Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Разработка визуального интерпретатора моделей в системе QReal

Курсовая работа студента 345 группы
Полякова Владимира Александровича

Научный руководитель ст. пр. Брыксин Т.А.

Санкт-Петербург
2011

Оглавление

Введение

В последнее время становится все более популярным предметно-ориентированный подход к разработке ПО, в частности и в области проектирования, основанной на моделях. Модельно-ориентированная разработка основывается на описании программы при помощи некоторого количества моделей, характеризующих её с различных сторон. Это довольно общий подход, а, значит, детали его реализации зависят от контекста поставленной задачи.

Предметно-ориентированный же подход состоит в том, что создать новый специализированный язык, основанный на конкретной поставленной задаче, и решить её с его помощью можно значительно быстрее, чем решать её при помощи языков общего назначения. Автоматическая генерация полноценного кода по моделям является второй важной составляющей данного подхода.

1 Постановка задачи

Обычно в описанных подходах используются визуальные языки моделирования (программирования). Визуальные языки делятся на статические (которые описывают структуру создаваемого приложения) и поведенческие (описывающие алгоритмы, взаимодействие частей системы и прочую динамику поведения).

Возможность интерпретации поведенческих диаграмм значительно повысила бы скорость разработки, позволив проектировщику искать ошибки и отлаживать модели еще до завершения их создания. Данный подход уже так или иначе используется в некоторых CASE-системах (например, в Borland Together [2]), однако для нас интерес представляет применение этого подхода в metaCASE-средстве, т.е. возможность быстрого создания интерпретаторов и отладчиков для разрабатываемых пользовательских визуальных языков.

Также интересно подключение к этому процессу уже существующих отладчиков для текстовых языков программирования таких, как gdb или pdb (т.е. визуальная отладка модели при помощи отладчика целевого языка генерации), поскольку возможность работы с привычными средствами значительно упростит весь процесс отладки.

2 Обзор существующих решений

На данный момент есть несколько основных программных продуктов, имеющих функциональности на подобную тему:

2.1 IBM Rational Software Architect

IBM® Rational® Software Architect — это мощное средство визуального моделирования и разработки программных систем. Оно основывается на UML и позволяет проектировать архитектуру для C++, J2EE приложений и веб-сервисов, имеет хорошую документацию, триальную версию на 30 дней, однако предоставляется бесплатно для учебных целей в рамках программы IBM Academic Initiative [18].

В рамках визуальной интерпретации и визуальной отладки Architect обладает следующими возможностями:

- Позволяет интерпретировать и отлаживать диаграммы последовательностей, диаграммы активностей и взаимодействия, диаграммы состояний (конечных автоматов), opaque behavior diagrams (“диаграммы непрозрачных действий”, т.е. поведения с зависящей от реализации семантикой, например, с кодом на Java или на C++), основанные на UML моделях, а также модели бизнес-процессов
- Для указанных типов диаграмм имеется как пошаговая, так и автоматическая интерпретация с выделением маркером текущего элемента и элемента, которые будет исполнен следующим, а также с возможностью подсветки истории, т.е. тех элементов, которые уже были интерпретированы. Также можно произвести интерпретацию вплоть до указанного заранее места (“run to here”)
- Изменять диаграмму можно прямо во время отладки. Если из-за этого нарушилась логика диаграммы и дальнейшая интерпретация невозможна, то об этом будет сообщено пользователю
- Интерпретацию/отладку можно приостановить, возобновить, прекратить, начать заново
- Есть полная поддержка точек останова. Расставлять их можно при помощи контекстного меню для любого элемента, также их можно удалять, пропускать, сохранять и загружать в новые проекты
- Для диаграмм поведений, основанных на событийной системе, введён специальный синтаксис поддержки событий
- В качестве свойств элементов, а также для событий и прочего добавлена поддержка переменных, за которыми можно наблюдать и которые можно изменять в специальной части пользовательского интерфейса
- Для топологических схем имеется два типа анимации интерпретации: в виде стрелок, показывающих последовательность действий, а также в виде сообщений в цветных прямоугольниках, плавно переходящих по связям между элементами, т.е. присутствует возможность трассировки топологии
- Для opaque behavior diagrams имеется поддержка UAL (UML Action Language). На данный момент она не завершена, но будет доведена до подмножества языка Java. Некорректный код помечается маркером. За переменными, использующимися в коде, можно следить в интерфейсе отладчика
- Поддержка различного рода ветвлений, циклов и прочего осуществляется как на автоматическом уровне, так и на ручном, т.е. для диаграмм с несколькими возможностями потока исполнения на каждом ветвлении пользователь сам выбирает по какой ветке процесс будет исполняться дальше

В качестве недостатков данного средства можно перечислить следующее:

- Высокая стоимость: от 466 до 932 USD
- Узкая направленность средства т.е. поддержка различного рода диаграмм, основанных только на UML
- Отсутствие отладки при помощи отладчиков текстовых языков программирования (хотя кодогенерация в Java, C++, C# присутствует)
- Слишком обширный пакет, если нужно выполнить не очень большую задачу

Примеры работы с IBM Rational Software Architect[1]:

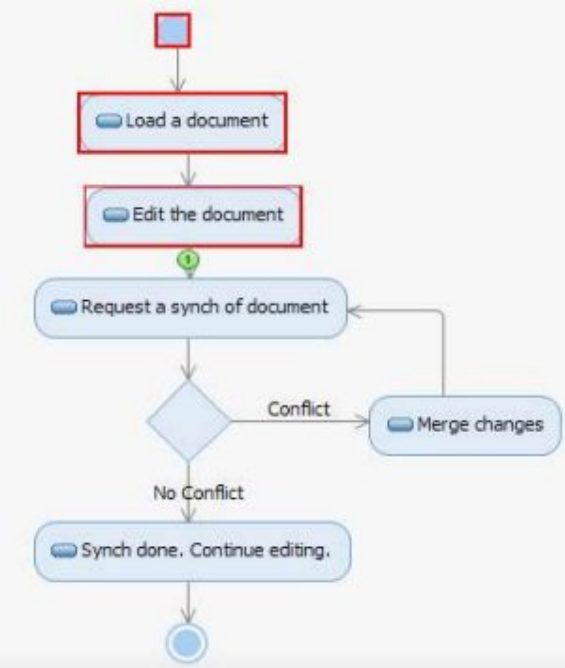

Рис. 1: Подсветка истории интерпретации и обозначение текущего элемента для диаграммы активностей

