Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Средства создания визуальных интерпретаторов диаграмм в системе QReal

Курсовая работа студента 445 группы
Полякова Владимира Александровича

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

Санкт-Петербург
2012

Оглавление

Введение

В настоящее время большая часть уже существующих и разрабатываемых технологий программирования основана на текстовых языках. Для них создано множество средств, позволяющих существенно повысить скорость и удобство разработки новых программ. Одной из функциональностей таких средств является возможность пошаговой интерпретации и отладки написанного кода.

Распространение этой функциональности на визуальные языки по аналогичным причинам значительно улучшило бы условия работы разработчика. Также визуальная интерпретация диаграмм хорошо структурирует понимание проектировщиком принципов работы создаваемой системы, что, в конечном счёте, в лучшую сторону скажется на её качестве.

Такие визуальные интерпретаторы и отладчики уже существуют в некоторых CAE-системах (например, в Borland Together[1]), однако данный подход надо распространить на metaCASE-средства для того, чтобы работать с произвольными поведенческими визуальными языками. Таким образом необходимо добавить возможность быстрого создания интерпретаторов и отладчиков для разрабатываемых пользовательских визуальных языков.

[1] Borland Together, http://www.borland.com/us/products/together

1 Постановка задачи

Данная работа является логическим продолжением курсовой работы, выполненной в прошлом году [15]. В ней был проведён обзор основных программных продуктов, имеющих похожие на визуальную интерпретацию функциональности, была разработана инфраструктура визуального интерпретатора в системе QReal [16], создан каркас модуля для работы с отладчиком целевого языка генерации. Также были реализованы несколько частных примеров визуальных интерпретаторов.

В первую очередь, задачей курсовой работы этого года является переход от частных примеров и понимания структуры визуального интерпретатора в целом к конкретным методам визуальной формулировки семантики пользовательских языков. Поэтому первой фазой работы было изучение существующего набора материалов по методам визуальной интерпретации и задания семантики для различных языков.

Второй фазой является анализ данных методов на предмет применимости в нашем случае. После этого необходимо провести синтез, по возможности, положительных черт этих методов и создать собственное решение, подходящие для рассматриваемой задачи по отношению к metaCASE-системе QReal.

Создание прототипа с ограниченной функциональностью, соответствующего созданному решению, составляет третью фазу работы. Прототип поможет оценить степень применимости данного подхода и расставит приоритеты в дальнейшем развитии реализации и корректировки самого решения.

2 Обзор существующих методов

Для того, чтобы поведенческие диаграммы можно было интерпретировать, необходимо задать формальную семантику для соответствующего визуального языка. В ходе исследования было выделено несколько основных способов определения и задания этой семантики, рассмотренных ниже.

2.1 xUML

2.1.1 Обзор

Как известно, наиболее сильное распространение визуального проектирования и программирования началось после создания унифицированного языка моделирования UML[2], предназначенного для объектно-ориентированного моделирования в основном в области разработки программного обеспечения.

Чистый UML — это язык проектирования, а не программирования, поэтому для того, чтобы можно было точно определять семантику различных элементов и конструкций, был создан исполняемый UML (Executable UML, xUML [4, 5, 11]). Он является профилем языка UML, т.е. подмножеством языка UML с ясно определённой семантикой для каждого элемента. Притом это подмножество выбрано так, чтобы сделать из UML полноценный визуальный язык общего назначения. Также как и UML, xUML основан на объектно-ориентированном подходе в программировании (т.е. описание системы будет вестись в терминах классов, атрибутов и т.д.).

Семантика элементов xUML записывается при помощи явно определённой семантики действий (Precise defined Action Semantics, PAS), использование которой повышает уровень абстракции относительно текстовых языков программирования общего назначения. PAS обеспечивает независимость как от конкретной целевой платформы или языка, на которых будут исполняться модели, так и от разбора архитектурных вопросов в процессе проектирования. Также, в связи с тем, что семантика элементов зафиксирована, становится возможной однозначная генерация исполняемого кода по модели, а, значит, появляется и возможность интерпретации модели.

При проектировании при помощи xUML в основном используются 3 составляющие:

- диаграмма классов, описывающая модель наблюдаемой системы (преобразование реальных объектов в классы; классы имеют атрибуты, а связи между классами представляются в виде отношений);
- диаграмма состояний, фиксирующая действия, которые совершит активный объект, т.е. изображающая жизненный цикл объекта (представленный в виде конечного автомата);
- язык действий (Action Language, AL), определяющий поведение объекта при проходе каждого состояния в диаграмме состояний, использующий сигналы как способ взаимодействия между объектами и необходимый для создания экземпляров классов, установки связей, выполнения операций над атрибутами и т.п.

[2] UML, http://www.uml.org/

В общих чертах, данный подход основан на представлении поведения системы в виде набора диаграмм состояний для каждого активного объекта. Действия при проходе каждого состояния в таких диаграммах записываются при помощи AL.

Исполняемая часть xUML основывается на понятии действия — конкретная операция, заданная на элементе модели, относящаяся к классу и характеризующая поведение класса после его инициализации, и PAS — фиксированный набор семантических операций, не имеющий синтаксиса и представляющий как бы интерфейс для создания семантики (таким образом, PAS обеспечивает простоту обучения любому AL, являющимися конкретными реализациями PAS: достаточно выучить только один, т.к. остальные будут иметь тот же семантический набор операций).

На данный момент существует довольно много готовых AL (OAL [1], SMALL [2], TALL [11] и т.д.). Являясь реализацией PAS, AL повышают абстрактность данного подхода, а также подчёркивают независимость от конкретных технологий, ведь все низкоуровневые операции и решения принимаются на уровне компилятора модели.

2.1.2 Примеры

Как видно из рисунка 1, все языки действий очень похожи на SQL [3] или "текстовое" описание того, что должно произойти.

```
create object instance d of dog;
d.name = "sparky";
Create object instance h of doghouse;
h.name = "sparkys house";
relate  d to h across R1;
unrelate d from h across R1;
delete object instance d;
delete object instance h;
select any d from instances of dog where .....
select many dogset from instances of d;
```

Рисунок 1: Пример описания действий на языке OAL[3]

Основополагающий UML (Foundational UML, fUML[4]) является другим видом исполняемого UML, но он также основан на тех же принципах, что и xUML. Язык действий для fUML (Action Language for fUML, Alf[5]) является AL, созданным специально для fUML. Следующие два примера взяты из доклада[6] на информационном дне, посвящённом xUML и организованном OMG[7].

[3] Пример взят из [4]

[4] fUML, http://www.omg.org/spec/FUML/

[5] ALF, http://www.omg.org/spec/ALF/

[6] http://www.omg.org/news/meetings/tc/agendas/va/xUML_pdf/Seidewitz_Tutorial.pdf

[7] OMG, http://www.omg.org/

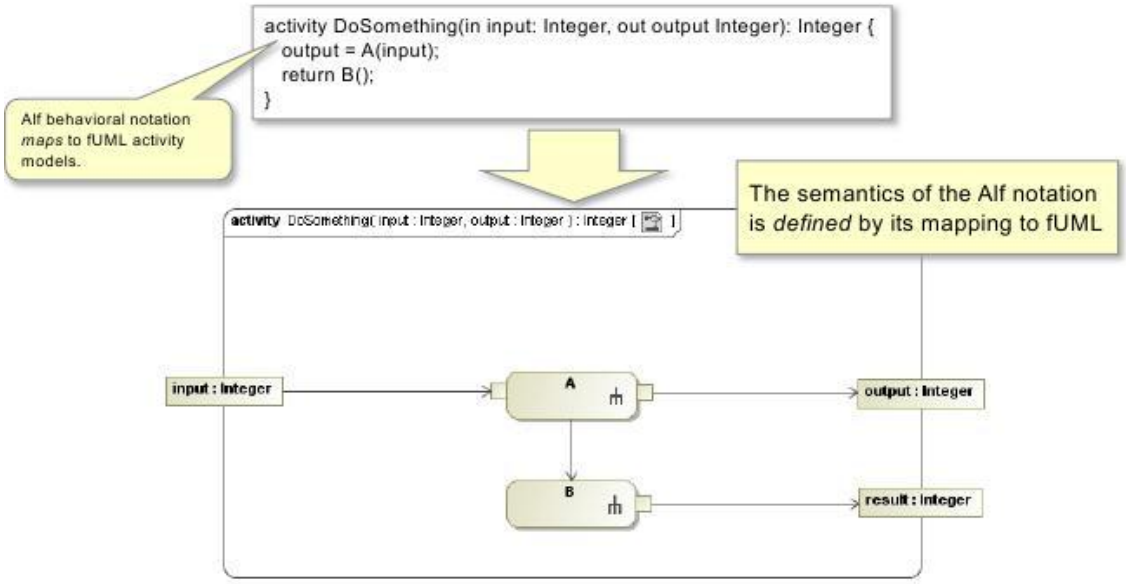

Рисунок 2: Пример описания поведения в связке fUML + Alf

Нотация определения поведения объекта в языке Alf определяется её соответствием диаграмме активностей, записанной на языке fUML, что иллюстрирует рисунок 2.

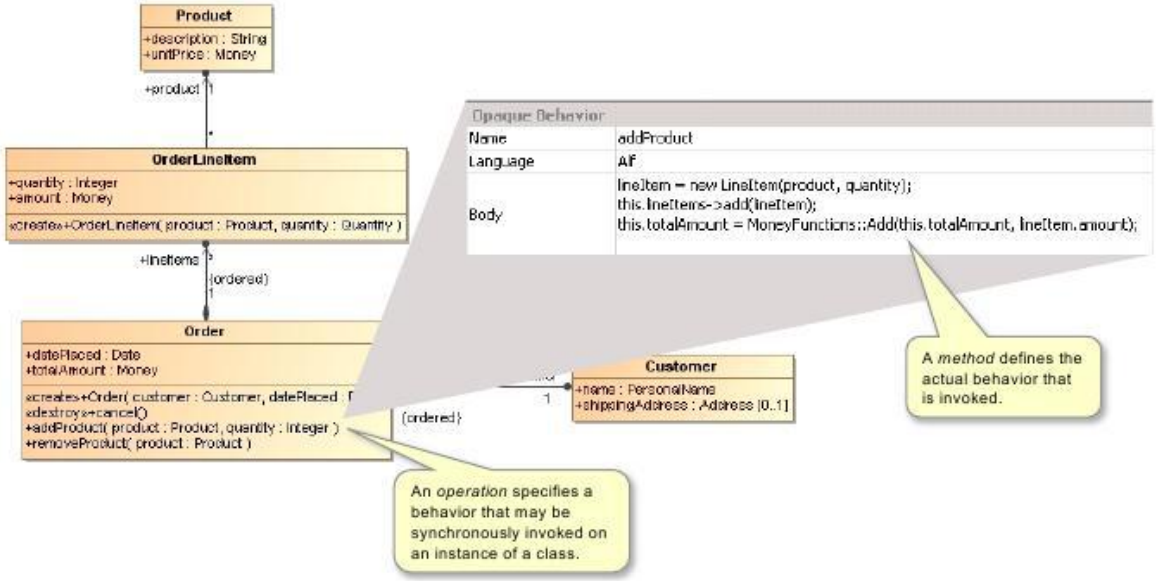

Рисунок 3: Пример диаграммы классов вместе с описанием возможных действий объектов в связке fUML + Alf

2.1.3 Анализ

Исходя из описания xUML, можно заметить, что данный подход основан на понятиях объектно-ориентированного подхода в программировании, а также на стандартных конструкциях большинства языков (например, ветвления, циклы и т.п.), поэтому существенного повышения уровня абстракции на практике он не даёт. Задание поведения системы в xUML представляет собой набор диаграмм состояний, в то время как в общих случаях у системы бывает более сложное поведение, нежели конечный автомат.

