САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра Системного Программирования

Колантаевская Анна Сергеевна

Исследование удобства процесса моделирования на базе DSM-платформы QReal

Бакалаврская работа

Допущена к защите.

Зав. кафедрой:

д. ф.-м. н., профессор А.Н. Терехов

Научный руководитель:

ст. преподаватель Брыксин Т.А.

Рецензент:

ст. преподаватель Литвинов Ю.В.

Санкт-Петербург

2013

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics&Mechanics Faculty

Department of Software Engineering

Anna Kolantaevskaya

An investigation into modeling process usability based on QReal DSM-platform

Bachelor's Thesis

Admitted for defence.

Head of the chair:

professor Andrey Terehov

Scientific supervisor:

Senior Lecturer Timofey Bryksin

Reviewer:

Senior Lecturer Yurii Litvinov

Saint-Petersburg

2013

Введение

Рост программной отрасли в последние десятилетия делает все более актуальным вопрос об оптимизации как труда отдельного программиста, так и процесса разработки в целом. Программная инженерия является дисциплиной, направленной на улучшение производства сложных многомодульных систем высокого качества. Среди прочих средств и подходов к оптимизации можно выделить совершенствование инструментария разработчиков: от улучшения и расширения диапазона средств разработки до парадигм программирования и возможностей переиспользования готовых решений.

Необходимость оптимизации и автоматизации в рамках работы как отдельных разработчиков, так и групп программистов больших компаний привела к появлению так называемых CASE-систем (Computer Aided Software Engineering), определяемых в широком смысле как ”инструменты и методы для поддержки инженерного подхода к разработке программного обеспечения на всех стадиях процесса” [10]. Особую нишу среди подобных решений занимают средства визуального моделирования и программирования: языки и соответствующие им среды разработки.

Считается, что возможность повышения уровня абстракции, достигаемая с помощью визуальных средств программирования, способна существенно упростить процесс создания программ и программных комплексов. Главное преимущество визуальных языков — то, что они позволяют наглядно представить программные структуры, как, например, алгоритмы и данные, в противовес традиционным текстовым языкам программирования, где такие многомерные структуры

закодированы в одномерные строки с помощью достаточно сложного синтаксиса. Визуальные языки добавляют слой абстракции, позволяя программисту непосредственно наблюдать за процессом разработки, манипулируя сложными программными элементами.

Большинство исследований показывают, что использование CASE-инструментов, поддерживающих визуальные средства программирования, может положительно воздействовать на качество ПО и эффективность работы специалистов (см., например, [15]). Использование таких средств визуального программирования в противовес программированию “ручному” имеет много преимуществ, однако, несмотря на сравнительно возросшую популярность CASE-инструментов, существуют определенные сложности, связанные с их введением в производственный цикл. Многие инструменты не используются вовсе, скорость внедрения других в производственный процесс очень невысока ([14], [8]).

Итак, несмотря на очевидные преимущества, в реальном промышленном программировании подобные инструменты практически не используются. Почему же не происходит массового перехода на визуальные средства программирования? По мнению экспертов и разработчиков, существующие на текущий момент реализации визуальных языков неудобны, а потому сложно считать их адекватной альтернативой привычному интерфейсу разработки — текстовому программированию. Какие существуют возможности совершенствования визуальных средств разработки с точки зрения удобства их использования и чем определяется такое удобство?

Целью данной работы является исследование возможных подходов к

повышению удобства процесса моделирования.

На кафедре системного программирования математико-механического факультета СПбГУ уже несколько лет разрабатывается DSM-платформа QReal, позволяющая создавать редакторы предметно-ориентированных визуальных языков. Еще одной задачей данной работы стало дальнейшее расширение возможностей платформы QReal, позволившее бы повысить удобство использования создаваемых на базе нее графических редакторов.

Глава 1. Обзор предметной области

1.1. Использование CASE-средств

CASE-системы (Computer Aided Software Engineering) определяются в широком смысле как ”инструменты и методы для поддержки инженерного подхода к разработке программного обеспечения на всех стадиях процесса” [10]. Под инженерным подходом подразумевается “четко определенная, хорошо скоординированная повторяемая деятельность, имеющая некоторое представление и осуществляемая в соответствии с определенными правилами и заданными стандартами качества" [11]. Изучение и использование таких средств, применимых на некоторых или даже всех этапах процесса разработки, в настоящее время более, чем когда-либо, становится ключевой областью в программной инженерии. Полный жизненный цикл разработки программного обеспечения нуждается в инструментальной поддержке, начиная с фазы анализа предметной области и формализации требований и до проектирования, реализации, генерации кода и отладки, так как программные системы становятся все более сложными, большими, и зачастую такое сопровождение имеет решающее значение. Расходы на программное обеспечение с каждым годом растут, и даже локальные улучшения производительности разработчиков будут означать существенную экономию ([6]).

Большинство исследований показывают, что использование CASE-инструментов может положительно воздействовать на качество ПО и эффективность работы специалистов [15]. Использование таких

CASE-инструментов в противовес “ручному” программированию имеет много преимуществ:

- улучшает понимание системы сложно взаимодействующих между собой объектов, представляя их визуально;
- упрощает процесс создания и изменения произвольных участков программного кода, а также управления отдельными программными модулями;
- дает дополнительные возможности отслеживать изменения переменных, атрибутов и процедур через набор диаграмм;
- позволяет осуществлять итеративный переход от концептуальных моделей на логическом уровне (упрощающих восприятие, как указано выше) к деталям реализации системы;
- способствует повышению эффективности коммуникации разработчиков за счет сопроводительных свойств визуального представления программы.

Однако, несмотря на преимущества использования CASE-инструментов, существуют определенные сложности, связанные с их введением в производственный цикл. Большинство исследований указывает на ограниченность использования CASE-инструментов: многие инструменты не поставляются вовсе, отчасти в связи с высокой стоимостью, отчасти из-за предполагаемой сложности адаптации к производству, в отношении других инструментов можно говорить о лишь очень невысокой скорости внедрения в производственный процесс [14]. Согласно одному из исследований ([20]) лишь 24% компаний использовали CASE-инструменты в разработке. В ходе опроса 53 компаний по данным [9] было обнаружено, что 39 из них (73.5%) не

используют CASE-средства вовсе, из 14 же компаний, пытающихся вводить подобные инструменты в производственный цикл, 5 впоследствии были вынуждены от них отказаться. Интервью ([15]), проведенные среди компаний, заявляющих об использовании CASE-инструментов, показали, что большая часть членов таких компаний по факту такими инструментами не пользуется. В 57% участвующих в исследовании организаций менее чем 25% разработчиков использует CASE-инструменты.