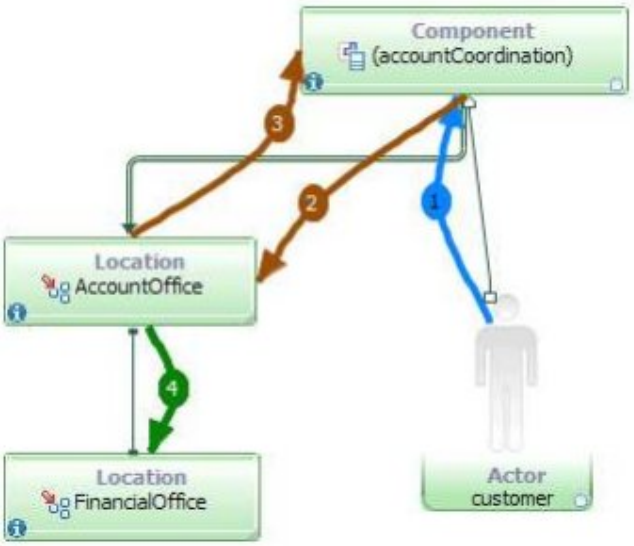

Рис. 2: Пример истории интерпретации для топологической схемы

[1]Скриншоты взяты из статей [21] [23]

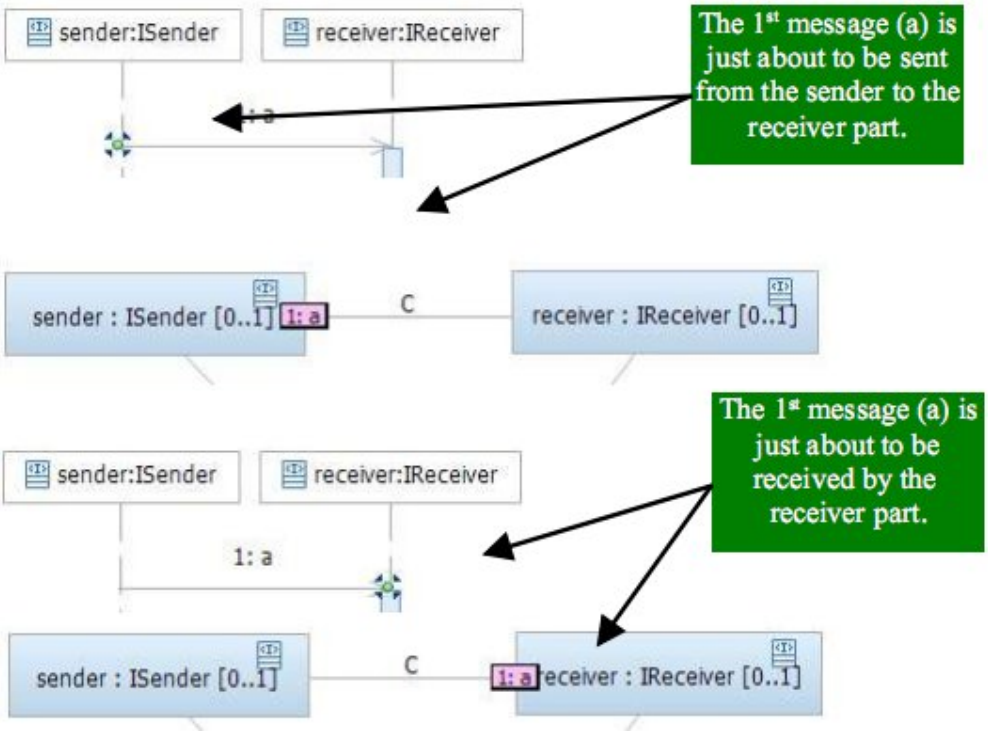

Рис. 3: Пример анимации интерпретации на диаграмме последовательностей и топологической схеме

Подробнее с этим продуктом можно ознакомиться на официальном сайте [9], а также из статей [21] [22] [23] [24].

2.2 AMUSE for SparxSystems Enterprise Architect

Advanced Modeling in UML with Simulation and Execution (AMUSE) for Enterprise Architect — это дополнение к известному кроссплатформенному средству UML-моделирования и проектирования, которое содержит в себе многочисленные возможности: от многопользовательской разработки и хорошего интерфейса до генерации документации и кода на различных языках (таких как Java, C++, C#, Python, PHP и др.). AMUSE позволяет интерпретировать и отлаживать UML-диаграммы состояний. Подробнее можно ознакомиться на официальном сайте EA [12] и AMUSE [1].

В рамках этого предоставляется следующее:

- Подсветка текущего состояния, возможных переходов, точек останова
- На элементы можно накладывать произвольные ограничения притом как на состояние до входа в элемент, так и после выхода. Места, где ограничения не выполнились, подсвечиваются, а впоследствии пользователю сообщается об ошибке
- Из одного состояния может быть много возможностей перехода (которые назвыаются триггерами). Для запуска одного из триггеров достачно дважды на нём кликнуть
- Можно сделать диаграмму последовательностей для запуска триггеров для конкретной модели. Тогда триггеры будут запускаться в определённом порядке автоматически
- Имеется поддержка процедур, т.е. запуск других диаграмм состояний (а также и диаграмм активностей) из произвольного элемента. Их тоже можно отлаживать при интерпретации основной модели
- Имеется поддержка диаграмм активностей и их преобразования в диаграммы последовательностей
- Возможен параллельный запуск нескольких диаграмм
- Имеется поддержка переменных в качестве полей у элементов. Их можно просматривать после каждого шага интерпретации в специальной части интерфейса. При приостановке интерпретации их можно изменять
- Написанный внутри элементов код компилируется, поэтому интерпретация может протекать очень быстро. Для этого в настройках можно регулировать задержку в миллисекундах при переходах между состояниями

В качестве недостатков выделяются:

- Необходимость Enterprise Architect для работы и, как следствие, от $199 до $699 за лицензию
- Узкая направленность
- Нет отладки при помощи текстовых отладчиков

Примеры работы с AMUSE[2]:

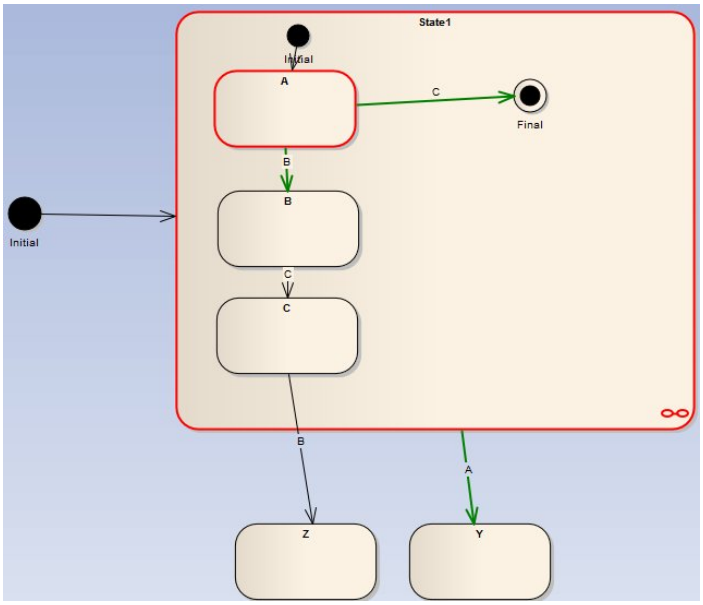

Рис. 4: Пример подсветки текущего состояния (красный) и возможных триггеров (зелёный)

