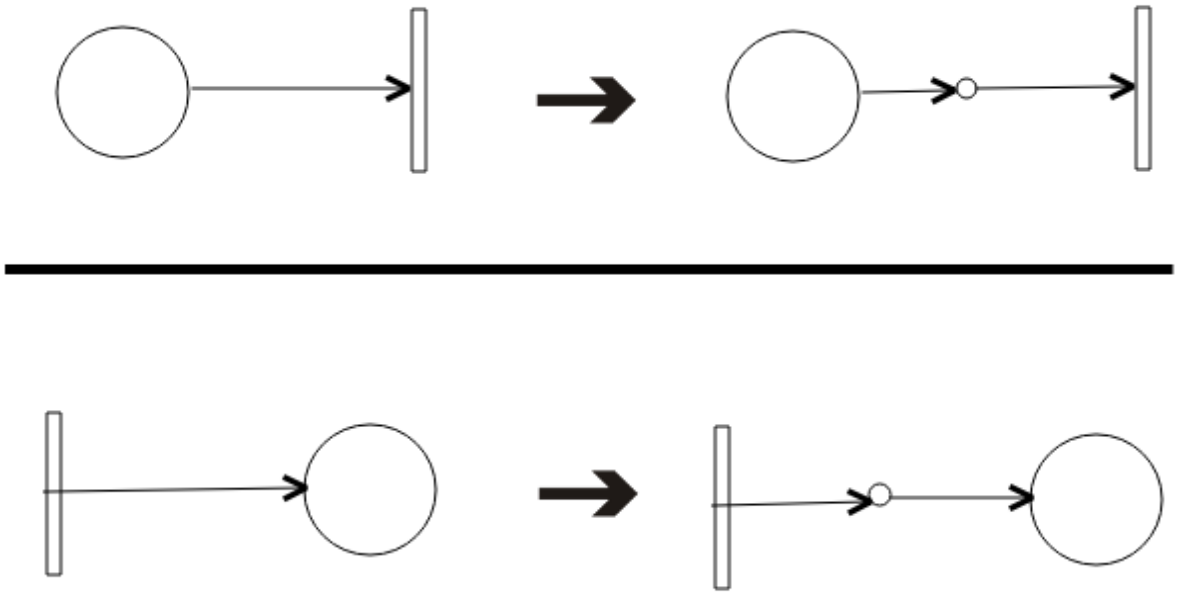

Рис. 9: Правила миграции сетей Петри.

5.3. Соответствие требованиям и ограничения

На основе проведённой апробации можно сделать вывод о соответствии реализованного механизма миграции установленным требованиям.

Использование визуального синтаксиса для описания правил преобразования улучшает наглядность и простоту восприятия правил пользователем, в особенности по сравнению с их спецификацией на языке общего назначения.

Выразительности языка оказалось достаточно, чтобы реализовать оба представленных примера миграции. Однако в более сложных случаях для корректного описания правила могут потребоваться ещё не реализованные средства, такие как указание негативных условий применения. Кроме того, отсутствие контроля над разрешением конфликтов при сопоставлении с шаблоном может привести к потере точности преобразования.

Поскольку изменение модели производится “на месте”, отпадает необходимость в копировании большей части модели.

Для сопоставления с образцом значений свойств элементов пользователь должен знать язык регулярных выражений, что является серьёзным требованием в отношении пользователя, не обладающего навыками программирования. Однако это требование представляется значительно более мягким по сравнению с необходимостью описывать преобразования полностью на C++.

Пользователь имеет возможность управлять применением правил путём изменения их порядка. Помимо этого, он может определить, как должно применяться правило: один раз либо пока его применение приводит к изменениям в модели.

Реализованный механизм совместим как с конвертерами, так и с подсистемой автоматической миграции: обеспечивается сохранение результатов заданных вручную преобразований после применения автоматически выведенных правил.

Поскольку метамодель языка содержит всю необходимую для миграции информацию, преобразования вполне могут производиться и в режиме интерпретации метамодели.

Несмотря на то, что ни одно из средств преобразования моделей в QReal не подходит для описания миграций напрямую, возможным оказалось переиспользовать значительную часть кода, относящуюся к поиску шаблона на графе модели.