Также иногда при визуализации потока управления программы для простых переходов между операциями необходимо создавать направленные связи между соответствующими элементами модели, в то время как в текстовых языках общего назначения это будет просто блочный оператор (например, фигурная скобка) или переход к следующей операции (например, точка с запятой), что приводит к нарушению главнейшей парадигмы предметно-ориентированного подхода в программировании.

2.2 «Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Language»

2.2.1 Обзор

В данной кандидатской диссертации [9] Ян Хаусман (Jan Hausmann) из университета в Падерборне (Германия) исследовал вопрос задания семантики для визуальных языков моделирования и предложил новый подход к решению этой задачи: Dynamic Meta Modeling. Проведённое исследование изложено с фокусом на UML, т.к. он объединяет основные понятия многих визуальных языков и понятен для широкой аудитории.

Структурно, в работе можно выделить три главные составляющие: анализ вопроса в целом, описание предложенного способа и его применение для UML Activity Diagram.

2.2.1.1 Описание вопроса в целом

В первой части рассмотрен вопрос, в чём состоит сложность задания семантики для визуального языка и почему важно иметь именно формальное определение семантики в отличие от текстуального. Как показывает практика языка UML, полагающегося на текстовое описание смысла каждой диаграммы, при неформальном описании существуют различные двусмысленности и фактические недостатки в спецификации, которые порождают многочисленные споры по поводу правильной интерпретации созданных моделей. Формальное описание же позволяет этого избежать, а также облегчает создание средств автоматической обработки диаграмм (например, генератор кода, различные анализаторы и верификаторы и т.п.). Значительно упрощается обучение этой технологии будущих пользователей, т.к. она становится более похожа на язык программирования.

В ней также сформулированы требования к разрабатываемой технике задания семантики. Формальность позволит автоматически анализировать создаваемые модели и искать в них ошибки по заданной спецификации. Для более детального анализа моделей необходимо, чтобы в технике не было большего количества мощных ключевых моментов, которые нужно учитывать и использовать при описании семантики, т.к. их наличие существенно затруднит анализ.

Точность поможет избежать проблем неоднозначности интерпретации семантики. Понятность для целевой аудитории является ключевым требованием к создаваемому подходу, т.о. визуальное описание более предпочтительно, нежели текстовое. Адекватность представления уменьшит величину семантического разрыва (semantic gap), возникающего, к примеру, при попытке представления высокоуровневых особенностей языка UML через какую-то базовую математическую модель. Разрабатываемая технология также должна обладать свойством универсальности, позволяющим применять её к любым визуальным языкам.

Дальше в первой части рассмотрен вопрос применимости денотационного и операционного способа описания семантики именно для визуальных языков. Введены

понятия синтаксической и семантической области, семантического соответствия (синтаксическому элементу при помощи семантического соответствия сопоставляется семантический элемент, отражающий его смысл и поведение).

Характеристиками денотационного подхода являются: чёткое разделение синтаксической области, семантической области и семантического соответствия, семантическая область состоит из математических функций, соответствие осуществляется индуктивно через лямбда-выражения. Данный подход довольно удобен в применении, но часто слишком сложен, поэтому он остаётся лишь теоретическим средством задания семантики.

Развитием денотационного подхода является компиляционный (трансляционный), основанный на переводе выражений на исходном языке (синтаксическая область) в уже известный другой (семантическая область). Данный подход является более практическим, т.к. не содержит математической строгости, но по-прежнему остаётся формальным.

В качестве семантической области в компиляционном подходе были рассмотрены преобразования графов и ядровая семантика (core semantics, [8, 13]).

В первом случае существует бременский подход (The Bremen approach, [10]), имеющий явные недостатки в виде отсутствия адекватности использования преобразований графов, а также в виде недостатка точности и понятности. Тем не менее, преобразования графов выглядят подходящим способом описания семантической области, т.к. обладают большинством перечисленных выше требований.

Во втором случае определяется подмножество синтаксической области, и семантика задаётся только для него. Данный подход обладает высокой понятностью, но не может охватить весь спектр возможностей UML и поэтому не обладает адекватностью и универсальностью.

Операционный подход определяет, что элемент делает, а не то, чем он является, как в денотационном. Исходя из этого, операционная семантика определяется в терминах правил и состояний системы. Каждое правило состоит из предусловий и набора операций по изменению текущего состояния. Данный подход является понятным и хорошо анализируемым, но также обладает и набором недостатков.

Среди них можно выделить следующие. Модели не всегда имеют форму дерева, что значительно усложняет интерпретацию и не гарантирует её завершение. У модели может отсутствовать начальное состояние, в отличие от текстовых языков программирования. Визуальное представление семантики в большинстве случаев получается не очень адекватным, но уже существует решение этого вопроса в виде графической операционной семантики (Graphical Operational Semantics, GOS, [7]), основанной на преобразованиях графов. Неполнота моделей может привести к существенным ошибкам при интерпретации.

2.2.1.2 Предлагаемое решение

В связи с тем, что ни один существующих и рассмотренных подходов не смог целиком удовлетворить требованиям, сформулированным выше, Хаусман предложил своё решение — Dynamic Meta Modeling. Этот подход обладает высокой степенью точности и визуальности, понятности и адекватности в задании семантики и опирается на понятие семантики действий (action semantics), которая совместила в себе как денотационную, так и операционную семантику.

Общий принцип DMM-подхода изображён на рисунке 4.

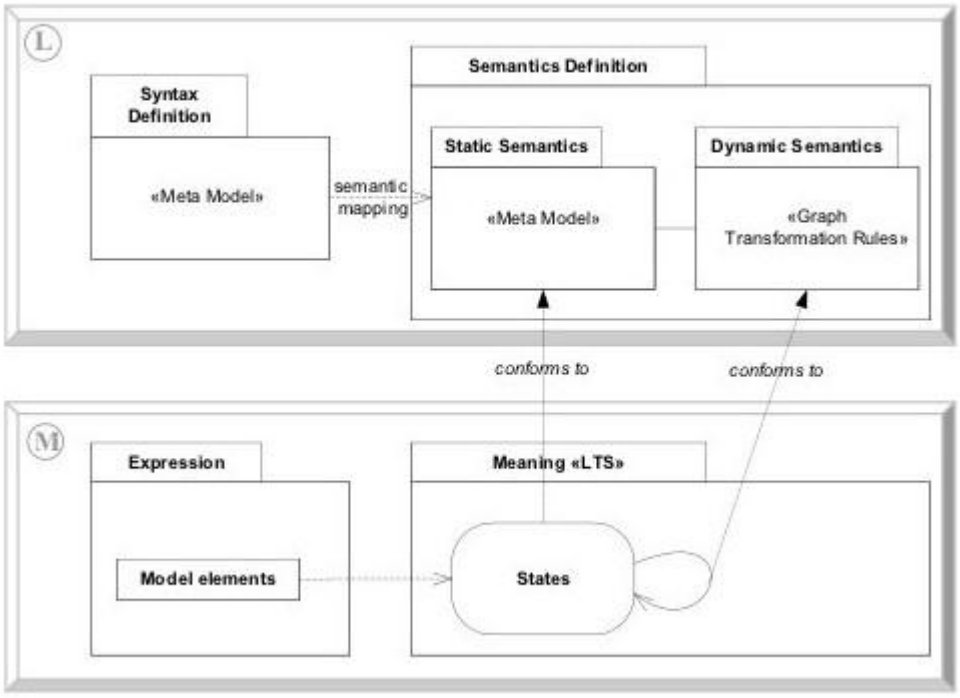

Рисунок 4: Общая схема DMM-подхода

В работе присутствует детальное формальное описание каждой составляющей части данной схемы, здесь же будут рассмотрены лишь основные моменты.

Во-первых, для всех визуальных языков существует разделение на два уровня: уровень языка и уровень модели (в русскоязычной литературе это эквивалентно уровню метамодели и уровню модели соответственно). На уровне языка идёт описание синтаксиса языка и прочих его особенностей, на уровне модели — создание конкретной диаграммы на этом языке. Данные уровни помечены на рисунке буквами “L” и “M”.

Во-вторых, благодаря этому разделению компоненты на уровне модели должны быть согласованы с компонентами на уровне языка, т.е. выражение и его элементы модели должны соответствовать синтаксической модели языка, семантическое соответствие на уровне модели должно подходить под семантическое соответствие на уровне языка, конкретная реализация поведения системы должна быть согласована с определением семантики на уровне языка.

В качестве синтаксической области и её дальнейшего представления на уровне модели в предложенном подходе может выступать любой визуальный язык, что обеспечивает универсальность данной технологии.

Для реализации семантического соответствия Хаусман предложил концепцию мета отношений (Meta Relations). Она сама, а также причины её создания подробно описаны в работе. Можно сказать, что она значительно сложнее, чем простое соответствие одного синтаксического элемента семантическому, т.е. его поведению, т.к. даже при таком способе необходимо учитывать вложенные соответствия, показанные на рисунке 5.

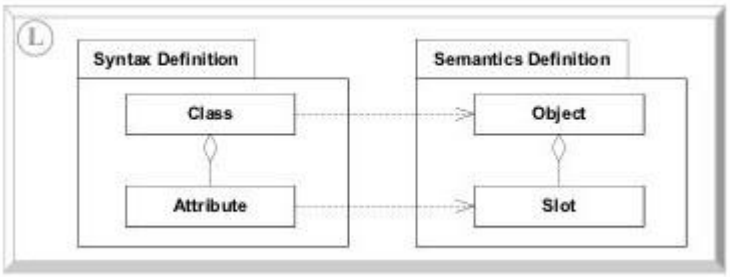

Рисунок 5: Пример вложенного соответствия

Семантической областью в подходе, предложенном Хаусманом, являются преобразования графов, как наиболее подходящие под все требования. Семантика же в данном подходе состоит из двух частей: статической и динамической. Статическая семантика на уровне языка представляет из себя метамодель для описания набора состояний системы, а также доступных правил динамической операционной семантики без конкретной реализации. Семантическое соответствие синтаксической области на статическую семантику языка составляет денотационную часть данного подхода.