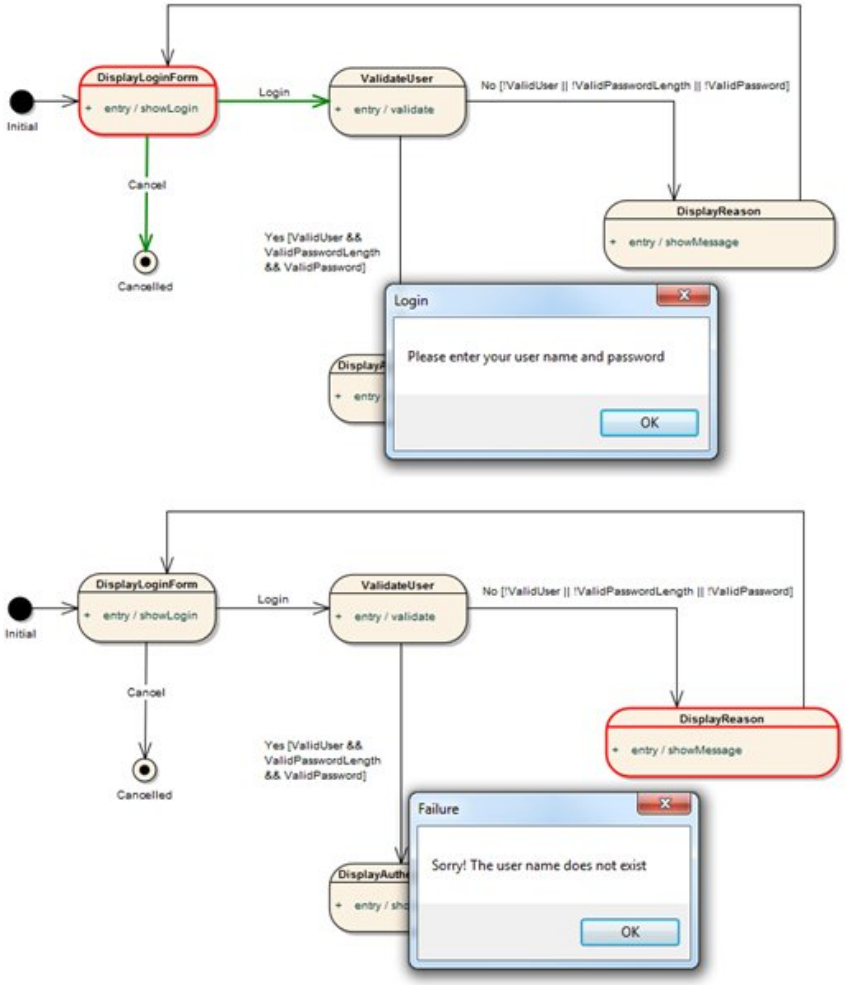

Рис. 5: Пример перехода по триггеру и обработки ошибок

[2]Скриншоты взяты из обзора AMUSE [14] и онлайн-документации [19]

2.3 Cameo Simulation Toolkit

Cameo Simulation Toolkit — специальное расширение к MagicDraw, являющейся известной средой визуального проектирования и моделирования на UML, SysML, BPMN и UPDM, имеющей поддержку DSL, которая обеспечивает ещё большее удобство использования при решении конкретных задач. Также MagicDraw позволяет легко рисовать графические интерфейсы и их соответствия в виде диаграмм, которые впоследствии можно отлаживать при помощи Cameo Simulation Toolkit. Подробнее ознакомиться с продуктами можно на официальных сайтах MagicDraw [10] и Cameo Simulation Toolkit [3].

В рамках визуальной отладки Simulation Toolkit имеет следующие особенности:

- Предоставляет возможность интерпретации, отладки и визуализации этих двух процессов для диаграмм состояний и диаграмм активностей, основанных на UML 2.0. Поддерживаются стандарты fUML, W3C SCXML, для математических моделей имеется SysML, а в качестве языка программирования для выражения действий поддерживаются JavaScript, Ruby, Python и др
- Пошаговая отладка с подсветкой, подсветка истории интерпретации, поддержка точек останова
- Слежение за переменными в ходе отладки, а также возможность их изменения
- Возможность установки ограничений (как пост-, так и пред-), описанных на различных языках для выражения действий
- Возможность легкого создания графических интерфейсов и их последующей отладки при помощи запуска нужных поведений

В качестве недостатков можно выделить следующие:

- Значительная стоимость лицензии (как на сам продукт, так и на MagicDraw)
- Направленность на UML, хоть и имеется поддержка DSL
- Нет отладки при помощи текстового отладчика

Пример работы с Cameo Simulation Toolkit[3]:

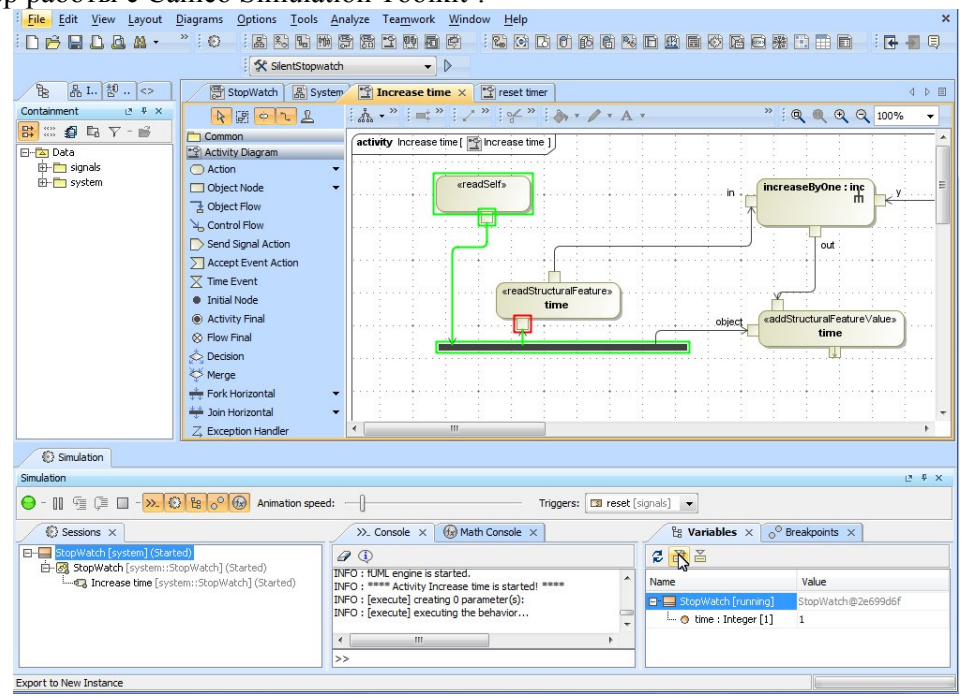

Рис. 6: Пример отладки диаграммы активностей

[3]Скриншот взят из обучающего видео [20]