Наконец, конкретный синтаксис правил обеспечивает синтаксическую корректность описаний образцов относительно исходного и целевого языков.

Можно заключить, что предложенный подход в целом удовлетворяет обозначенным требованиям, однако возможны дальнейшие улучшения выразительности языка и точности преобразований.

Заключение

В результате данной работы были решены следующие задачи:

- исследованы подходы к описанию преобразований моделей, рассмотрены наиболее популярные средства миграции;
- разработан и реализован подход к спецификации миграций в QReal, учитывающий особенности данной системы;
- получен целостный механизм миграции моделей, сочетающий преимущества ручных и автоматических преобразований;
- проведена апробация, которая показала, что выбранный подход позволяет достичь результатов, сопоставимых с существующими средствами, и улучшить старый механизм миграции моделей в QReal.

Дальнейшие перспективы

Развитие реализованного подхода можно произвести в нескольких направлениях.

Во-первых, выразительность языка может быть улучшена с помощью поддержки отрицательных условий в шаблонах. Эта же возможность позволила бы влиять на разрешение конфликтов при сопоставлении с образцом и, таким образом, добиться большей точности преобразования.

Во-вторых, не лишней может оказаться поддержка произвольных преобразований свойств с использованием скриптовых языков, что предоставило бы более гибкие возможности для преобразования значений.

Также желательно было бы заменить при описании значений свойств регулярные выражения на какой-либо более простой, пусть и менее выразительный, способ описания строковых шаблонов.

Ещё одним полезным дополнением могут быть средства переиспользования правил, что ускорило бы процесс описания правил, похожих на уже реализованные.

Наконец, сложные миграционные стратегии могут потребовать продвинутых средств управления применением правил, к примеру, возможностей описания зависимостей между правилами или дополнительных условий применимости.

Список литературы

[1] Balasubramanian, D., Narayanan, A., Buskirk, C.V., Karsai, G. The Graph Rewriting and Transformation Language: GReAT // Proceedings of the Third International Workshop on Graph Based Tools (GraBaTs 2006). — 2006.

[2] Brambilla, M., Cabot, J., Wimmer, M. Model-Driven Software Engineering in Practice. — Morgan & Claypool, 2012. — P. 182.

[3] Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A. Automating Co-evolution in Model-Driven Engineering // 12th International IEEE Enterprise Distributed Object Computing Conference. — 2008. — P. 222–231.

[4] Czarnecki, K., Helsen, S. Classification of Model Transformation Approaches // 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. — 2003.

[5] Garces, K., Jouault, F., Cointe, P., Bezivin, J. Managing Model Adaptation by Precise Detection of Metamodel Changes // Model Driven Architecture – Foundations and Applications. — Vol. 5562 of LNCS. — Springer Berlin Heidelberg. — P. 34–49.

[6] Herrmannsdoerfer, M., Benz, S., Juergens, E. COPE – Automating Coupled Evolution of Metamodels and Models // ECOOP 2009 – Object-Oriented Programming. — Vol. 5653 of LNCS. — Springer Berlin Heidelberg, 2009. — P. 52–76.

[7] Jouault, F., Kurtev, I. Transforming Models with ATL // Proceedings of the 2005 International Conference on Satellite Events at the MoDELS. — Springer-Verlag, 2006. — P. 128–138.

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

[9] Kolovos, D.S. An Extensible Platform for Specification of Integrated Languages for Model Management : Ph. D. thesis / Kolovos, D.S. ; University of York, United Kingdom. — 2009.

[10] An Overview of QVT : Rep. / Budapest University of Technology and Economics Department of Automation and Applied Informatics ; Executor: Lengyel, L. : 2002.

[11] Mens, T., Czarnecki, K., Gorp, P.V. A Taxonomy of Model Transformation // Proc. Dagstuhl Seminar on "Language Engineering for Model-Driven Software Development". Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl. — Electronic, 2005.

[12] Paternostro M., Hussey, K. Advanced Features of Eclipse Modeling Framework // EclipseCon 2006. — URL: http://www.eclipse.org/modeling/emf/docs/presentations/EclipseCon/EclipseCon2006_EMF_Advanced.pdf (online; accessed: 2015-05-22).