Более современные исследования в целом показывают аналогичные результаты, по данным одного из них [8] в среднем после года внедрения 70% CASE-инструментов не используется вовсе; 25% CASE-инструментов используются лишь отдельными разработчиками; 5% CASE-инструментов широко используются, не актуализируя, однако, полный спектр своих возможностей.

Итак, несмотря на очевидные преимущества, в реальном промышленном программировании CASE-инструменты используются редко. Почему же не происходит массового перехода на новые средства программирования? Кроме ряда причин экономического и организационного характера, многие исследователи выделяют также и сложности технические, в частности — вопросы удобства использования инструментов.

Помимо стратегического решения компании об использовании CASE-технологии существует и решение конкретных разработчиков. Что побуждает разработчика включить новый инструмент в свой арсенал разработки?

Специалист в области дизайна программных систем Джоэль

Спольски писал, чем больше интуитивно понятных, упрощающих использование аспектов содержит инструмент, тем ниже порог входа, тем меньше усилий затрачивается на обучение — и тем больше шансов, что его станут использовать [21]. Авторы [19] и [5] также показывают, что наравне с перспективой повышения производительности не менее важным для программистов доводом, побуждающим к использованию нового инструмента, является удобство его использования, некоторые непосредственные выгоды от использования. Большая же часть существующих на текущий момент CASE-инструментов, по мнению специалистов, неудобны, а потому замена ими текстуального программирования труднодостижима. Так, например, часть опрашиваемых в исследовании [9] разработчиков, имеющих высокие ожидания в отношении прироста производительности, впоследствии были вынуждены отказаться от использования CASE-средств, в том числе вследствие их сложности.

Многие другие исследования также показывают, что CASE-средства являются излишне сложными, тяжелыми в использовании и требующими больших временных затрат на обучение ([15], [17], [18]). Согласно данным опросов, в основном инструменты воспринимаются разработчиками как имеющие высокую степень сложности, что является серьезным барьером для их использования. Авторы [15] приходят к выводу, что, в частности, снижение сложности восприятия интерфейса инструментов способствовало бы повышению их используемости.

Многофункциональность и настраиваемость инструментов создают дополнительные сложности в их освоении. Функциональность CASE-инструментов часто не согласуется с потребностями

пользователей. Некоторые из предоставляемых возможностей не используются, другие оказываются слишком труднодоступными. Авторы [19] показывают, что лишь небольшая часть возможностей CASE-инструментария используется разработчиками, и говорят о необходимости упора на доступность чаще употребимых функций. Исследователи [13] также говорят об актуальности реализации простого и удобного и понятного интерфейса для доступа к наиболее часто используемым функциям в первую очередь.

Также многие авторы говорят о необходимости поддержки творческих процессов ([19], [7], [16]). Сложности в обучении и необходимость разбираться с малопонятными интерфейсам заставляют разработчика сосредоточится на использовании инструментов или их освоении [12]. Ряд ограничений, накладываемых системой, необходимость соблюдения формальных правил заставляет программиста думать о представлении идеи, а не о самой идее, что особенно критично на начальных стадиях процесса разработки, когда внимание сфокусированно в основном на понимании проблемы, мышлении, интуиции. Плохо развитая поддержка разработки на ранних фазах жизненного цикла, ограничивающая проявления потенциала креативности ([15]) — еще одна причина, сдерживающая программистов от использования CASE-средств.

1.2. Язык ДРАКОН

Попытки создавать удобные графические средства разработки уже предпринимались. Рассматриваемый далее язык **ДРАКОН** (*Дружелюбный Русский Алгоритмический язык, Который Обеспечивает Наглядность*) является одним из примеров удобных в использовании языков моделирования ([3]).

ДРАКОН — визуальный алгоритмический язык, разработанный в конце 80-х – начале 90-х годов XX века под руководством Владимира Паронджанова при участии Российской Академии Наук в рамках космической программы Буран. Целью, стоявшей перед разработчиками, было создание единого универсального языка, который должен был объединить в себе свойства языков для систем реального времени, проблемно-ориентированных языков и языков моделирования. Среди прочих задач, стоявших перед разработчиками, были:

- предоставление человеку языковых средств, упрощающие восприятие сложных процедурных проблем для уменьшения вероятности ошибок и роста производительности труда;
- улучшение качества программного обеспечения по критерию "понимаемость алгоритмов и программ".

Отказ от текстовых управляющих структур, используемых в языках высокого уровня, и замена их на управляющую графику, с точки зрения человеческого фактора, сам по себе делает интерфейс взаимодействия программиста и программы более понятным и удобным для человека. Управляющие же структуры языка ДРАКОН по замыслу разработчика специальным образом приспособлены к восприятию, делая, таким

образом, представление программы еще более понятным, а программирование — еще более удобным.

На рис. 1 приведен пример ДРАКОН-схемы, созданной для определения кислотности раствора.

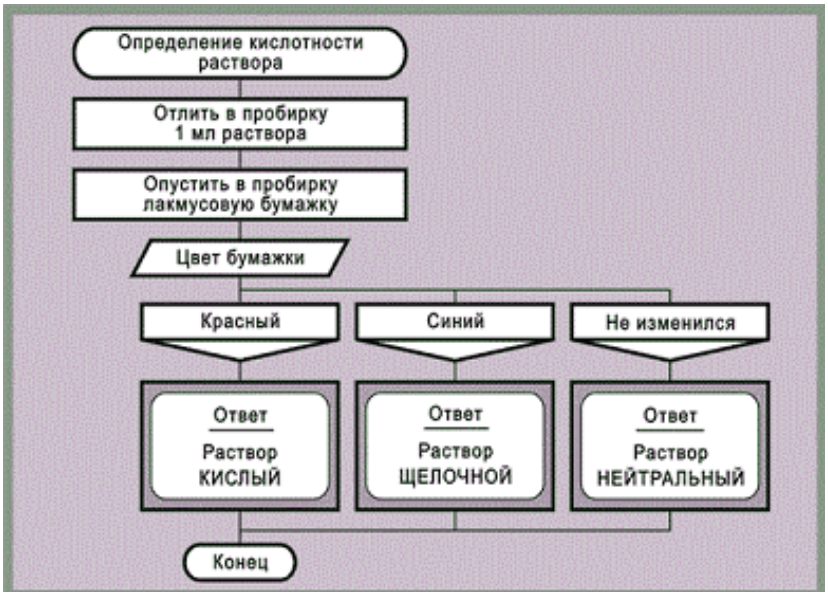

Рис. 1. Пример ДРАКОН-схемы[1].