2.4 UniMod

UniMod — это плагин для среды разработки Eclipse, реализующий идеи SWITCH-технологии и xUML и созданный группой разработчиков из СПбГУ ИТМО. UniMod создан для разработки, проверки и отладки программ, представленных в виде набора конечных автоматов, построенных при помощи UML. Таким образом он предназначен для поддержки автоматного программирования в Eclipse. Данный продукт основан на диссертационной статье [28], распространяется бесплатно под лицензией Free LGPL, имеет документацию на русском. Подробнее с ним можно ознакомиться на официальном сайте [13].

В рамках рассматриваемой задачи имеет следующие особенности:

- Простой интерфейс, подсветка при интерпретации, поддержка точек останова, слежение за переменными
- При интерпретации также может использоваться генерация и компиляция кода для ускорения ускорения процесса
- Поддерживается код на Java, C++, C#, JavaScript, Perl

В качестве минусов можно выделить лишь узкую направленность на описание программ при помощи конечных автоматов, отсутствие отладки при помощи текстовых отладчиков.

Пример работы с UniMod[4]:

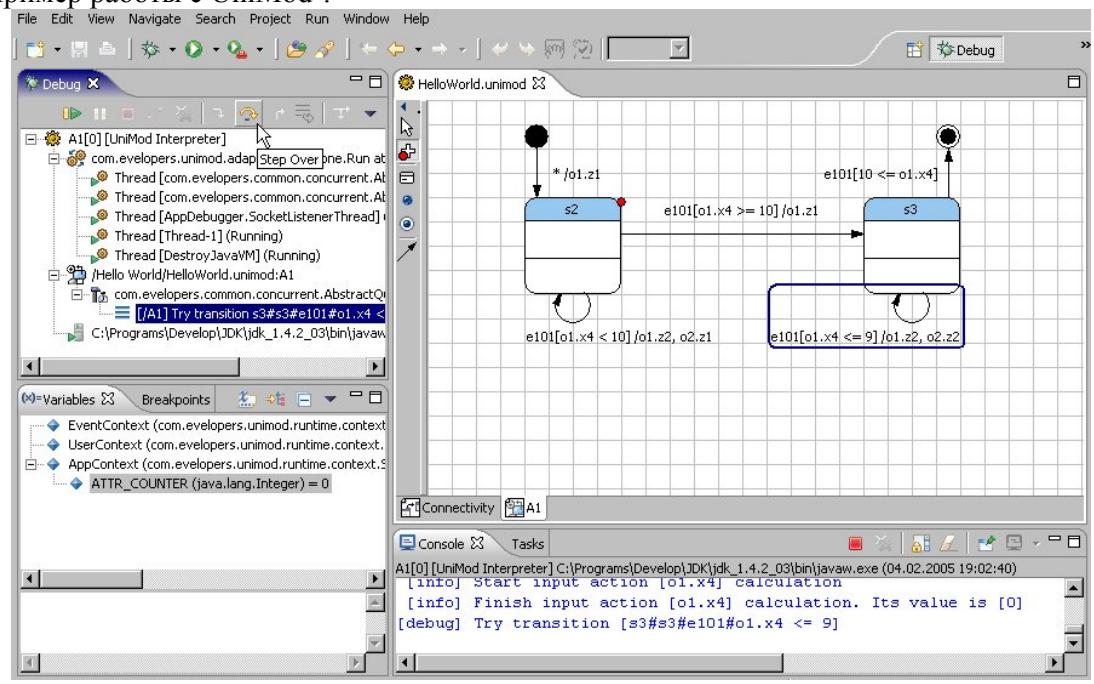

Рис. 7: Пример отладки диаграммы состояний (текущее положение отмечено синим прямоугольником)

[4]Скриншот взят из обучающего видео [17]

2.5 *"Visual interpreter and debugger for dynamic models based on the Eclipse platform"*

2.5.1 Обзор

Дипломная работа Nils'a Bandener'a [25] посвящена созданию визуального интерпретатора и отладчика для моделей, нарисованных на основе визуальных языков поведенческого типа. В ней в первую очередь рассматривается именно сам процесс создания и приводятся изображения того, как это могло бы выглядеть. Сама реализация программного средства, подходящего для произвольных диаграмм поведенческого типа отсутствует.

В работа подробно описывается DMM (Dynamic Meta Modeling) подход в задаче составления семантики исполняемого визуального языка. Он разделяется на две составляющие: метамодель времени исполнения (runtime meta model), обычно основывающаяся на синтаксической метамодели и определяющая соответственно исполняемую часть языка, а также правила преобразования графов (graph transformation rules), необходимых для описания переходов между состояниями, т.е. между двумя сущностями runtime meta model.

Также рассмотрено возможное применение уже существует программных средств для облегчения реализации DMM-подхода. Например, EMF (Eclipse Modeling Framework [4]) для описания общей метамодели визуального языка, GROOVE (Graphs for Object-Oriented Verification [8]) для осуществления преобразований над графами, GMF (Graphical Modeling Framework [7]) для графического описания создаваемых визуальных языков.

Подробно разобраны основные функциональности, характерные для интерпретации и отладки, в проекции на визуальные языки. Например, рассмотрена работа с моделями, имеющими несколько путей исполнения, понятие "один шаг" при интерпретации визуальных языков, подробно описаны понятия точек останова и контрольных точек данных. Также сформулированы и разобраны другие характерные для средств отладки требования.

Для процесса отладки рассмотрено возможное использование EProvide (Eclipse Plugin for Prototyping Visual Interpreters and Debuggers [6]), основанный на Eclipse Platform Debug [5], и объяснено, почему он не подходит для поставленной задачи.

Приведены примеры возможной реализации графического интерфейса для создаваемого программного средства, чётко сформулированы случаи применения данного средства в виде use-case диаграммы и последующего подробного объяснения её составляющих. Также сформулированы административные use-case'ы, непосредственно вытекающие из подробностей реализации данного средства.