Динамическая семантика представляет собой набор правил преобразований графов и составляет операционную часть данного подхода. В итоге, на уровне модели имеем конечный автомат (Labeled Transition System, LTS), состояния которого соответствуют состояниям, описанным в статической семантике на уровне языка. Правила перехода же между этими состояниями являются правилами преобразования графов, которые реализованы на уровне языка, согласованы с статической семантикой и удовлетворяют синтаксическим особенностям создания таких правил, о которых будет рассказано далее.

2.2.1.3 Преобразования графов

Большое место в работе занимает формальное описание сути правил преобразования графов и их дальнейшее сужение для применения в DMM-подходе. Рассмотрим основные понятия оттуда.

В этом подходе модель на визуальном языке, для которого задаётся семантика, рассматривается в виде типизированного мультиграфа с атрибутами и метками на узлах и рёбрах, допускающего наследование, связанного с возможностью расширения классов узлов и рёбер.

В общем виде, правило преобразования графов имеет в себе левую и правую часть. В исходном графе ищется подграф, совпадающий с левой частью, и заменяется на граф из правой части. Для удобства левую и правую часть скрестили в одну, добавив к каждому добавленному элементу метку <new>, а к каждому удалённому — метку <deleted>.

Также к этому были добавлены следующие расширения: отрицающие применение условия (Negative Application Conditions, NAC) и универсальное замыкание (Universal Quantification, UQ). Использование NAC сводится к простому добавлению к правилу перечеркнутого элемента, означающего, что правило может быть применено только в случае отсутствия данного элемента в графе. Наличие же в правиле универсальной структуры (Universally Quantified Structures, UQS) означает, что правило будет применено сразу ко всем подходящим местам в исходном графе.

Для удобства использования у каждого правила есть имя и, возможно, базовый элемент, на котором оно должно применяться, а также список параметров. Это даёт возможность внутри создаваемых правил вызывать уже созданные правила (invocations). Общая сигнатура как правила, так и вызовов других правил приведена на рисунке 6.

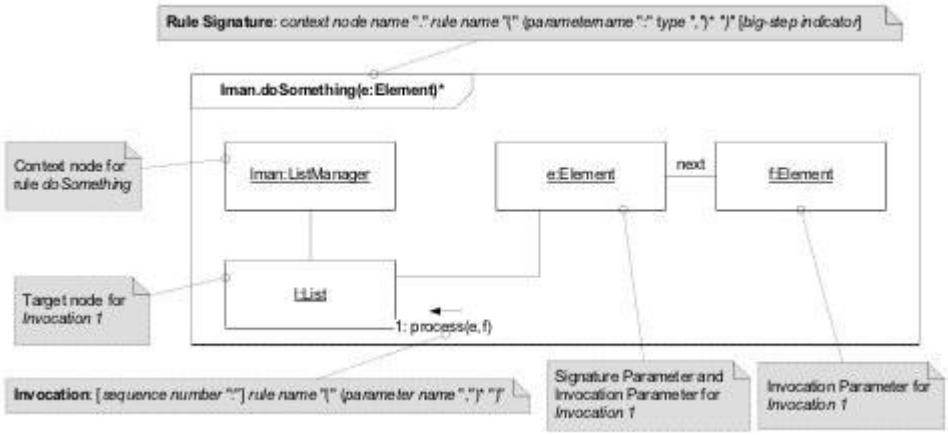

Рисунок 6: Сигнатура правила преобразования графов

В работе также приводятся рассуждения по поводу необходимости удалять все рёбра, связанные с узлом, при его удалении или же отрицать применение правила, наведён формализм в возможности применения вызовов в различных частях правила, а также рассмотрена общая процедура работы применения этих правил к исходному графу. Подробнее об этом можно прочитать в самой статье. Примеры правил преобразования графов можно посмотреть в пункте “2.2.2 Примеры”.

2.2.2 Примеры

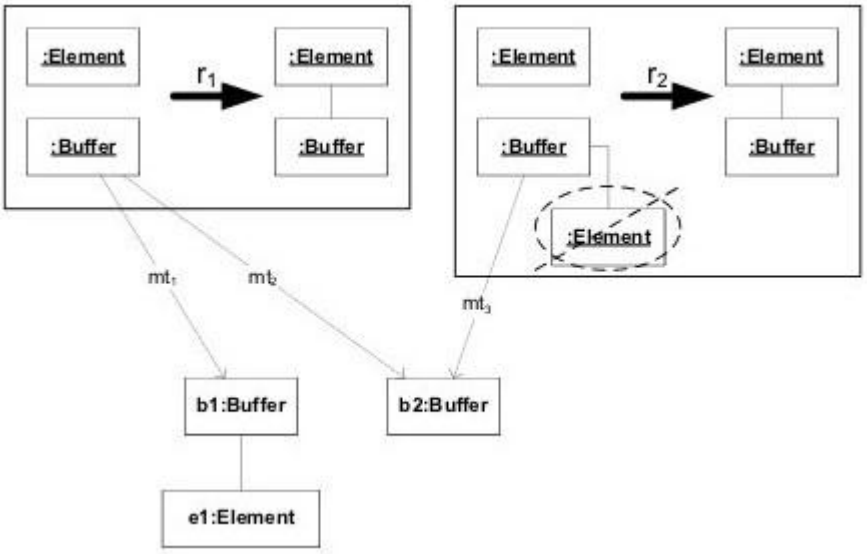

Рисунок 7: Пример NAC

Рисунок 7 иллюстрирует применение NAC. В то время как первое правило сработает в обоих случаях, второе — только во втором.

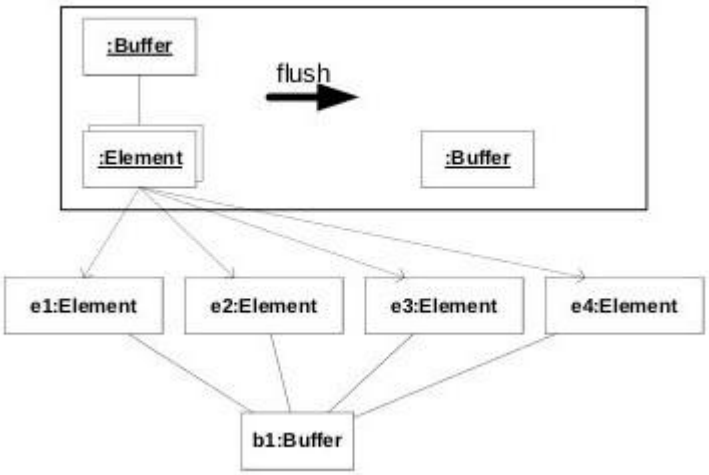

Рисунок 8: Пример правила с UQS

Правило на рисунке 8 показывает способ использования UQS для очистки буфера. При применении данного правила буфер очистится сразу целиком.

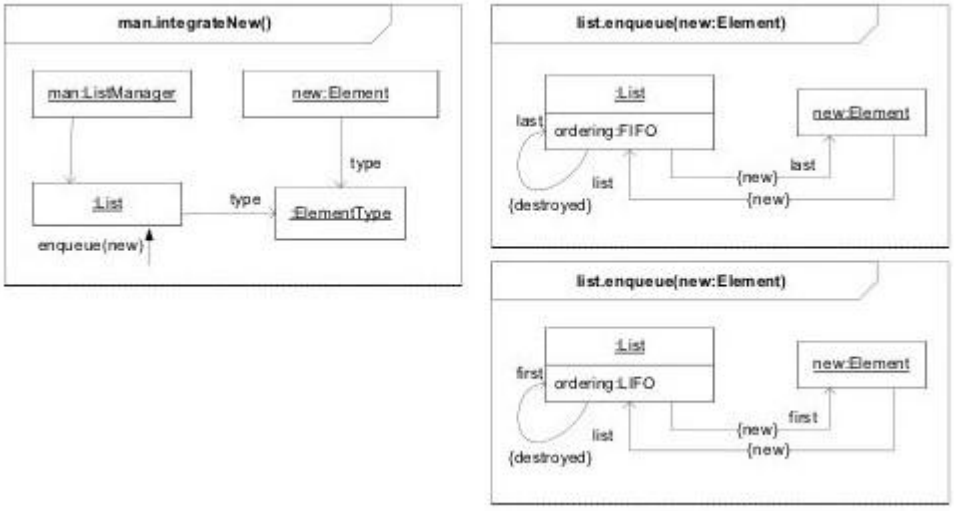

Рисунок 9: Пример использования вызовов других правил

Набор правил на рисунке 9 содержат правила для добавления очередного элемента в список. Есть ListManager и набор списков. Левое правило сначала при помощи ListManager'а выбирает нужный список, потом вызывает правило enqueue, которое и добавляет элемент в список с учётом нужного порядка (FIFO или LIFO).

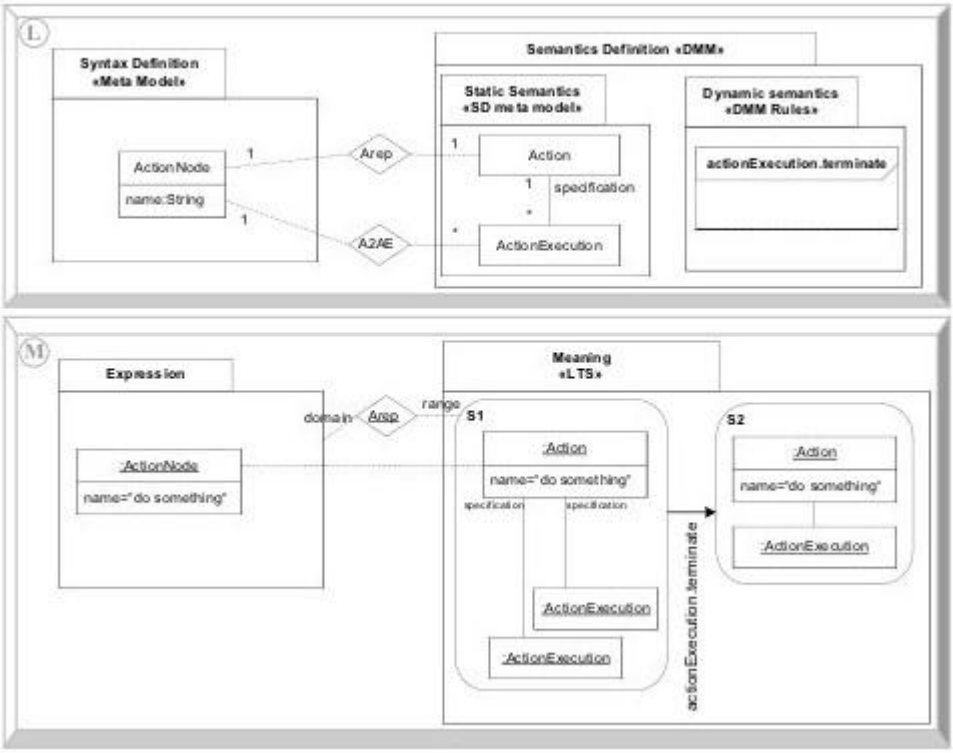

Рисунок 10: Пример перехода LTS из одного состояния в другое при применении правила

На рисунке 10 изображено общая концепция задания семантики для элемента ActionNode диаграммы активностей (вместе с Meta Relations), а также пример перехода LTS уровня модели из состояния S1, когда система исполняет действие дважды параллельно, при помощи правила terminate в состояние S2, где есть только одно активное исполнение данного действия.