Как и все прочие языки, ДРАКОН опирается на математику и логику. Однако сверх того, он самым тщательным образом учитывает когнитивные вопросы ([3], [4]). В основе языка ДРАКОН лежит идея когнитивной формализации знаний, позволяющая сочетать строгость логико-математической формализации с точным учетом когнитивных (познавательных) характеристик человека. По мнению создателей языка, именно благодаря систематическому использованию

[1] ‘здесь и далее все рисунки из данной главы взяты из [2].

когнитивно-эргономических методов ДРАКОН приобрел свои уникальные эргономические характеристики.

1.2.1. Процесс моделирования на языке ДРАКОН

Диаграмму на языке ДРАКОН или *ДРАКОН-схему* можно вывести путём исчисления над алфавитом вершин-операторов (*икон*) и словарём подграфов-операторов (*шаблонов или атомов*) из некоторых начальных шаблонов (*заготовок)* и последовательного применения правил языка с учетом существующих ограничений, запрещающих или разрешающих на каждом этапе разработки добавление отдельных элементов в диаграмму и жестко регламентирующих расположение этих элементов.

Создание любой ДРАКОН-схемы начинается с создания заготовки. Заготовки бывают двух видов: одна используется для построения ДРАКОН-схемы “примитив”, из другой получается “силуэт” (рис. 2).

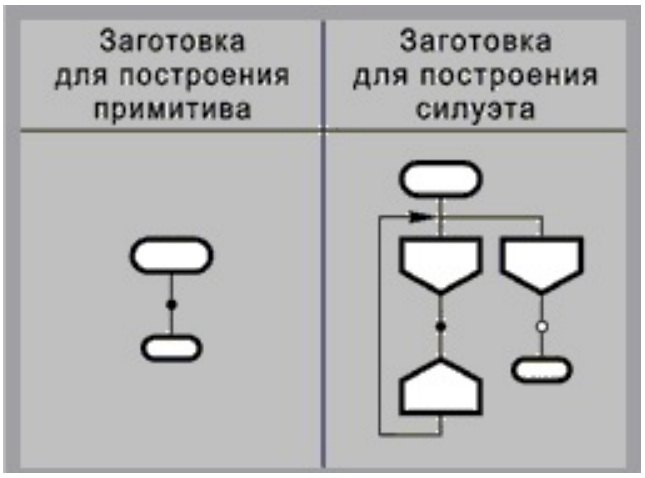

Рис. 2. Заготовки для построения ДРАКОН-схемы.

Первая заготовка — самая простая — может использоваться для

создания “примитивов” — коротких, одноэтапных программ. Программы, разделенные на несколько этапов, носят название “силуэтов”. Каждый этап представляется отдельной веткой. Первым оператором ветки объявляется название этапа. Создание таких программ начинается с использования заготовки для силуэта либо с заготовки для примитива, но с последующим добавлением в схему дополнительных этапов работы программы.

После выбора заготовки начинается собственно конструирование схемы. В основе метода конструирования лежит возможность добавления новых элементов в уже существующий начальный прототип схемы посредством операции *ввод атома.*

Атомом называется элемент языка, который за одну транзакцию мы можем добавить в схему. Атомы являются основными синтаксическими единицами языка ДРАКОН — словами или операторами, составляющими любую схему. Атомы языка делятся на простые и составные, функциональные и нефункциональные. Простые содержат одну икону, составные атомы (или шаблоны) не менее двух. Функциональными являются все атомы, кроме оператора комментария. Полный список атомов см. рис. 3.

Операция ввода атома выполняется в два этапа: сначала пользователь выбирает нужный атом, затем обращается к дракон-схеме и указывает точку, в которую нужно его ввести. Ввод атома производится так: происходит разрыв соединительной линии в выбранной пользователем некоторой точке, после чего в место разрыва вставляется атом, как показано на рис. 4.

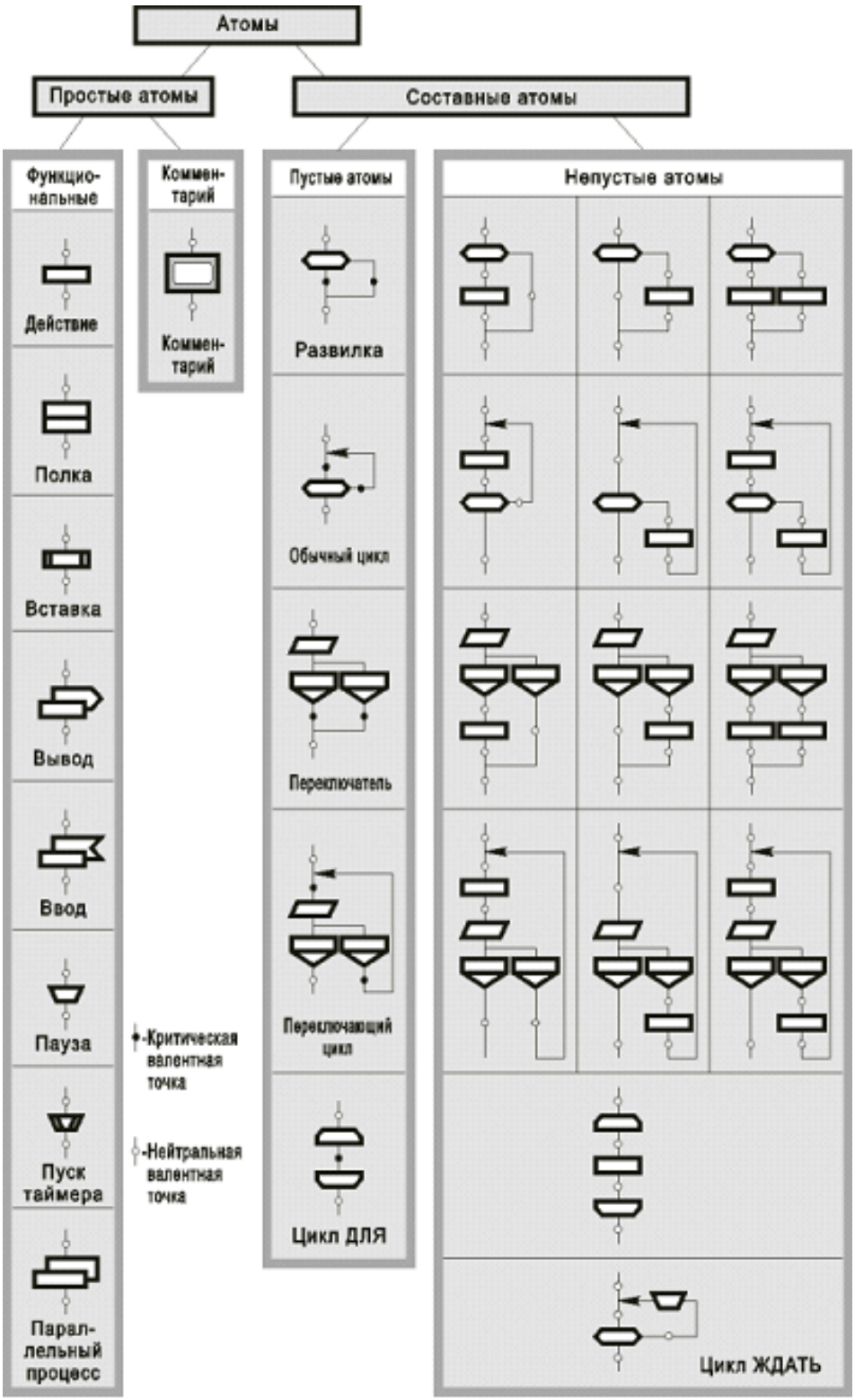

Рис. 3. Атомы языка ДРАКОН.