В конце работы в деталях рассказано об архитектурной составляющей создаваемого программного средства, включая использование EMF, GMF и GROOVE.

2.5.2 Возможное использование

В контексте поставленной ранее задачи данная работа представляет интерес в первую очередь наличием подробного разбора DMM-подхода для описания семантики поведения объектно-ориентированных языков, чётких формулировок необходимых требований, функциональностей и возможных проблем при создании визуального интерпретатора и отладчика, а также способов их разрешения.

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

3 Реализация

Данная разработка ведётся применительно к metaCASE-средству QReal. Исследование производится путём создания примеров с дальнейшим их обобщением и повышением уровня абстракции.

3.1 QReal

QReal [11] — кроссплатформенное средство визуальной разработки ПО. Оно поддерживает многопользовательскую разработку, удалённый доступ к репозиторию системы, быстрое создание новых визуальных языков, а также генераторов кода для них и другие характерные для таких средств функциональности.

Архитектура QReal базируется на шаблоне проектирования Model/View и может быть представлена в виде:

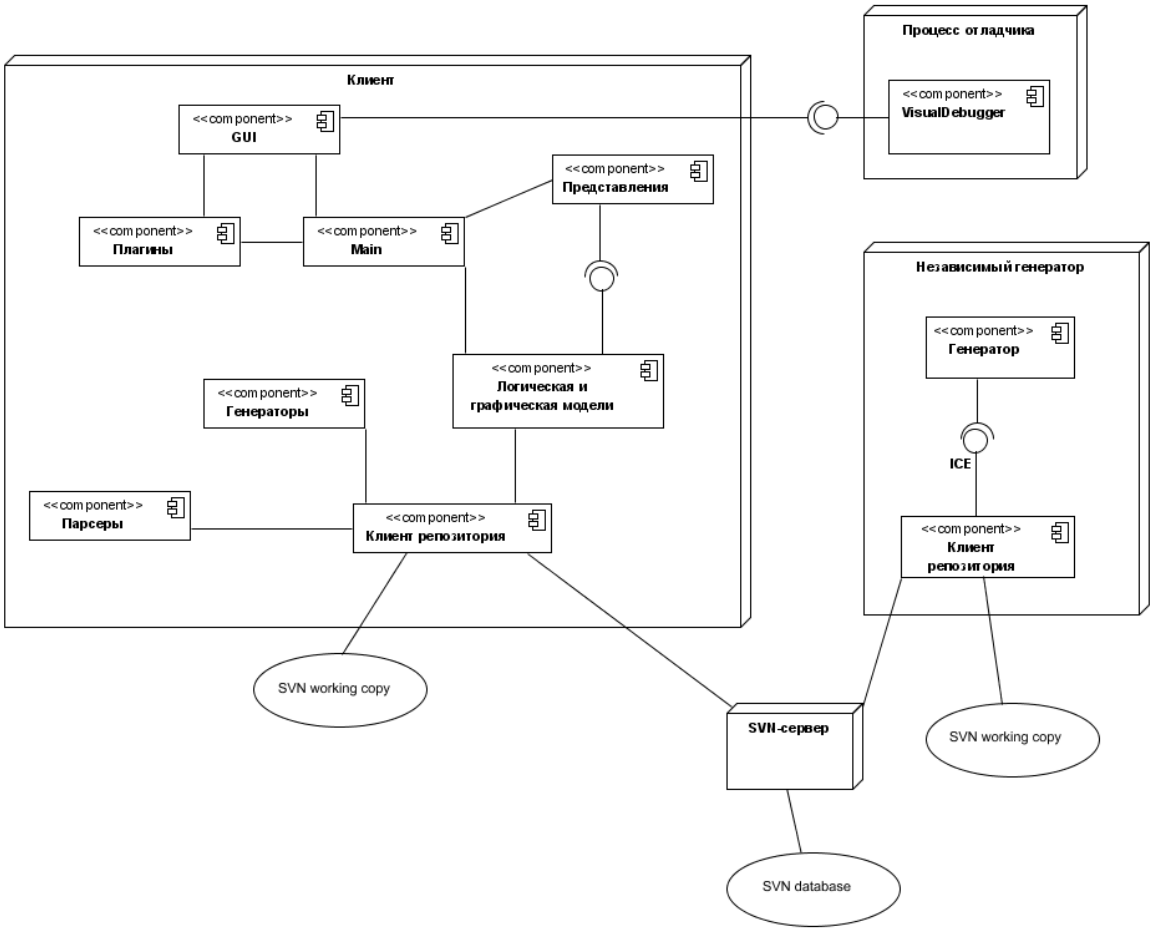

Рис. 8: Архитектура QReal

Модель работает с логическими и графическими описаниями диаграмм, предоставляет доступ к репозиторию, а представлениями являются элементы пользовательского интерфейса (редактор свойств, миникарта, отображение состояния графической и логической модели).

QReal позволяет быстро создавать новые визуальные языки, поэтому каждый визуальный редактор является отдельным подключаемым модулем, а архитектура QReal как CASE-системы включает в себя абстрактное ядро, поставляющее базовый функционал для всех редакторов (например, отрисовка элемента), и модули, реализующие специфики языков и одинаково разбираемые абстрактным ядром.

Поэтому основным для работы является создание удобного способа пользовательского задания логики интерпретации, которая будет добавлена в данные модули и правильным образом разобрана ядром.

Также в QReal присутствует разделение логической и графической модели при представлении и хранении данных. Так, например, некоторые элементы модели могут несколько визуальных представлений (т.е. графических) или вообще не иметь либо визуального представления, либо логического. Но всегда верно, что у одного графического представления элемента есть не более одного логического, а у логического элемента 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.

Подробнее об архитектуре QReal написано в [29].

3.2 Визуальный интерпретатор

С архитектурной точки зрения, визуальный интерпретатор встраивается в QReal в виде отдельного модуля, использующего общедоступное API интерпретаторов, генераторов, репозиториев, графической и логической модели и реализующего основную функциональность. Графическая модель используется для подсветки и выделения элементов, логическая — для выяснения названия элемента, для обращения к его полям и нахождения элементов, соединённых с ним связью.

В качестве примера был выбран наиболее интуитивно понятный язык блок-схем. Для него был создан визуальный язык, а также текстовый язык, на котором можно писать выражения внутри элементов модели (для этого у элементов есть специальные текстовые поля, а для поддержки ветвлений у переходов есть поле, отвечающие за его суть в математической логике).