[13] Rose, L.M., Herrmannsdoerfer, M., Mazanek, S., Gorp, P.V., Buchwald, S., Horn, T., Kalnina, E., Koch, A., Lano, K., Schätz, B., Wimmer, M. Graph and model transformation tools for model migration // Software & Systems Modeling. — 2012.

[14] Rose, L.M., Herrmannsdoerfer, M., Williams, J.R., Kolovos, D.S., Garcés, K., Paige, R.F., Polack, F.A.C. A Comparison of Model Migration Tools // Model Driven Engineering Languages and Systems, 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I. — 2010.

[15] Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C. Model Migration with Epsilon Flock // Theory and Practice of Model Transformations. — Vol. 6142 of LNCS. — Springer Berlin Heidelberg. — P. 184–198.

[16] Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C. An analysis of Approaches to Model Migration // Models and Evolution (MoDSE-MCCM) Workshop. — 2009.

[17] Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C., Poulding, S. Epsilon Flock: a model migration language // Software & Systems Modeling. — 2012.

[18] Rozenberg, G. Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations. — World Scientific, 1997.

[19] Sendall, S., Kozaczynski, W. Model Transformation – the Heart and Soul of Model-Driven Software Development // Software, IEEE. — 2003.

[20] Sprinkle, J. Metamodel Driven Model Migration : Ph. D. thesis / Sprinkle, J. ; Vanderbilt University, TN, USA. — 2003.

[21] Tolvanen, J.-P., Pohjonen, R., Kelly, S. Advanced Tooling for Domain-Specific Modeling: MetaEdit+ // Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling, Montreal, Canada. — 2007.

[22] Wachsmuth, G. Metamodel Adaptation and Model Co-adaptation // Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP'07). — Vol. 4609 of Lecture Notes in Computer Science. — Springer-Verlag, 2007. — P. 600–624.

[23] Агапова Т.Ю. Поддержка эволюции визуальных языков в DSM-платформе QReal. — Курсовая работа 3 курса, Санкт-Петербургский государственный университет. — 2014. — URL: http://se.math.spbu.ru/SE/YearlyProjects/2014/YearlyProjects/2014/344/344-Agapova-report.pdf (online; accessed: 2015-05-26).

[24] Кузенкова А.С. Поддержка механизма рефакторингов в metaCASE-системе QReal. — Курсовая работа 3 курса, Санкт-Петербургский государственный университет. — 2012. — URL: http://se.math.spbu.ru/SE/YearlyProjects/2012/YearlyProjects/2012/345/345_Kuzenkova_report.pdf (online; accessed: 2015-05-26).

[25] Литвинов Ю.В., Кириленко Я.А. TRIK Studio: среда обучения программированию с применением роботов // V Всероссийская конференция «Современное технологическое обучение: от компьютера к роботу» (сборник тезисов). — СПб. : ЗАО «Полиграфическое предприятие № 3», 2015. — P. 5–7. — URL: https://robofinist.ru/uploads/2015/Thesis_2015.pdf (online; accessed: 2015-04-29).

[26] Поляков В.А. Средства задания исполнимой семантики визуальных языков в системе QReal. — Дипломная работа, Санкт-Петербургский государственный университет. — 2013. — URL: http://se.math.spbu.ru/SE/diploma/2013/s/PolyakovVladimir_thesis.pdf (online; accessed: 2015-05-26).

[27] Птахина А.И. Интерпретация метамоделей в metaCASE-системе QReal. — Курсовая работа 3 курса, Санкт-Петербургский государственный университет. — 2012. — URL: http://se.math.spbu.ru/SE/YearlyProjects/2012/YearlyProjects/2012/345/345_Ptakhina_report.pdf (online; accessed: 2015-05-26).

[28] Птахина А.И. Эволюция языков при метамоделировании ”на лету” в DSM-платформе QReal. — Дипломная работа, Санкт-Петербургский государственный университет. — 2014. — URL: http://se.math.spbu.ru/SE/diploma/2014/s/PtakhinaAlina_Diploma.pdf (online; accessed: 2015-26-05).

[29] Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. QReal: платформа визуального предметно-ориентированного моделирования // Программная инженерия, №6, С. 11-19. — 2013.