Атомы вставляются только в разрешенные места, которые называются *валентными точками* дракон-схемы. Перечень точек включает:

- валентные точки заготовок;
- валентные точки на составных иконах;
- входы и выходы атомов.

Валентные точки схемы делятся на нейтральные и критические. Точка является нейтральной, если применение операции “ввод атома” к данной точке является возможным, но не обязательным. В отличие от нее критическая точка требует обязательного ввода атома.

Таким образом, если на схеме на некотором этапе процесса моделирования появляется критическая валентная точка, то следующим шагом разработчика должен стать ввод атома в этот участок ДРАКОН-схемы. При этом точки входа и выхода этого нового атома становятся нейтральными.

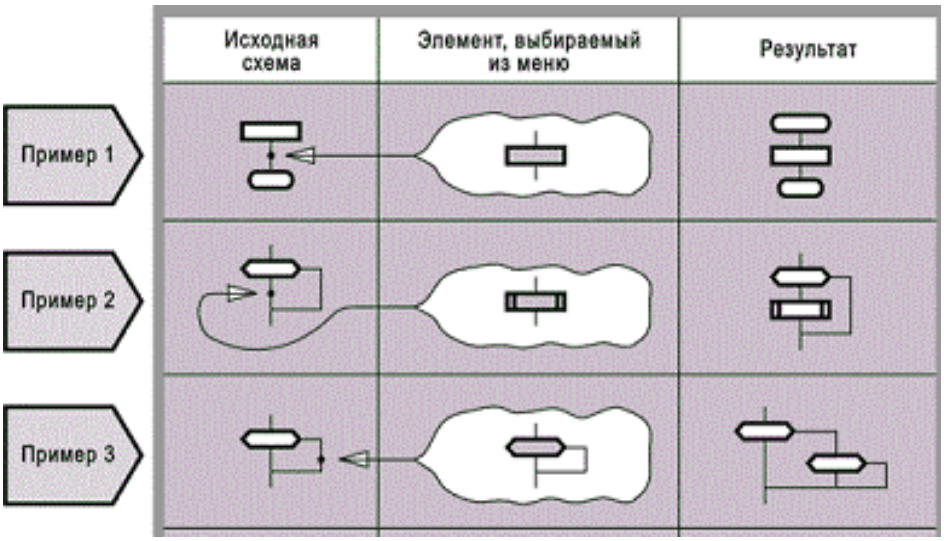

Рис. 4. Примеры операций ввода атома.

Если на схеме в какой-то момент появляется несколько критических точек, то одна из них на следующем этапе конструирования должна быть заполнена, оставшаяся точка становится нейтральной.

Ниже приведен пример конструирования ДРАКОН-схем на базе заготовки “примитив” (рис. 5).

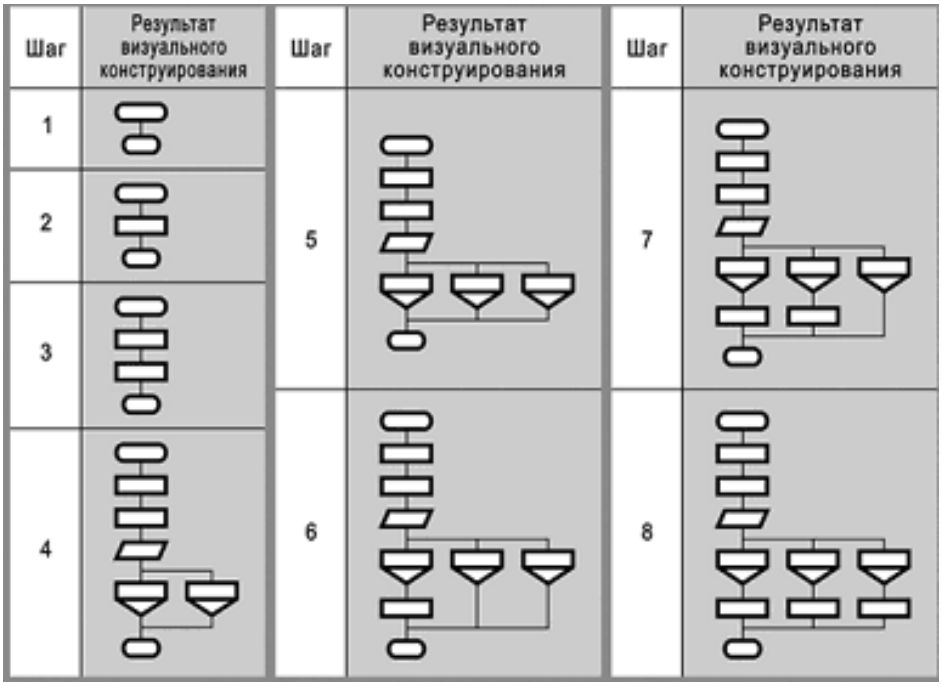

Рис. 5. Конструирование ДРАКОН-схемы на базе заготовки “примитив”.

1.2.2. Визуальные особенности моделирования на языке ДРАКОН

Касаясь процесса создания программы, нельзя не сказать отдельно о связанных с ним непосредственно способах укладки элементов

программы на рабочую область: от расположения элементов диаграмм на листе напрямую зависит их читабельность (для сравнения см. рис. 6 — разные варианты “раскладок” для одной и той же диаграммы).