В визуальном языке есть элементы, отвечающие за действие, ветвление, переход от одного элемента к другому, а также явно выраженные элементы начала и конца программы. Корректная программа должна содержать ровно один начальный элемент и заканчиваться конечным (которых может быть много для удобства изображения диаграммы). Иначе интерпретатор выдаст ошибку в первом случае и предупреждение во втором.

В текстовом языке есть целые числа и числа с плавающей точкой, переменные, полная поддержка арифметических и логических выражений. В роли грамматик для выражений использовались следующие, записанные в РБНФ:

Term ::= + Term | - Term | (Expression) | Number | Identifier

Mult ::= Term [(| /) Term]*

Expression ::= Mult [(+ | -) Mult]

SingleComparison ::= Expression (> | >= | < | <= | ==) Expression

Disjunction ::= SingleComparison | !(Condition | (Condition)

Conjunction ::= Disjunction [&& Disjunction]

Condition ::= Conjunction [|| Conjunction]

Трудность определения в логических выражениях, чему принадлежит открывающая скобка ‘(’, арифметическому или же логическому выражению, была решена при помощи сравнения позиций ближайшей закрывающей скобки ‘)’ и ближайшего знака сравнения.

В виде отдельного модуля для визуального интерпретатора был разработан парсер и интерпретатор данного текстового языка. Также в этом модуле осуществляется полная проверка синтаксической и семантической корректности кода, написанного внутри элементов блок-схемы. Результаты проверки могут быть легко извлечены извне. Также модуль может выдавать предупреждения, например, о потере точности в вычислениях или о завершении исполнения программы не в конечном узле.

Для созданного визуального языка интерпретатор позволяет производить пошаговую отладку как в ручном режиме (каждый переход требует нажатия определённой кнопки), так и в автоматическом (переходы осуществляются автоматически через указанный в настройках промежуток времени вплоть до завершения созданной программы или до места возникновения ошибки). В процессе интерпретации автоматически проверяется синтаксическая и семантическая корректность, а также имеется возможность изменения диаграммы, что увеличивает удобство отладки созданной диаграммы. В ручном режиме корректность программы проверяется на каждом шаге, а в автоматическом — результат и предупреждения выводятся после завершения работы, либо в момент критической ошибки.

Об ошибках и предупреждениях интерпретатор сообщает системе, которая их показывает пользователю с явным выделением элемента схемы, а также с указанием позиции в коде внутри элемента, на котором они случились.

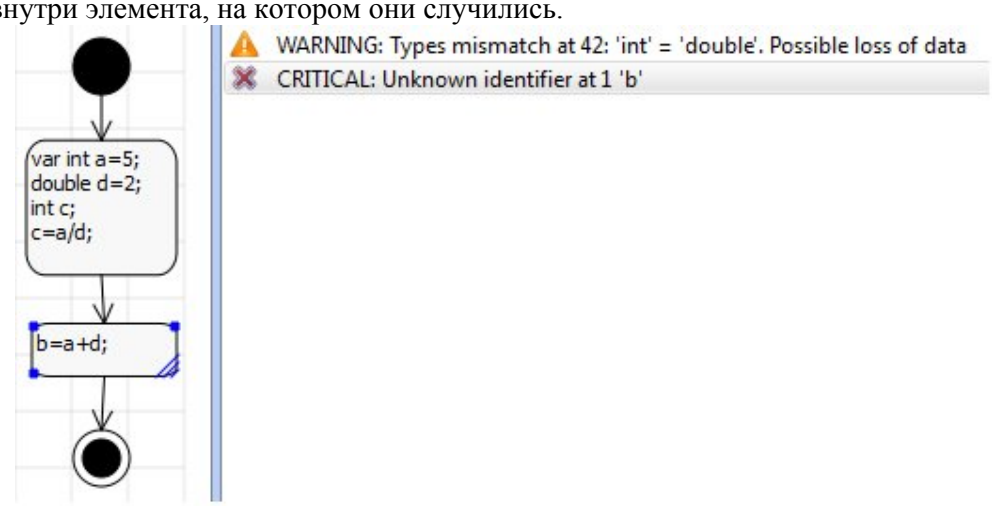

Рис. 9: Предупреждение и сообщение об ошибке

Вся подсветка и вывод ошибок осуществляется через специальный интерфейс, предоставляющий интерпретаторам удобно сообщать об ошибках, выводить необходимую информацию, подсвечивать и выделять указанные элементы модели, убирать такое выделение и прочее. Программно подсветка происходит при помощи QGraphicsColorizeEffect — специального средства для накладывания графических эффектов на QGraphicsItem.

3.3 Визуальный отладчик

Работа с отладчиком целевого языка генерации с архитектурной точки зрения происходит аналогично (Рис. 8). Соответствующий модуль даёт возможность системе собирать и запускать программу в нужном отладчике, устанавливать точки останова, выполнять пошаговую отладку и использовать другие характерные для средств отладки команды. После выполнения команды модуль сообщает системе о результатах работы, а также дополнительную информацию, например, о текущей точке останова, значениях переменных и др.

Доступ к большинству функций можно осуществить через меню, наиболее важные же вынесены в панель инструментов. Например, действия, такие как сгенерировать код, собрать бинарный файл, запустить текстовый отладчик, установить точку останова на начало программы, а также начать отладку объединены в одно, вынесенное на панель инструментов. Также там присутствует полная остановка отладки и переход к следующей строке.

Точки останова могут устанавливаться на произвольных элементах диаграммы, что в сгенерированном коде соответствует первой строке кода в элементе и именно на неё устанавливается отладчик текстового языка. Для того, чтобы распознавать как точки останова, так и просто вывод текстового отладчика при выполнении очередной строчки кода, создаётся соответствие между каждой строчкой сгенерированного кода и элементом диаграммы, из которого она получилась.

Имеются следующие возможности автоматической расстановки таких точек: установить их во всех элементах диаграммы, чтобы отлаживать программу поэлементно, а не построчно, что значительно упрощает использование данного способа отладки, или же установить точку в начало программы и отлаживать её построчно.

Подсветка элементов, генерация кода, вычисление корректного соответствия точек останова в блок-схеме и в коде и дальнейшее использование этого соответствия во время отладки предоставляются модулем визуального интерпретатора.

Информация, выдаваемая текстовым отладчиком, выводится пользователю. Из неё извлекается номер отлаживаемой строчки кода и при помощи соответствия подсвечивается нужный элемент. После завершения работы текстового отладчика с программой его вывод остаётся, таким образом при выделении его частей будут выделяться соответствующие элементы диаграммы.