2.2.3 Анализ

DMM-подход является очень мощной технологией задания семантики визуальных языков, имеющей под собой долгий анализ предметной области и существующих методик. Он формален, точен, универсален и понятен, имеет высокую наглядность как в задании семантики, так и в процессе интерпретации модели по ней.

Среди главных недостатков этого подхода можно выделить высокую алгоритмическую сложность, связанную с задачей поиска подграфа в графе, которая является NP-полной, а также то, что этот подход так и не был реализован.

2.3 «Visual interpreter and debugger for dynamic models based on the Eclipse platform»

2.3.1 Обзор

Дипломная работа Нилса Банденера (Nils Bandener) [6] из университета в Падерборне (Германия) посвящена созданию визуального интерпретатора и отладчика для поведенческих визуальных языков для платформы Eclipse[8]. В ней, в первую очередь, рассматривается именно инфраструктура интерпретатора и отладчика, а также приводятся примеры возможного пользовательского интерфейса. Сама реализация программного средства, подходящего для произвольных диаграмм поведенческого типа, отсутствует.

Для представления семантики визуальных языков в данной работе используется DMM-подход, описанный ранее. Большинство примеров являются диаграммами активностей (UML Activity Diagram) с семантикой на основе DMM, предложенной Хаусманом в своей работе [9].

Пример интерпретации простой такой диаграммы приведён на рисунке 11.

[8] Eclipse, www.eclipse.org/

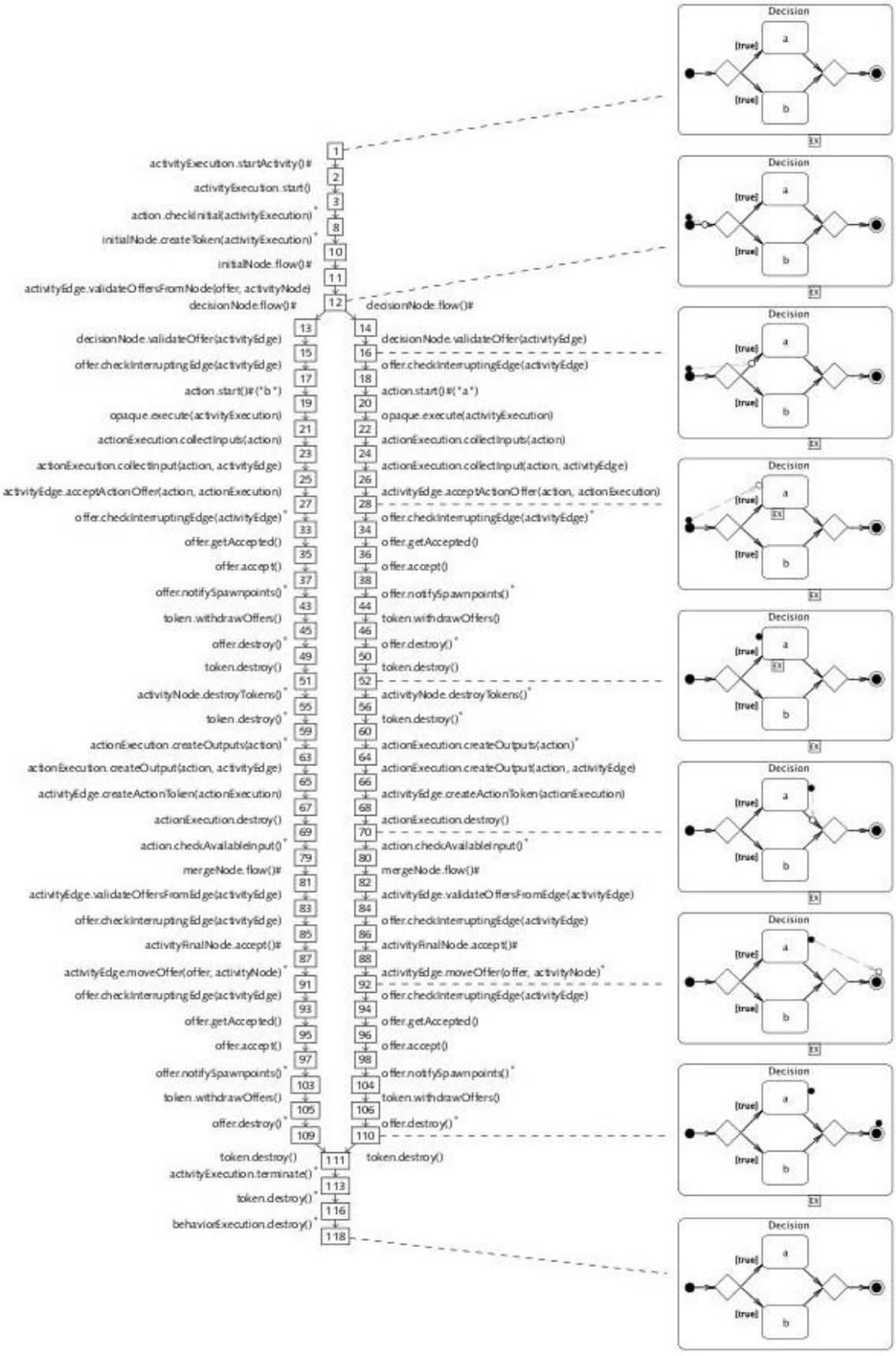

Рисунок 11: Пример интерпретации диаграммы активностей

Для облегчения реализации DMM-подхода большое место в работе занимает анализ уже существует программных средств с целью их дальнейшего применения. Например, Eclipse Modeling Framework, EMF[9], для описания общей метамодели визуального языка, Graphs for Object-Oriented Verification, GROOVE[10], для осуществления преобразований графов, Graphical Modeling Framework, GMF[11], для графического описания создаваемых визуальных языков. Для реализации процесса отладки рассмотрено возможное использование Eclipse Plugin for Prototyping Visual Interpreters and Debuggers, EProvide[12], основанный на Eclipse Platform Debug, EPD[13], и объяснено, почему он не подходит для поставленной задачи.

Ещё одним важным моментом в работе является изучение вопроса самого процесса разработки семантики. В рамках этого вопроса был рассмотрен процесс спецификации через тестирование (test-driven specification process), предложенный в [12]. В этом процесса есть 2 фазы. Во время первой надо покрыть все языковые конструкции примерами с чётко прослеживающийся логикой, во время второй - нужно покрыть только отдельные элементы языка, также во время этой фазы можно расширять семантическую область. После этого примеры превращаются в тесты, и дальнейшие изменения семантики сначала проверяются на них. Рисунок 12 иллюстрирует данный алгоритм.

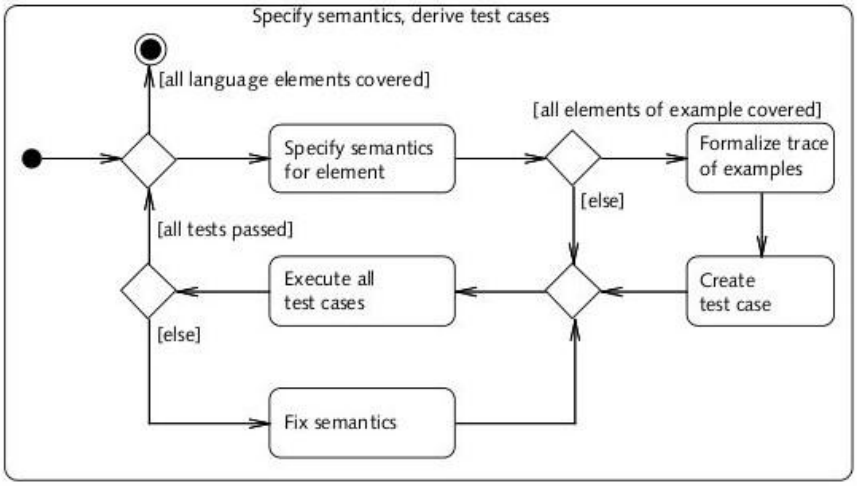

Рисунок 12: Процесс спецификации через тестирование

Упрощённый процесс создания конкретной модели на визуальном языке, для которого доступна семантика, представлен на рисунке 13.

[9] EMF, http://www.eclipse.org/modeling/

[10] GROOVE, http://groove.sourceforge.net/groove-index.html

[11] GMF, http://www.eclipse.org/modeling/gmp/

[12] EProvide, http://eprovide.sourceforge.net/Welcome.html

[13] EPD, http://www.eclipse.org/eclipse/debug/index.php

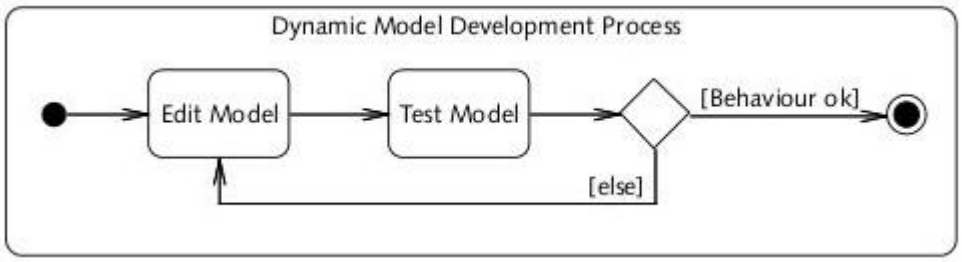

Рисунок 13: Процесс создания модели на языке с доступной семантикой

Также в работе был проанализирован набор основных функциональностей для удобства интерпретации и отладки визуальных языков. Например, рассмотрена работа с моделями, имеющими несколько путей исполнения, подробно описаны понятия точек останова и контрольных точек данных.

Сформулированы и разобраны другие характерные для средств отладки требования к функциональности. Более подробно с случаями применения визуального интерпретатора и отладчика можно ознакомиться на рисунке 14.