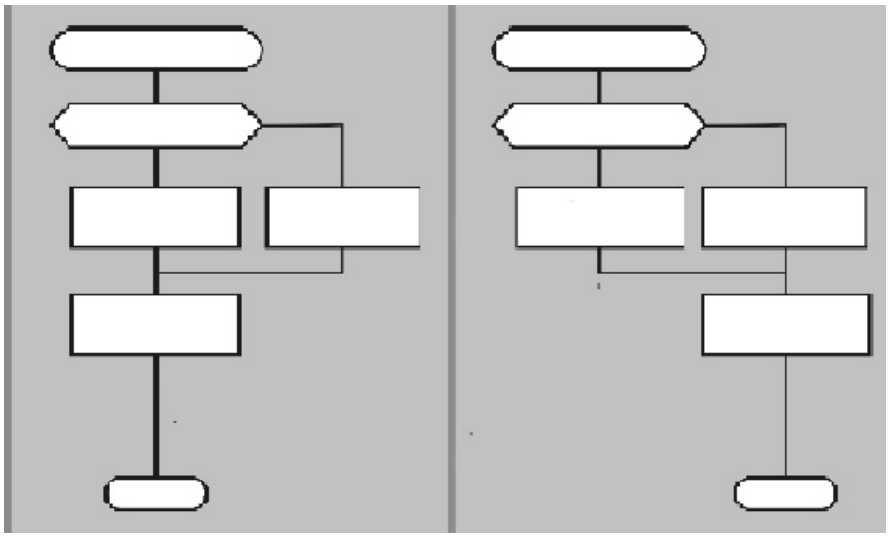

Рис. 6. Пример разных “раскладок” диаграммы.

Исчисление, разработанное для ДРАКОНа, называется *шампур-методом*. Оно даёт возможность строить «слепыш» алгоритма, последовательно применяя правила построения к базовым заготовкам-шаблонам, получая на каждом шаге построения ДРАКОН-схему, удобную для восприятия. В основе метода визуальной стороны моделирования лежит небольшое число базовых принципов.

1) Принцип *шампура* — атомы-операторы, следующие в алгоритме *безусловно* друг за другом, т.е. представляющие собой прямую и непосредственную последовательность действий, упорядочиваются по вертикали, так что исполняемая первой икона всегда лежит сверху, последняя — снизу, и все они лежат на одной оси. Пример шампур-блока

смотри на рис. 7.

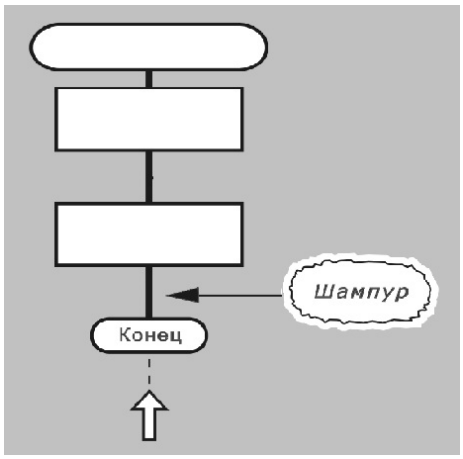

Рис. 7. Пример шампур-блока.

2) Принцип *главной/побочных вертикалей* — выходы развилок, образующиеся при появлении условных операторов или ветвлений, упорядочены друг относительно друга так, что не лежащий на главной вертикали выход (называемый *побочным*) всегда располагается правее *главного*. То есть среди множества выходов-“развилок”, выделяют основной, главный путь потока управления и располагают его на главной вертикали, прочие же ветви-шампуры - справа от него (рис. 8).

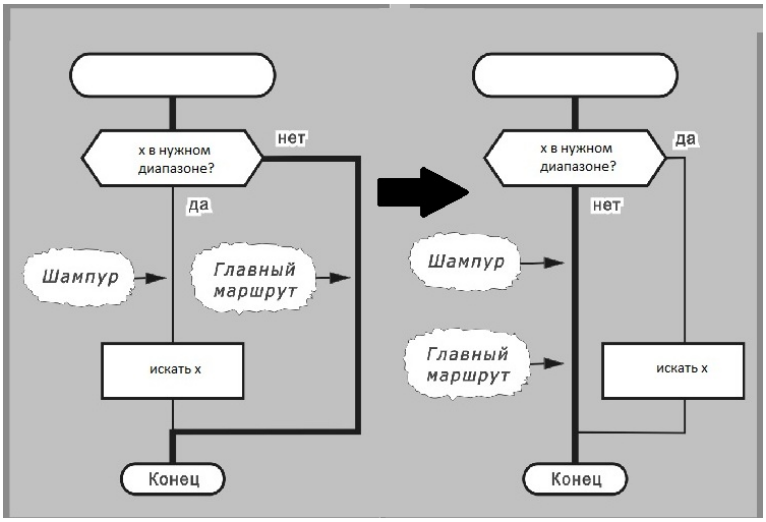

Рис. 8. Иллюстрация принципа главной/побочных вертикалей.

1.3. QReal

Апробацию данной работы было решено осуществлять применимо к системе QReal[2] — платформе предметно-ориентированного моделирования (DSM-платформе), позволяющей быстро и удобно разрабатывать новые визуальные редакторы, ориентированные на специализированные визуальные языки, посредством описания их модели на метаязыке [1]. В проекте QReal уже существует ряд средств, ускоряющих и упрощающих работу проектировщика (редактор форм, распознавание жестов мыши, визуальный отладчик и т.д.). Дальнейшее расширение визуального языка метаредактора QReal некоторыми эвристиками, упрощающими разработку, позволило бы сделать процесс моделирования более удобным для всего ряда редакторов, описанных посредством такого расширенного языка в QReal.

1.3.1. Об архитектуре QReal

QReal — сложная система с многоуровневой архитектурой. Каждый визуальный редактор, создаваемый QReal, 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 (см. рис.9).

[2] http://qreal.ru/

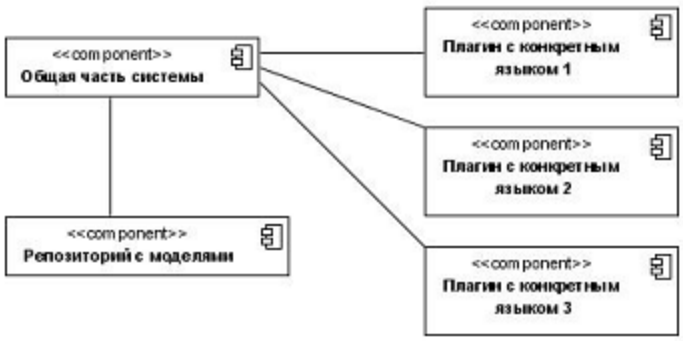

Рис. 9. Представление архитектуры QReal как metaCASE-системы.[3]

Генерация редактора по описанию проходит в несколько этапов, проиллюстрированных на рис. 10.

Рис. 10. Этапы генерации нового подключаемого модуля в QReal.

[3] Рисунок взят из [1].