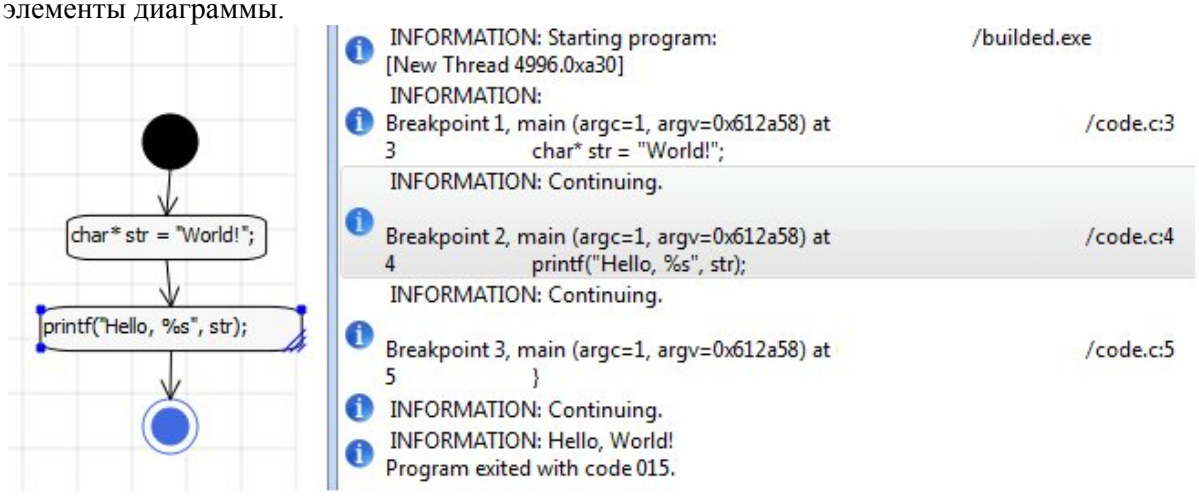

Рис. 10: Отладка с использованием gdb, подсветка и выделение элементов

Сам отладчик запускается в отдельном потоке в виде нового процесса, взаимодействие с которым осуществляется через класс QProcess. Передача информации между основным потоком системы и этим процессом выполнена при помощи системы слотов и сигналов, созданной разработчиками Qt, и проходит через модуль визуального отладчика.

4 Апробация

4.1 QReal:Robots

Данная работа использовалась при создании редактора алгоритмов работы роботов Lego Mindstorms NXT [15] [26]. Был доработан парсер для арифметико-логических выражений: добавлены логарифмические и тригонометрические функции. Для этого редактора была реализована пошаговая интерпретация, притом команды роботу посылаются по bluetooth прямо во время отладки и можно вживую наблюдать за его действиями. Сам визуальный язык довольно понятный и простой в обращении, позволяет быстро научиться как использованию существующих элементов редактора, так и созданию новых. На рисунке в качестве примера изображена программа, реализующая следующее действие робота: движение вдоль стены на заданном расстоянии.

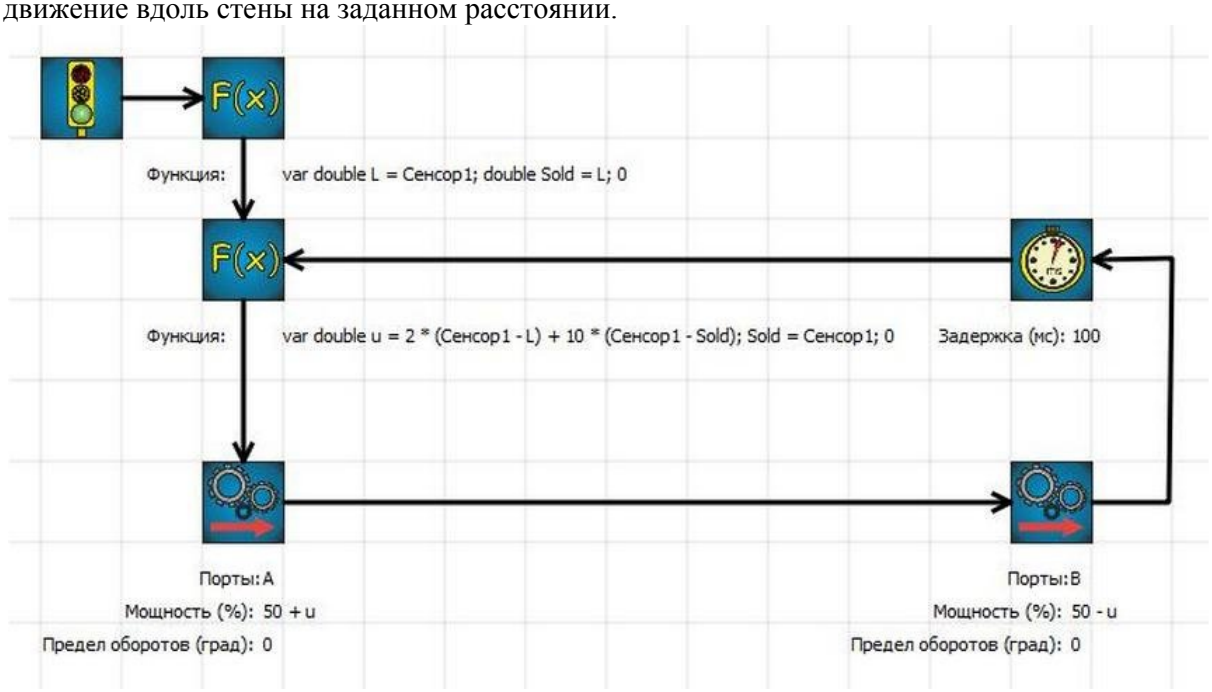

Рис. 11: Пример диаграммы в редакторе алгоритмов поведения роботов

4.2 Отладка при помощи gdb

Для редактора блок-схем создан C-подобный язык, для которого реализована генерация кода и отладка в gdb с подсветкой элементов блок-схемы, а также с выделением элементов после завершения работы отладчика при выделении соответствующей строки с кодом или точкой останова. Впоследствии возможно добавление языка python и отладки в pdb, а также других языков с доступными текстовыми отладчиками, расширение способа генерации кода, повышение удобства использования данного вида отладки, посредством улучшения интерфейса.