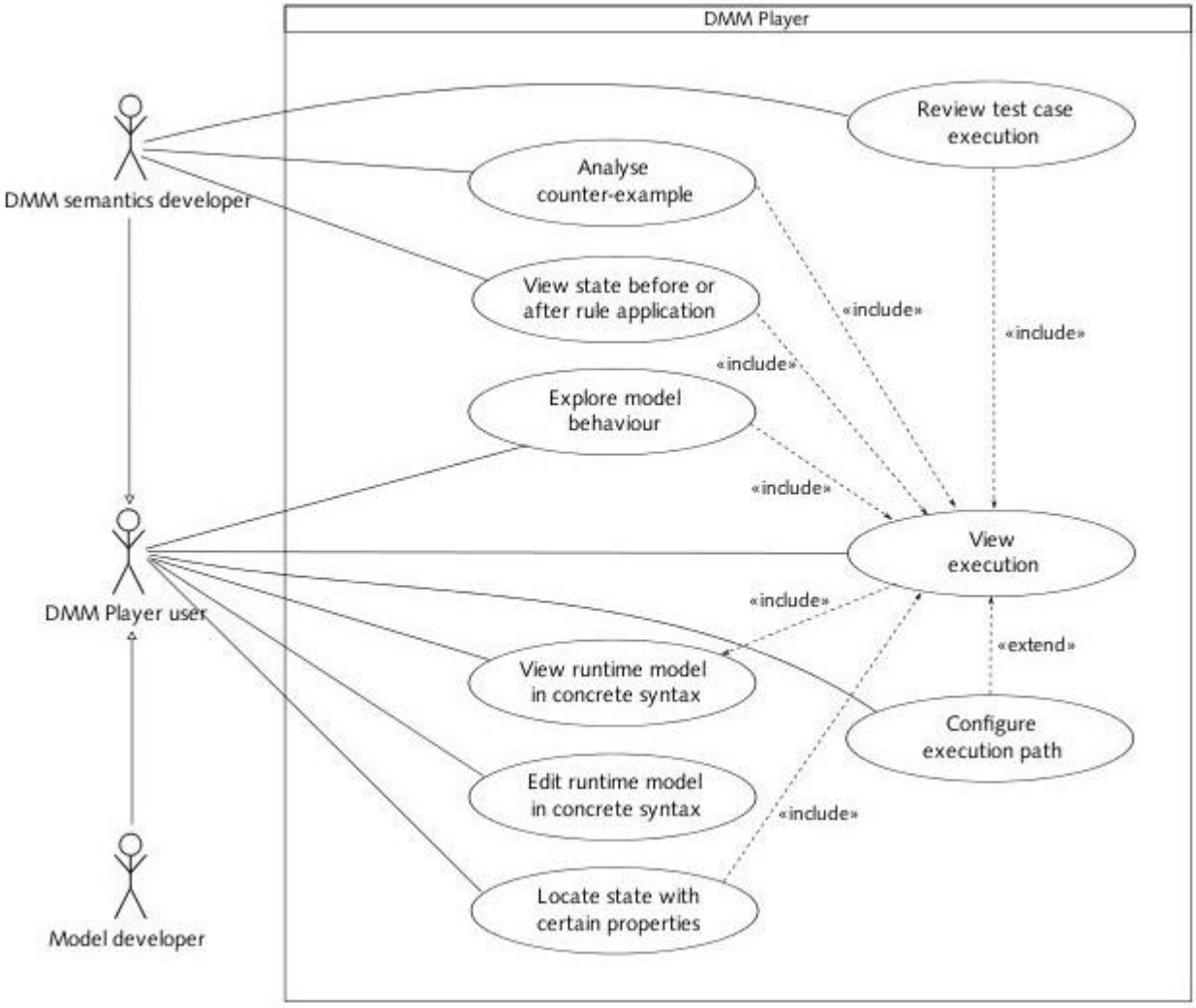

Рисунок 14: Use-case диаграмма для разрабатываемого средства визуальной отладки DMM Player

В итоге, исходя из необходимой функциональности и анализа программных средств, была предложена инфраструктура разрабатываемого DMM Player'a в виде диаграммы компонент и описан общий алгоритм его работы, а также приведены примеры возможной реализации графического пользовательского интерфейса для него.