Метамодель задает синтаксические правила языка. Метамодели различных диаграмм обрабатываются генератором редакторов, генерирующим по ним код на языке C++. Полученный код компилируется вместе с кодом QReal. На выходе компилятора для каждого описанного редактора мы получаем подключаемый модуль, представляющий собой файл динамической библиотеки (один модуль может содержать несколько редакторов), использующийся системой в дальнейшем для визуализации редактора. В подключаемый модуль входят классы, реализующие отображение фигур, содержащие такую информацию о них, как свойства портов, допустимых связей и т.д., и, в соответствии с общим для всех модулей редакторов интерфейсом, предоставляющие доступ к этой информации вовне. Приложение использует полученную информацию для реализации поведения редактора.

1.3.2. О метамодели редакторов

Любой редактор, генерируемый QReal, может быть задан посредством .xml-описания его модели на языке метамодели. Некоторые сущности-элементы, принадлежащие метамодели, описаны ниже.

Diagram — корневой элемент модели описания языка. В его свойствах перечисляется системная информация: расположение генерируемого редактора на диске и тому подобное. Описывается тегом <diagram>;

Editor — сущность редактор. В свойствах этого элемента указывается его рабочее имя и имя, которое отображается при загрузке

нового редактора в среду. Содержит поле nodeName, которое должно содержать имя элемента, который будет являться корневым для диаграмм создаваемого редактора языке. Описывается тегом <editor>. Вкладывается в элемент <diagram>

Node — сущность, описывающая узел на диаграмме. Содержит свойства: name — идентификатор узла, displayedName — отображаемое на палитре элементов имя, необязательное к заполнению свойство description , отвечающее за всплывающую подсказку. Описание формы узла и возможность его масштабирования определяется сущностью shape. Описывается тегом <node>, вкладывается в элемент editor.

Edge — сущность, описывающая связи или ассоциации на диаграмме. Содержит свойства: name — идентификатор ассоциации, displayedName — отображаемое имя ассоциации. Среди необязательных свойств также доступно задание типа линии (прямая, пунктирная и т.д.), тип текста, который будет отображаться на связи. Описывается тегом <edge>. Вкладывается в элемент editor.

Глава 2. Реализация

2.1. Когнитивно-ориентированные эвристические приемы моделирования языка ДРАКОН

Наиболее близким аналогом языка ДРАКОН принято считать язык блок-схем. Но хотя его диаграммы похожи на блок-схемы алгоритмов, фактически ДРАКОН отличается от них как синтаксически (что вовсе не принципиально), так и семантически. Именно посредством семантических отличий, дополнительных, особых ограничений разработчики усовершенствовали процесс конструирования диаграмм. Особые правила моделирования ДРАКОН-схем позволили создать дополнительный, когнитивный или “познавательный” слой абстракции над обычным инструментарием визуального средства программирования, делающий программы более воспринимаемыми, и, соответственно, заложенные в них алгоритмы — упрощенными для понимания.

Можно заметить, что такие когнитивно-ориентированные семантические правила или эвристики языка ДРАКОН не привязаны к его синтаксису непосредственно, т.е. могут быть отделены от него и использованы в любом другом процедурном визуальном языке. Далее описываются некоторые такие отделяемые свойства или правила, которые, введенные в семантику некоторого визуального языка, способствовали бы созданию аналогичного языку ДРАКОН когнитивного слоя абстракции и позволили бы сделать процесс моделирования посредством языка более удобным.

2.1.1. Использование шаблонов

Использование начального прототипа, как и использование шаблонов в целом, относится к группе повышающих скорость и удобство разработки эвристик наиболее естественным образом. Необходимость возможности оперирования в процессе создания схемы помимо атомарных элементов группами элементов очевидна и оправдывается целиком существованием групп элементов, не использующихся отдельно друг от друга (как в циклах, рис. 11), а также групп элементов, постоянно использующихся совместно. Такой поход позволяет сократить количество неконструктивных операций пользователя (например, нажатий мыши), а также дает возможность программистам избежать ряда ошибок (программа без пиктограммы конца, незакрытые циклы).

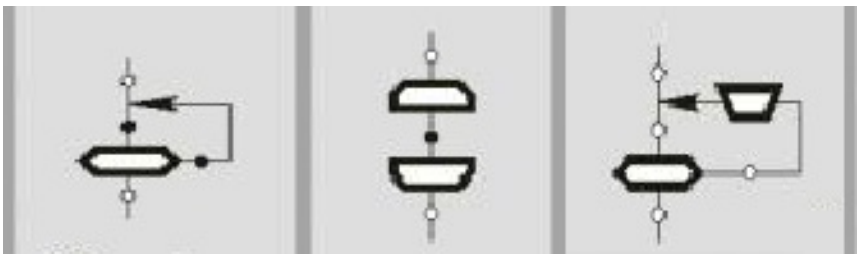

Рис. 11. Примеры групп элементов - шаблонов.

2.1.2. Принцип вложения

В основе процесса конструирования языка ДРАКОН лежит принцип вложения — возможность добавления новых элементов в уже существующий начальный прототип схемы посредством операции “ввод

атома”(см. п.1.2.1). Такая эвристика, реализующая возможность расширять поток управления существующей программы непосредственной в него вставкой допустимого элемента в сочетании с использованием шаблонов, существенно упрощает процесс разработки, позволяя создавать программы инкрементально, а также сократить количество неконструктивных операций пользователя, необходимых для добавления новых ассоциаций на сцену и назначения им элементов-адресатов.

2.1.3. Линейные блоки

Одним из принципов укладки элементов на диаграммы, являющимся основополагающим графическим принципом ДРАКОН’а, является концепция шампура — предполагается, что пиктограммы-операторы, следующие в потоке управления программой друг за другом, т. е. представляющие собой безусловную линейную последовательность операторов, должны лежать на одной вертикальной оси и быть выровненными по ней, первый оператор - самым верхним, последний - самым нижним (см. п.1.2.2, рис. 7). Это несложное правило позволяет существенно повысить воспринимаемость диаграммы на некотором языке, что объясняется склонностью человеческого зрения выделять и воспринимать в первую очередь стройные линейные структуры.

2.1.4. Принцип “главной/побочной вертикалей”

Человек, принадлежащий европейской культуре, в силу особенности чтения (справа-налево и сверху-вниз) склонен и любые последовательности воспринимать аналогичным образом, в порядке следования. На свойстве человеческого восприятия “слева-направо” основано еще одно полезное правило укладки, отделяемое от языка ДРАКОН: принцип главной/побочных вертикалей (см. п.1.2.2, рис. 8). Среди множества выходов некоторой точки или некоторого узла, возникновение которых связано с такими структурами, как “switch-case” или множественным использованием ”if...else”, выделяют главный путь алгоритма и помещают его первым, левее всех прочих, ответвления же помещают чем второстепеннее, тем правее. В любом алгоритме существует некоторый “основной путь”, который должен быть замечен и проанализирован в первую очередь, исключения же, случаи нетипичные, находясь в поле периферийного зрения, рассматриваются как побочные, что позволяет программисту концентрировать внимание на сути алгоритма.