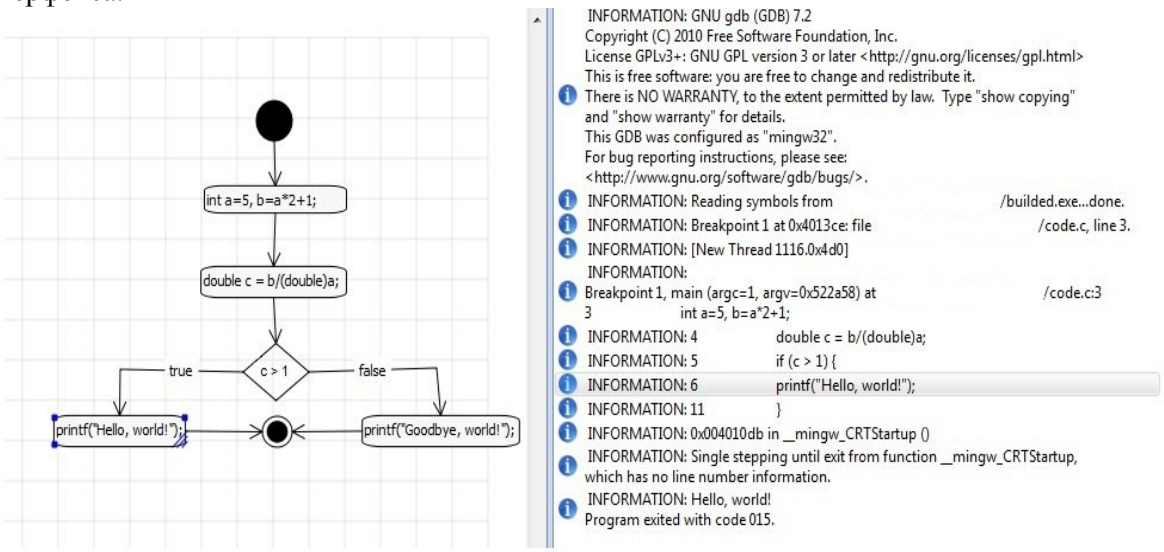

Рис. 12: Отладка при помощи gdb

4.3 Доклад

По материалам данной работы был осуществлен доклад на межвузовском конкурсе-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования" [27].

Заключение

Результаты

На текущий момент создан редактор блок-схем, для которого реализована пошаговая интерпретация с автоматической проверкой синтаксической и семантической корректности и с подсветкой текущего элемента. В результате данной работы был также разработан интерфейс для визуального интерпретатора, позволяющий работать с произвольными диаграммами. Ведется работа и над визуальной отладкой сгенерированного кода при помощи подключения к отладчику целевого языка генерации, т.е. по модели генерируется код (например, на c++, python), запускается отладчик языка генерации (например, gdb, pdb) , а далее работа с отладчиком, совмещенная с подсветкой места в диаграмме, в котором находится поток исполнения отладчика. Для редактора блок-схем создан С-подобный язык, для которого реализован подобный вид отладки. Также создан редактор алгоритмов работы роботов Lego, для которого реализована пошаговая интерпретация.

В итоге разработана инфраструктура визуального отладчика в системе QReal, создан каркас модуля для работы с отладчиком целевого языка генерации. Также были реализованы несколько частных примеров визуальных отладчиков. С результатами данной разработки можно ознакомиться по адресу [11]. Автор в ней принимал участие под учетной записью vapolyakov, основной результат соответствует коммиту номер 4a04f25.

Дальнейшее развитие

Данное исследование может быть продолжено по следующим направлениям: расширение и обобщение возможностей визуального интерпретатора или улучшение процесса визуальной отладки.

Для визуального интерпретатора созданы несколько частных примеров, которые позволяют впоследствии их обобщить и перейти на новый уровень абстракции, т.е. сначала добавить интерпретацию текстовых пользовательских языков в редакторе блок-схем, предварительно расширив этот редактор, а затем разрабатывать удобный способ задания логики интерпретацию визуальных пользовательских языков. Также возможна доработка самого интерфейса интерпретатора для большего удобства пользования.

Для визуального отладчика возможно добавление поддержки новых языков (например, python), улучшение способа генерации кода (добавить специальное место для подключения библиотек, генерацию кода, содержащего циклы, сделать поддержку модульности кода и т.п.), улучшение текущего интерфейса так, чтобы он позволял напрямую писать любые команды текстовому отладчику. Также возможно добавление визуализации уже написанного кода с целью удобства его отладки.

Также возможно более тщательное изучение статей, посвящённых DMM-подходу в описании семантики исполнения объектно-ориентированных языков (в частности статьи, рассмотренной в данной работе), чёткая формулировка требований к интерпретатору и отладчику и поиск уже существующих инструментальных средств, которые могут помочь в реализации итоговой цели.

Список литературы

1. AMUSE, http://www.lieberlieber.com/en/our-offering/amuse.html
2. Borland Together, http://www.borland.com/us/products/together/
3. Cameo Simulation Toolkit, http://www.magicdraw.com/simulation
4. EMF, http://www.eclipse.org/modeling/
5. EPD, http://www.eclipse.org/eclipse/debug/index.php
6. EProvide, http://eprovide.sourceforge.net/Welcome.html
7. GMF, http://www.eclipse.org/modeling/gmp/
8. GROOVE, http://groove.sourceforge.net/groove-index.html
9. Rational Software Architect, http://www.ibm.com/developerworks/rational/products/rsa/
10. MagicDraw, http://www.magicdraw.com/
11. QReal, https://github.com/qreal
12. SparxSystems Enterprise Architect, http://www.sparxsystems.com/products/ea/index.html
13. UniMod, http://unimod.sourceforge.net/index.html
14. http://community.sparxsystems.com/tutorials/simulation/overview-amuse
15. http://mindstorms.lego.com
16. http://publib.boulder.ibm.com/infocenter/rsahelp/v8/index.jsp
17. http://unimod.sourceforge.net/viewlet/debugger-demo-eng.html
18. https://www.ibm.com/developerworks/university/academicinitiative/
19. http://www.lieberlieber.com/amuse/help/
20. http://www.magicdraw.com/files/viewlets/cst_viewlets_ui_simulation_viewlet.html
21. Anders Ek, Model Simulation in Rational Software Architect: Activity Simulation
22. Anders Ek, Model Simulation in Rational Software Architect: Sequence Diagram Simulation
23. Mattias Mohlin, Model Simulation in Rational Software Architect: Simulating UML Models
24. Mattias Mohlin, Model Simulation in Rational Software Architect: State Machine Simulation
25. Nils Bandener, Visual interpreter and debugger for dynamic models based on the Eclipse platform
26. А.С. Кузенкова, Ю.В. Литвинов, Т.А. Брыксин, Метамоделирование: современный подход к созданию средств визуального проектирования
27. В.А. Поляков, Т.А. Брыксин, Разработка визуального интерпретатора моделей в системе QReal // материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования". СПб.: Изд-во СПбГПУ, 2011. С. 58
28. В.С. Гуров, Технология проектирования и разработки объектно-ориентированных программ с явным выделением состояний (метод, инструментальное средство, верификация)
29. Т.А.Брыксин, Ю.В.Литвинов, Технология визуального предметно-ориентированного проектирования и разработки ПО QReal