2.3.2 Примеры пользовательского интерфейса

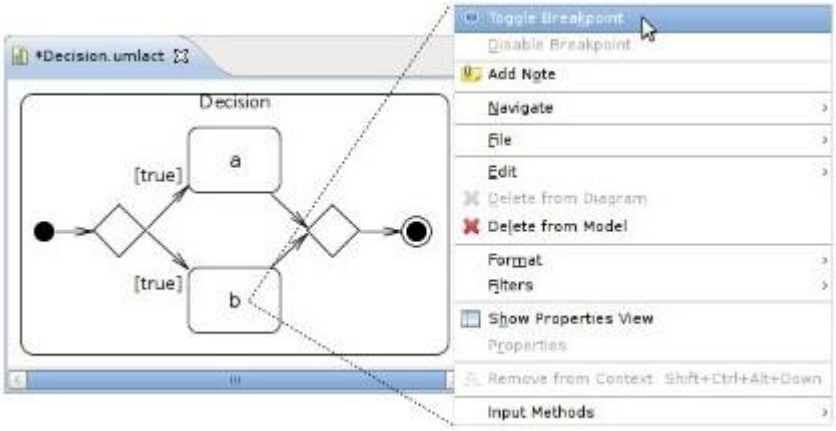

Рисунок 15: Пример задания синтаксической точки останова

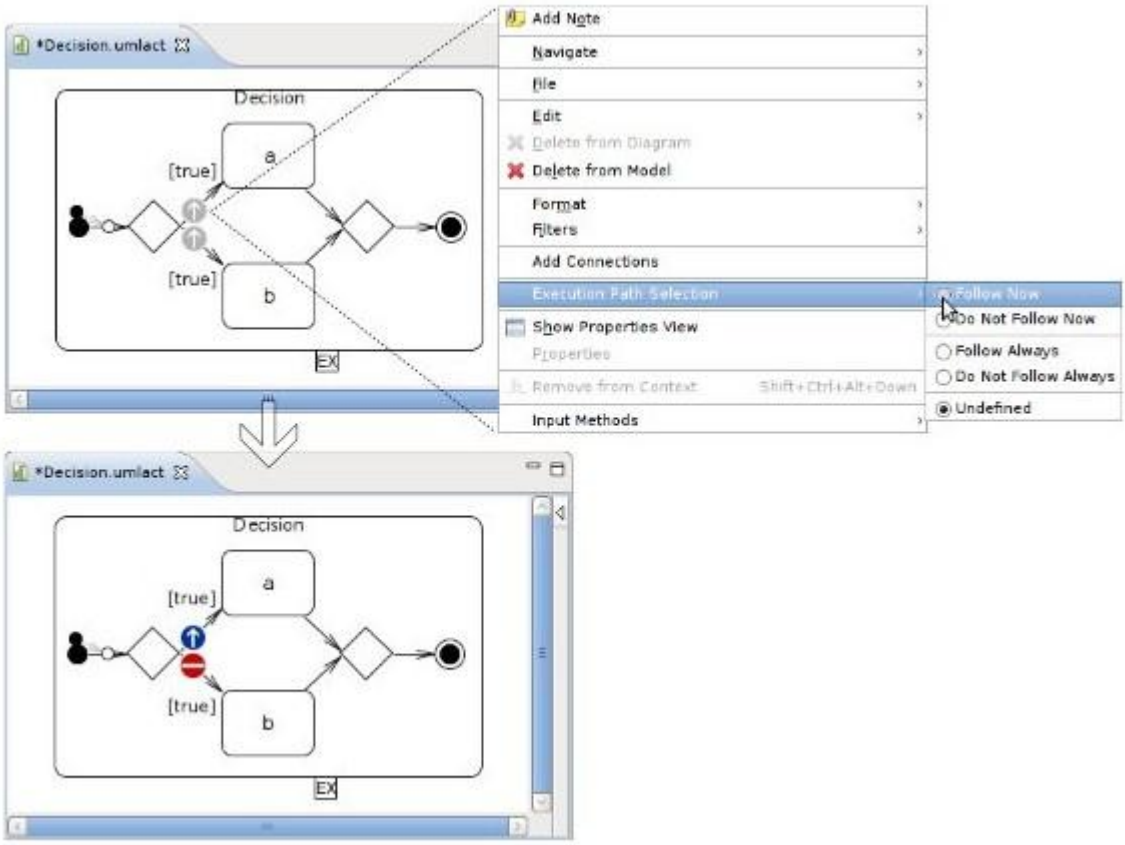

Рисунок 16: Пример настройки пути исполнения

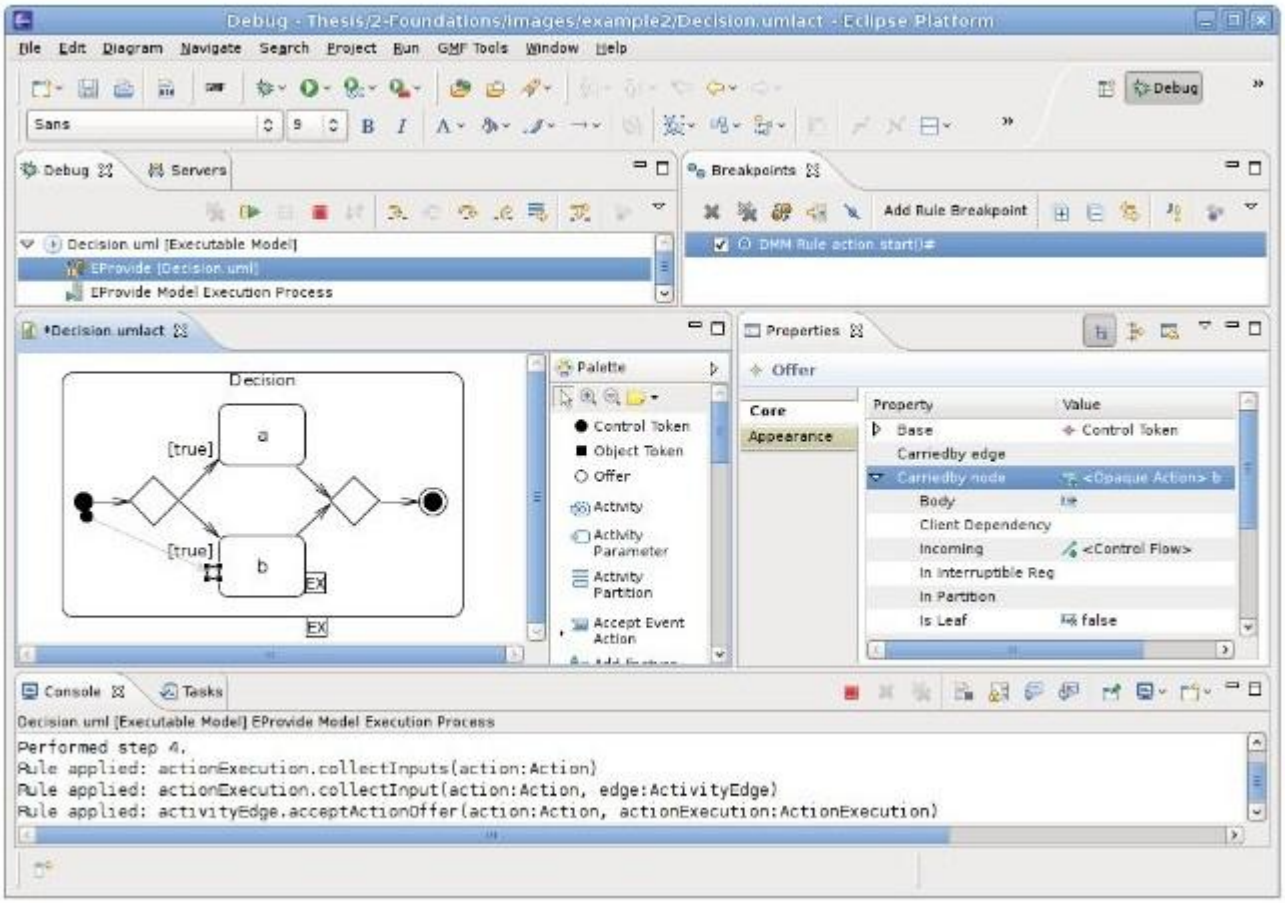

Рисунок 17: Пример работающего DMM Player'а

Рисунок 17 иллюстрирует работу DMM Player'а в отладочном режиме с настройками отладки, отображением точек останова, логом текущего исполнения модели и редактором свойств для выделенного элемента (offer'а).

2.3.3 Анализ

В контексте поставленной ранее задачи данная работа представляет интерес наличием чётких формулировок необходимых требований, функциональностей и возможных проблем при создании визуального интерпретатора и отладчика, а также способов их разрешения. Примеры пользовательского интерфейса облегчат его создание в системе QReal.

Также возможен поиск аналогов упомянутых в работе программных средств, интегрирующихся не только с платформой Eclipse, и их дальнейшее использование.

3 Описание предложенного решения

По результатам исследования и анализа существующих способов задания семантики визуальных языков был предложен собственный подход, разработанный применительно к metaCASE системе QReal. Он основан на преобразованиях графов, как и DMM-подход, а также имеет некоторое подобие языка действий из xUML. В нём можно выделить три основных момента, рассмотренных ниже.

3.1 Семантическая модель

В системе QReal существует разделение логической и графической модели при представлении и хранении данных. Так, например, некоторые элементы модели могут несколько визуальных (графических) представлений или вообще не иметь либо визуального представления, либо логического. Но всегда верно, что у одного графического представления элемента есть не более одного логического, а логический элемент может как иметь много графических представлений, так и не иметь ни одного. Поэтому пользователь может легко создавать различные визуальные представления для одной и той же логической модели. Поддержка всего этого в QReal осуществляется при помощи специального API графической и логической модели.

К этому разделению необходимо добавить новую семантическую модель, доступ к которой будет осуществляться посредством нового API семантической модели. Данная модель должна определяться в специальном редакторе, похожем на редактор для задания метамодели языка, который будет динамически генерироваться, компилироваться и загружаться в систему на основе исходной метамодели.

В нём будет, в первую очередь, целиком содержаться копии всех элементов метамодели визуального языка, для которого необходимо определить семантику. Также данный редактор будет содержать стандартный набор средств по добавлению новых атрибутов к существующим элементам и созданию новых элементов.

При работе с таким редактором к любому элементу из исходной метамодели можно будет добавить дополнительные семантические атрибуты. Также к ним могут быть добавлены некоторые процедуры над семантическими и логическими атрибутами, записанные либо на известном текстовом языке, либо на специальном языке, созданном для блок-схем в рамках курсовой работы прошлого года [15]. Данные процедуры являются возможным поведением элемента, а вместе с семантическими атрибутами они будут составлять семантику элемента.

Помимо расширения исходной метамодели языка при помощи добавления семантики к её элементам в редакторе к ней также можно добавлять новые семантические узлы и связи.

В результате работы с редактором семантической модели для выбранного визуального языка в системе должна генерироваться новая метамодель, содержащая все элементы исходного языка вместе с их семантикой, а также дополнительный пользовательский набор новых семантических связей и узлов. Впоследствии именно с

помощью этой метамодели придётся создавать модель, подлежащую интерпретации. Семантические атрибуты и поведение при её использовании в обычном редакторе показываться не будут.

В качестве примера создания такой метамодели с семантикой было произведено расширение языка блок-схем путём добавления новых атрибутов к элементам InitialNode, Action и Condition, которое будет рассмотрено позднее более подробно.

3.2 Редактор семантики визуального языка

Процесс визуализации интерпретации будет происходить при помощи применения к исходной модели различных правил преобразований графов, которые уже упоминались ранее. Их применение будет осуществляться по следующему принципу.

Существующий набор правил складывается в хеш-таблицу, ключом которой является название правила, а значением его содержимое. После этого запускается цикл, перебирающий каждый ключ в этой таблице и проверяющий, найдётся ли для правила с таким названием искомый подграф в модели. Если нашёлся, то данное правило применяется к найденному подграфу. Алгоритм поиска подграфа в модели будет описан позднее. Интерпретация будет происходить до тех пор, пока применимо хотя бы одно правило. Принцип работы такого процесса напоминает выполнение алгоритмов Маркова на строках.

Задаваться эти правила преобразования графов будут в специальном редакторе семантики исходного визуального языка, который будет рассмотрен далее.

3.2.1 Описание

Данный редактор семантики должен динамически генерироваться, компилироваться и загружаться в систему по полученной в результате работы с редактором семантической модели новой метамодели. Он будет содержать все элементы из новой метамодели, а также автоматически добавленный фиксированный набор новых элементов и новых атрибутов к уже существующим элементам.

Из новых атрибутов можно выделить метку семантического статуса элемента в правиле, по которой в дальнейшем будет определяться, нужно ли этот элемент удалить (метка “deleted”), создать (метка “new”) или оставить без изменений (метка не проинициализирована). В набор новых элементов входят: элемент «Rule», связь «Replacement» и различные унификаторы.

Элемент «Rule» является контейнером для всех элементов редактора семантики и будет целиком содержать ровно одно семантическое правило, т.е. правило преобразования графов и реакцию на него, описание которой будет дано позже. Направленная связь «Replacement» означает замену одного элемента на другой в модели при применении правила. Унификаторы же являются обобщёнными элементами и бывают двух типов: узла и связи. Сравнение элемента модели с ними всегда возвращает истину.

При составлении правил преобразования графов прямо в редакторе атрибутов можно ставить условия на их значение. Также в редакторе семантики отображаются и

семантические атрибуты, на которые можно ставить ограничения для обеспечения правильной интерпретации моделей.

В идеале необходимо обеспечить поддержку всех описанных в DMM-подходе возможных особенностей правил преобразования графов от вызовов в правиле других правил до отрицающих применение условий и универсального замыкания. Но на данный момент достаточно ограничиться просто созданием, заменой и удалением элементов, а также наличием унификаторов.

3.2.2 Пример

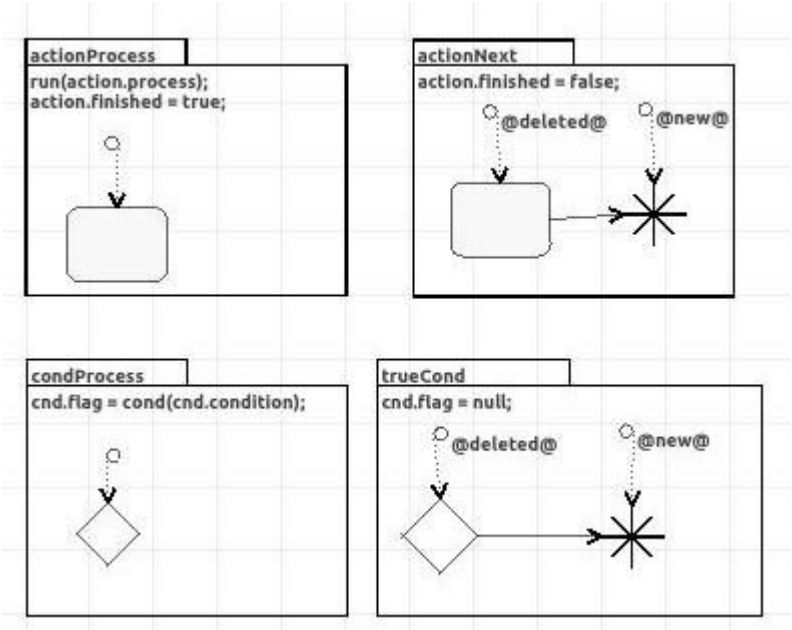

Рисунок 18: Пример правил преобразования графов

На рисунке 18 изображён пример задания правил преобразования графов в редакторе семантики для языка блок-схем. Там присутствуют 4 из 8 правил полной семантики блок-схем (целиком она будет разобрана позднее), но даже по ним сразу видны характерные особенности данного редактора.

Например, удаляемые и создаваемые элементы имеют показывающиеся на экране семантические метки, наличие унификатора узла, имеющего форму звёздочки, чёткое различим элемент «Rule», имеющий название.

Также у элемента Action в верхних двух правилах стоит ограничение на атрибут finished, который должен быть равен false, а у элемента Condition в нижних двух правилах атрибут flag должен быть не проинициализирован.

3.3 Реакция на применение правила

Для интерпретации поведения элементов и организации их взаимодействия между собой при применении правил было введено понятие «реакция на правило». Оно представляет из себя некую процедуру (на том же языке, что и поведение отдельного элемента, описанное выше), которая будет исполнена после преобразования модели согласно правилу, будет иметь доступ к логическим и семантическим атрибутам элементов правила, иметь возможность изменять их, а также, возможно, делать что-то более сложное, что позволяет выбранный язык.

По принципу работы реакция на применение правила похожа на исполнение действий при проходе состояний в диаграммах в xUML, а язык, при помощи которого эта реакция записывается, — аналог языка действий.

Пример реакции на правило присутствует на рисунке 18. Им является код, записанный в левом верхнем углу каждого правила. Как видно из рисунка, через него можно считывать и изменять атрибуты элементов.

4 Реализация

На основе предложенного подхода, описанного выше, в систему QReal в виде подключаемого модуля был встроен прототип визуального интерпретатора моделей. Этот плагин состоит из трёх основных компонент. Перед их подробным описанием рассмотрим типичный алгоритм работы разработчика визуального языка по добавлению семантики к нему и дальнейшей интерпретации с использованием данного прототипа.