2.1.5. Рокировка маршрутов

Еще одним инструментом, повышающим общее удобство пользования, является эвристика, в терминах ДРАКОНА называемая рокировкой маршрутов. Функциональность рокировки состоит в возможности быстрой автоматизированной перестройки диаграммы за счет смены главного и побочного выходов условного оператора (рис.12).

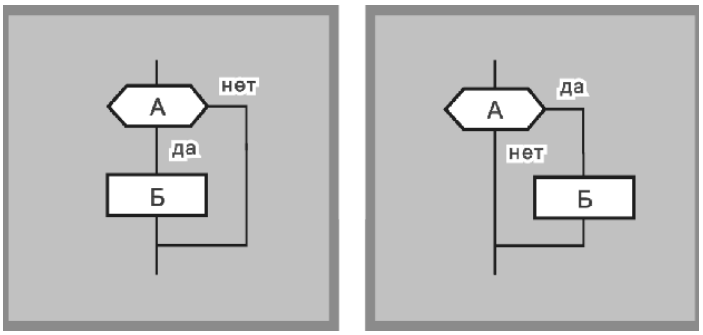

Рис. 12. Пример выполнения операции “рокировка маршрутов”: до — рисунок слева, после — рисунок справа.

2.1.6. “Операции с лианой”

Говоря о раскладке пиктограмм на сцене, также нельзя не вспомнить о приемах, называемых разработчиками ДРАКОН’а “операциями с лианой”. Лианой называется часть диаграммы, имеющая один вход (входящую ассоциацию) и один выход (выходящую ассоциацию). Операция с лианой - это перенос концов входящей или выходящей ассоциации без изменения в прочей структуре программы. Примерами оправданности таких операций могут служить случаи изменения конца блока условного оператора или включения в цикл еще нескольких операторов, находящихся ранее за его пределами. Первым шагом производится отрыв конца лианы от предыдущей точки присоединения, затем конец лианы с помощью вертикальных и горизонтальных линий присоединяется к любой другой доступной точке (см. рис.13).

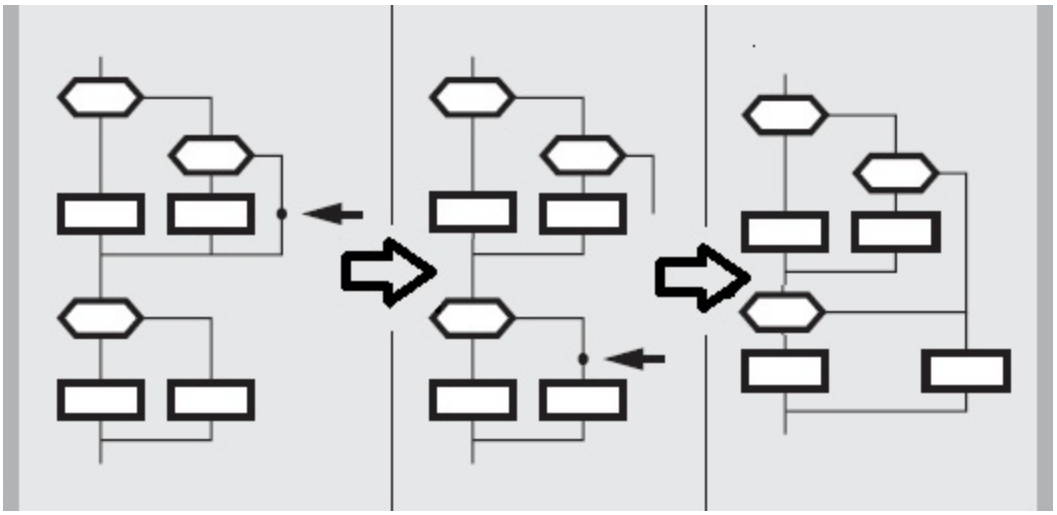

Рис. 13. Пример выполнения операции “пересадка лианы”.

Данная эвристика также позволяет сократить число нажатий клавиш мыши пользователем в процессе моделирования, необходимых для развернутой перестройки циклов либо операций удаления и вставки отдельных операторов в цикл.

2.1.7. Разделение на этапы

Еще одной из когнитивно-ориентированных эвристик является возможность разделить программу на несколько этапов-веток. Такая формализация смыслового разбиения проблемы, алгоритма или процесса на логически выделенные части позволяет не только существенно упростить восприятие программы как последовательности действий, но и вынуждает программиста локализовать основные вехи работы алгоритма. Каждый такой этап программы представляет собой единый логически завершенный линейный участок программы, может представляться отдельной ветвью алгоритма (разделение проблемы на N смысловых частей реализуется путем разбиения алгоритма на N веток). Ветка при

этом должна содержать пиктограмму-название участка программы, последовательность пиктограмм-операторов, составляющих часть программы, и, возможно, адресной пиктограммы, содержащей ссылку на участок кода, чье исполнение следует за исполнением текущего этапа (рис. 14).

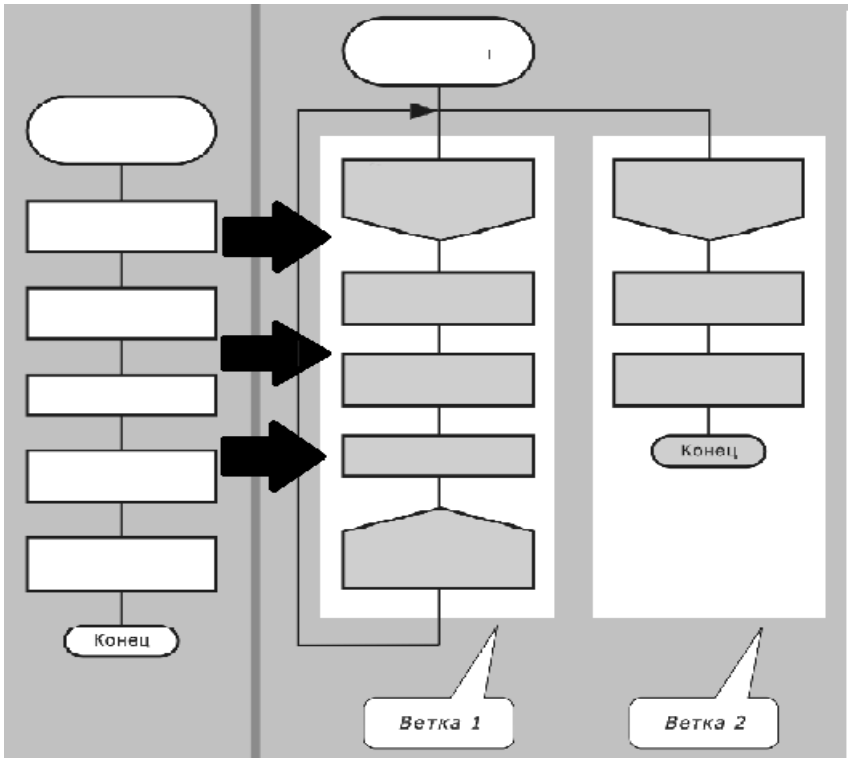

Рис. 14. Пример разделения программы на этапы: рисунок слева — диаграмма до разделения, рисунок справа — после