4.1 Алгоритм работы проектировщика

Типичный алгоритм работы создателя визуального языка можно представить следующим образом. В первую очередь, разработчик в специальном окне указывает расположение метамодели языка, который он хочет интерпретировать. Данная метамодель уже должна быть расширена добавлением семантики к элементам, а также новыми узлами и связями, при необходимости. Это расширение разработчик должен произвести самостоятельно либо сразу при создании языка, либо непосредственным редактированием текстового файла, содержащего метамодель, после. Притом семантику элементов необходимо добавлять в логическую модель, потому что прототип не предоставляет возможностей для работы с семантической, описанной ранее.

После этого в системе автоматически генерируется и подгружается редактор семантики для этого языка. Вторым действием разработчика является создание описания семантики в этом редакторе.

Затем при помощи пользовательского интерфейса разработчик загружает эту семантику в интерпретатор, рисует диаграмму при помощи исходного визуального языка и начинает её интерпретацию, следя за информационными сообщениями в специальном окне.

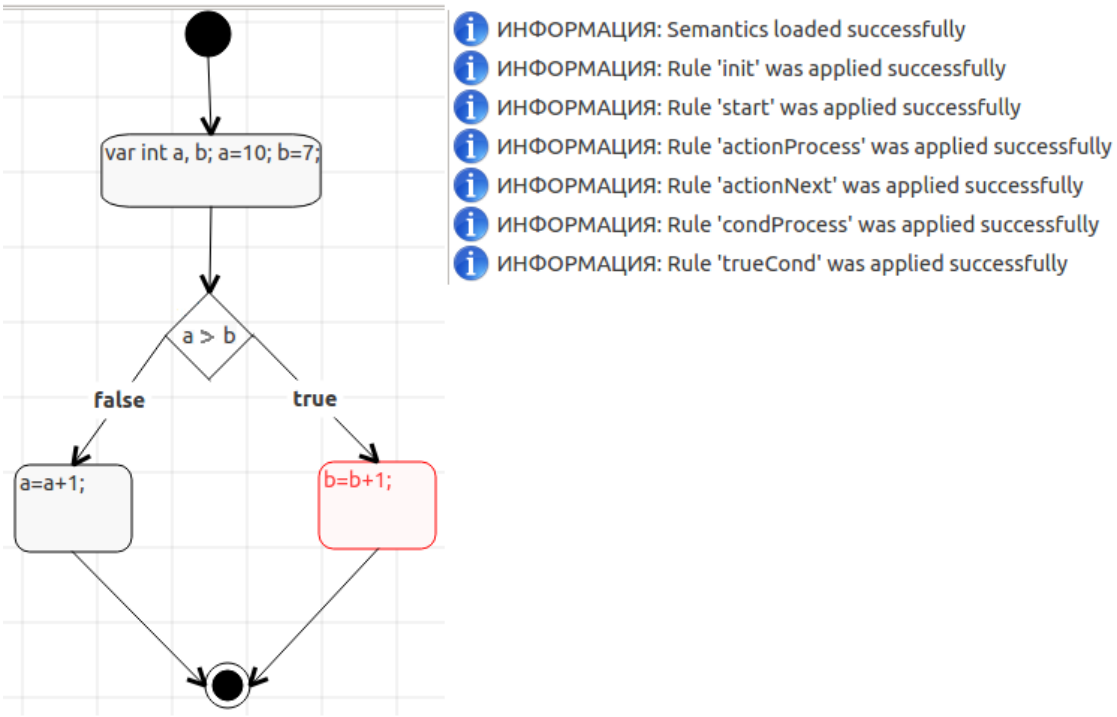

Рисунок 19: Пример интерпретации диаграммы блок-схем

Результаты применения различных правил при интерпретации выводятся на экран, как показано на рисунке 19. Также разработчик может наблюдать подсветку элемента с потоком исполнения.

4.2 Редактор семантики

4.2.1 Описание

Первая компонента прототипа отвечает за создание корректного редактора семантики. На вход она получает расположение метамодели визуального языка, при помощи стандартных средств Qt по разбору XML добавляет описанные ранее новые элементы и атрибуты, а полученный результат сохраняет на диск.

После этого генерируется файл с настройками сборки, также происходит корректировка путей до зависимостей. Далее идёт непосредственный процесс сборки при помощи компилятора gcc и динамическая загрузка нового редактора в работающую систему при помощи предоставляемого QReal API. Все эти действия происходят в автоматическом режиме.

Данный редактор позволяет создавать правила преобразования графов в соответствии с предложенным подходом. Его особенностью же является наличие специального элемента и связи, обозначающих текущее положение потока исполнения. При интерпретации вместо отрисовки этих элементов происходит просто подсветка места в диаграмме, в котором находится поток исполнения.

4.2.2 Пример: семантика языка блок-схем

Семантику языка блок-схем можно разделить на 3 части. Первая отвечает за инициализацию процесса исполнения на InitialNode, а также за корректное завершение интерпретации. Вторая отвечает за выполнение действий, записанных в элементе Action, третья — за правильную проверку условия в элементе Condition.

Общий принцип интерпретации блок-схем можно описать следующим образом. Вначале инициализируем исполнение, выставив флаг started у элемента InititalNode и поместив на него поток исполнения. После этого можно передать исполнение следующему элементу. Для завершения исполнения просто удаляем метку потока исполнения с элемента FinalNode.

Если попали на элемент Action, то исполняем действия внутри него и выставляем флаг finished, свидетельствующий о завершении исполнения этого элемента. Далее переходим к следующему элементу, предварительно опустив флаг finished у Action'а на случай возникновения циклов.

Если же попали на элемент Condition, то вначале считаем условие внутри него и в соответствии с этим выставляем флаг flag. Дальнейший переход осуществляем по той связи, флаг на которой совпадает с только что полученным. После перехода деинициализируем flag у Condition.

В качестве примера на рисунке 20 изображена полная семантика для языка блок-схем, записанная при помощи предложенного подхода.

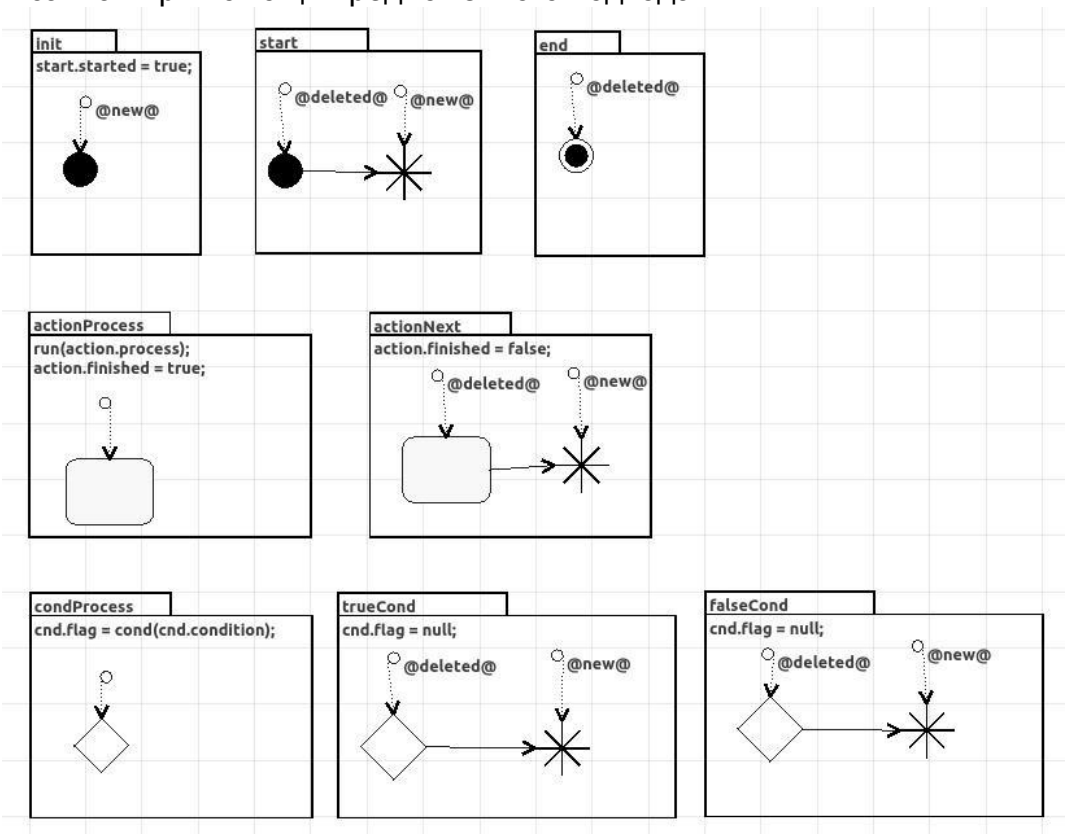

Рисунок 20: Полная семантика языка блок-схем

Рассмотрим подробнее данный рисунок. На нем изображено 8 семантических правил, у каждого из которых есть имя, которое будет выводиться на экран при

применении правила. Правила, соответствующие трём частям семантики, описанным ранее, расположены в одном ряду для каждой.

Сразу бросается в глаза активное использование добавленных семантических атрибутов started, finished и flag, которые влияют только на процесс исполнения и не должны отображаться при создании диаграммы, подлежащей интерпретации.

Маленький кружок и точечная связь означают текущее положение потока исполнения. Звездочка является унификатором узла, т.е. сравнение с любым элементом-узлом модели возвращает истину. Директивы @deleted@ и @new@ означают удаление и создание соответствующего элемента при применении правила. Код, написанный в левом верхнем углу любого правила, является реакцией на применение правила, через него можно исполнять действия внутри элементов исходной модели, проверять различные условия, а также считывать и изменять атрибуты элементов.

4.3 Модуль преобразования графов

Вторая компонента является модулем преобразования графов. Модель, нарисованная в соответствующем редакторе, является типизированным ориентированным мультиграфом с атрибутами и метками. В данном модуле должна быть реализована возможность поиска заданного подграфа в графе с последующим удалением, заменой и созданием нужных элементов.

4.3.1 GROOVE

GRaphs for Object-Oriented VErification, GROOVE[14], является программным средством, работающим с преобразованиями графов, которые являются базисом для преобразований моделей и операционных семантик. В нём реализована возможность применения таких преобразований, а также возможность автоматической верификации моделей относительно итоговой системы преобразований графов.

GROOVE является свободным проектом, распространяемым по лицензии Apache License v2.0 и реализованным на языке Java. В нём присутствует вся необходимая функциональность для предложенного способа, но его использование привнесёт в систему QReal дополнительную зависимость от сторонних программных средств, что крайне нежелательно.

Также при использовании GROOVE придётся создавать специальные довольно сложные модули, позволяющие переводить формат как интерпретируемой модели, так и самих правил преобразования графов из поддерживаемого в системе QReal в поддерживаемый GROOVE и наоборот. Подробнее о необходимых преобразованиях форматов моделей можно прочесть в работе [6]. Вдобавок может возникнуть проблема совместимости между кодом на Java в GROOVE и кодом на C++/Qt в QReal.

[14] GROOVE, http://groove.sourceforge.net/groove-index.html

4.3.2 Предложенное решение

Исходя из невозможности применимости существующих средств преобразования графов, в прототипе был реализован алгоритм поиска одного графа как подграфа в другом с учётом равенства типов, направлений связей, атрибутов и наличия потока исполнения на элементах. В конце работы этот алгоритм выдаёт набор соответствий элементов искомого графа и элементов исходной модели. Данный алгоритм является итеративно-рекурсивным, поэтому накопление результата будет происходить постепенно.

Рассмотрим набор структур данных, необходимых для работы этого алгоритма. Эти структуры будут пронумерованы цифрами, и при описании алгоритма будут использоваться именно они.

1. Соответствие, которое в конце будет содержать результат работы алгоритма. По элементу в искомом графе оно выдаёт соответствующий элемент в модели.
2. Список узлов в искомом графе, образующих подграф, который на данном шаге совпадает с некоторым подграфом в модели.
3. Список узлов в искомом графе, лежащих в (2), у которых есть связи с узлами искомого графа, не лежащими в (2).
4. Индекс узла в (3), рассматриваемого на текущем шаге. В начале работы проинициализирован нулём.

Таким образом, на каждом шаге имеется некий подграф в искомом графе, который уже был найден в модели. Соответствие узлов и связей этих подграфов уже содержится в (1). Также присутствует список узлов (3), который позволяет продолжать поиск искомого графа целиком в модели. Теперь рассмотрим принцип работы алгоритма целиком.

На первом шаге выбираем произвольный узел из искомого графа и ищем все узлы в модели, совпадающие с ним по критериям, которые были описаны выше. Запускаем цикл, проходящий по ним и при проходе каждого проделывающий следующие операции: очищение всех структур от старых данных и их заполнение информацией про соответствующие первые узлы.

Далее на втором шаге делаем следующее. Берём узел из списка (3) по индексу (4), находим для него первую связь (обозначим её за (а)) в искомом графе, не ведущую в список (2), т.е. второй узел данной связи не должен лежать в списке (2). Если такой не нашлось, то увеличиваем индекс (4) на 1 и запускаем второй шаг снова, если индекс (4) не равен длине списка (3). Если же равен, то, значит, искомый граф найден целиком и можно возвращать результат.

После этого сохраняем ссылки на найденную связь и на узел (обозначим его за (б)), с которым она соединена помимо начального. Находим по (1) и узлу (б) узел в модели (обозначим его за (в)), соответствующий (б). Ищем у него все связи, совпадающие с (а) и не ведущие в подграф в модели, соответствующий (2). Обозначим полученный список связей за (г). Если таких связей не нашлось, то совершаем откат и запускаем второй шаг снова.

Для каждой связи из (г) сохраняем ссылку на узел (обозначим его за (д)), с которым она соединена. Ищем все связи из (б), соединённые с узлом подграфа (3), и проверяем, есть ли такие же в модели из (д). Если какой-то не нашлось, то переходим к следующему ребру из (г). Если нашлись все нужные связи, то добавляем информацию в структуры данных о том, что (б) соответствует (д). В том числе, всю информацию про связи (б) с подграфом (3) и соответствующие связи в модели. Запускаем второй шаг снова, если индекс (4) не равен длине списка (3).

Откат происходит по следующему принципу. Убираем последний узел в списке (2), убираем последний узел в списке (3), уменьшив при необходимости индекс (4) на 1, убираем информацию об удаленном узле в соответствии (1) (как про сам узел, так и про его связи).

4.4 Реакция на применение правила

Третья компонента предоставляет возможность интерпретации реакции на применение правила. В специальном атрибуте элемента «Rule» в редакторе семантики можно писать на расширении уже созданного текстового языка [15]. Он будет выполнен тем же интерпретатором, что исполнял действия элементов Action и вычислял условия в элементах Condition.

Расширение языка состоит в том, что реализована возможность как чтения, так и записи новых значений в атрибуты элементов. Доступ к ним осуществляется по имени элемента и названию атрибута. Также расширение позволяет исполнять действия, записанные на том же языке внутри атрибута какого-либо элемента, при помощи функции run и, аналогично, вычислить значение условия при помощи функции cond. Прочие особенности синтаксиса остались неизменными.

5 Апробация

5.1 Взаимосвязь с рефакторингом

Основа предложенного подхода и прототипа непосредственно связана с работой Анастасии Кузенковой [14], посвящённой созданию средств рефакторинга моделей в системе QReal.

5.1.1 Преобразование графов

Рефакторинг модели представляет из себя поиск некоего шаблона в рассматриваемой модели с последующим правильным его преобразованием. Например, изменение направления связей между элементами, изменение имени элементов, вырезка и замена на один блок группы элементов модели в отдельную диаграмму.

В общем же случае преобразованием является изменение значений атрибутов элементов в найденном шаблоне, а также изменение направлений связей, замена, удаление и создание новых элементов. Данный набор действий является базой системы преобразований графов, рассмотренной ранее. Поэтому работа над реализацией модуля преобразования графов велась совместно.

5.1.2 Авторасскладывание элементов

Поскольку при применении правил преобразования графов происходит изменение исходной модели и её графического представления на экране, необходимо учесть проблемы, связанные с отображением добавленных элементов, т.к. при удалении и замене расположение остальных элементов можно не менять.

В работе Анастасии для решения данной задачи было предложено использовать программу dot пакета утилит по автоматической визуализации графов Graphviz[15], и с его помощью реализовано средство автоматической раскладки элементов модели на экране.

5.2 Доклад

По материалам данной работы был осуществлен доклад на Всероссийской научной конференции по проблемам информатики “СПИСОК-2012”.

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

Заключение

Результаты

В результате данной работы были изучены существующие подходы к визуальной интерпретации поведенческих диаграмм, и был проведен анализ таких подходов на предмет применимости по отношению к системе QReal. По итогам анализа было создано собственное решение, по возможности сохранившее положительные черты рассмотренных подходов.

Также был реализован прототип данного решения для системы QReal. В него входит создание корректного редактора семантики, удовлетворяющего предложенному подходу, модуль преобразования графов, отвечающий за поиск нужного подграфа в графе, а также расширение интерпретатора текстового языка блок-схем для организации исполнения реакции на применение правил.

С результатами данной разработки можно ознакомиться при помощи репозитория проекта[16]. Автор в нём принимал участие под учетной записью vapolyakov, основной результат соответствует коммитам с хэшем 37e3a05, 0d18873 и 4c3b168.

Дальнейшее развитие

Анализ созданного прототипа поможет оценить степень применимости данного подхода и расставит приоритеты в дальнейшем развитии реализации визуального интерпретатора и корректировки самого подхода.

Данное исследование можно продолжить по следующим направлениям: более детальная формализация созданного подхода с учётом его последующей реализации и расширение функциональности существующего прототипа.

Также возможно дальнейшее обобщение предложенного метода. Например, задание функции-веса для правил, которая будет влиять на порядок их применения, даст возможность визуализировать различные алгоритмы на графах. Выбор правила или места применения правила во время очередного шага, когда таких возможных правил или мест несколько, сделает процесс интерпретации ещё более интерактивным и наглядным.

После этого необходимо сосредоточиться на детальном изучении работы [6] и создании приемлемого пользовательского интерфейса, а также на превращении интерпретатора в полноценный отладчик с точками останова, пошаговой интерпретацией и возможностью просмотра текущих значений переменных.

[16] QReal, https://github.com/qreal

Список литературы

1. Object Action Language Reference Manual, URL: www.ooatool.com/docs/OAL08.pdf
2. Shlaer-Mellor Action Language, URL: www.modelint.com/downloads/small.pdf
3. Structured Query Language, URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498
4. http://se.cs.depaul.edu/ise/zoom/projects/statechart/SE690DetailedPresentation.ppt
5. http://www.devx.com/enterprise/Article/10717
6. Nils Bandener, Visual interpreter and debugger for dynamic models based on the Eclipse platform, Diploma Thesis, 2009, URL: http://is.uni-paderborn.de/uploads/tx_dsorexams/Diploma_Thesis_Nils_Bandener.pdf
7. A. Corradini, R. Heckel, U. Montanari, Graphical operational semantics // In Proc. ICALP2000 Workshop on Graph Transformation and Visual Modelling Techniques, Carleton Scientific, 2000.
8. A. Evans, S. Kent, Core Meta-Modelling Semantics of UML: The pUML Approach // In R. B. France and B. Rumpe (eds.), UML'99: The Unified Modeling Language - Beyond the Standard, Second International Conference, Fort Collins, CO, USA, October 28-30, 1999, Proceedings, volume 1723 of Lecture Notes in Computer Science. Springer, 1999.
9. Jan Hendrik Hausmann, Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages, PhD thesis, 2005, URL: http://is.uni-paderborn.de/uploads/tx_sibibtex/Dynamic_Meta_Modeling_-_A_Semantics_Description_Technique_for_Visual_Modeling_Languages.pdf
10. S. Kuske, A Formal Semantics of UML State Machines Based on Structured Graph Transformation // In M. Gogolla and C. Kobryn (eds.), UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 2001, Proceedings, volume 2185 of Lecture Notes in Computer Science (LNCS), pages 241–256. Springer, 2001.
11. Mellor Stephen J., Balcer Marc J, Executable UML: A foundation for model-driven architecture // Addison Wesley, 2002.
12. Christian Soltenborn, Gregor Engels, Towards test-driven semantics specification // In B. Selic A. Schürr, editor, Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009), Denver, Colorado (USA) (to be published), Berlin/Heidelberg, 2009. Springer.
13. F. Steimann, T. Kuhne, A Radical Reduction of UML's Core Semantics // In J.-M. Jezequel, H. Hussmann, S. Cook (eds.), UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools, 5th International Conference, Dresden, Germany, September/October 2002, Proceedings, volume 2460 of LNCS, pages 34–48. Springer, 2002.
14. А.С. Кузенкова, Поддержка механизма рефакторингов в metaCASE-системе QReal // Материалы третьей межвузовской. научной конференции по проблемам информатики. СПб.: ВВМ, 2012.

15. В.А. Поляков, Разработка визуального интерпретатора моделей в системе QReal, курсовая работа, 2011, URL: http://se.math.spbu.ru/SE/YearlyProjects/2011/YearlyProjects/2011/345/345_Polyakov_report.pdf
16. Т.А. Брыксин, Ю.В. Литвинов, Технология визуального предметно-ориентированного проектирования и разработки ПО QReal // Материалы второй научно-технической конференции молодых специалистов «Старт в будущее», посвященной 50-летию полета Ю.А. Гагарина в космос. СПб. 2011. С. 222-225.