В том, что касается расположения этих участков-этапов программы на сцене, возможно введение правила “чем правее — тем позже” — нарисованная правее, работает позже всех веток, находящихся левее, если это возможно.

2.2. Реализация эвристик языка ДРАКОН для DSM-платформы QReal

Ранее в рамках курсовой работы были реализованы две эвристики языка ДРАКОН:

1) возможность создания разделяемых ассоциаций;

2) возможность создания сложных элементов-шаблонов.

В данной работе для реализации были выбраны следующие эвристики, расширяющие функционал вышеозначенных:

- корректная вставка в изогнутую ассоциацию;
- сдвиг элементов части диаграммы при вставке в ассоциацию;
- реализация “объединяемых” ассоциаций;
- создание ассоциаций, не связанных с каким-либо узлом (безадресных) в группе;
- создание изогнутых ассоциаций в группе;
- создание множественных элементов в группе (узлов и ассоциаций);
- создание параметризованных количеством элементов в группе.

2.2.1. Разделяемые ассоциации

Существование инкрементального подхода в языке ДРАКОН обеспечивается возможностью расширять поток управления существующей программы непосредственной вставкой в него допустимого элемента. В языке ДРАКОН реализация этого свойства осуществляется за счет так называемых *точек входа* на связывающих операторы-вершины линиях потока управления — точек на ассоциациях,

в которые можно добавлять новую вершину

По аналогии с точками входа в QReal существуют разделяемые связи-ассоциации, обладающие аналогичной функциональностью (см. рис.15).

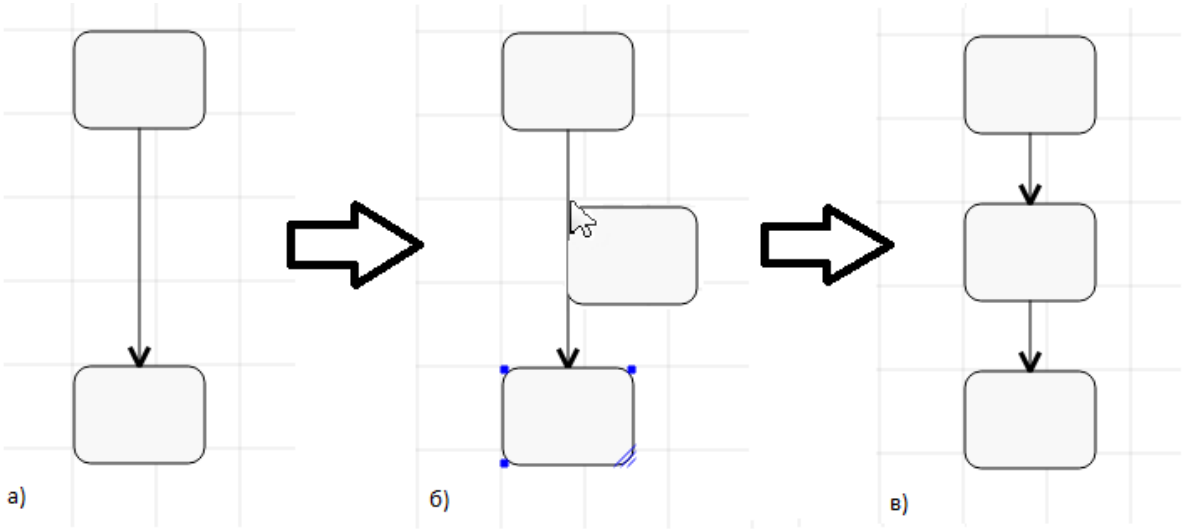

Рис. 15. Пример разделения ассоциации в QReal:
а) на сцене два элемента, соединенных ассоциацией; б) на сцену добавляется элемент из палитры на ассоциацию; в) ассоциация разделяется, элемент попадает внутрь потока управления.

2.2.1.1. Корректная вставка в изогнутую ассоциацию

Изогнутые ассоциации — это связи, представляющие собой не прямой отрезок, а ломанную линию (направленную либо нет). Первоначальная реализация разделяемых ассоциаций не предполагала корректной вставки в такие связи, см. рис. 16.

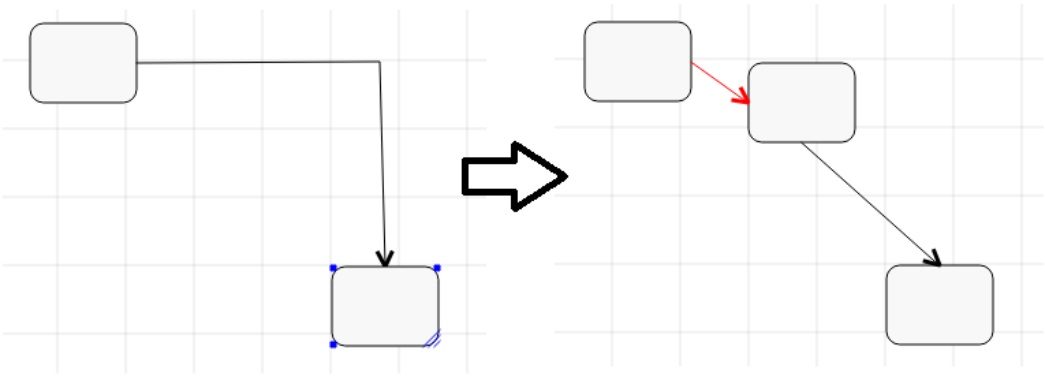

Рис. 16. Первоначальное разделение изогнутой ассоциации при добавлении внутрь нее элемента: диаграмма до (слева) и после вставки (справа).

Задачей была реализация ожидаемого результата вставки внутрь такой ассоциации: разделение отрезка кривой, на который попадает добавляемый элемент, см. рис. 17.

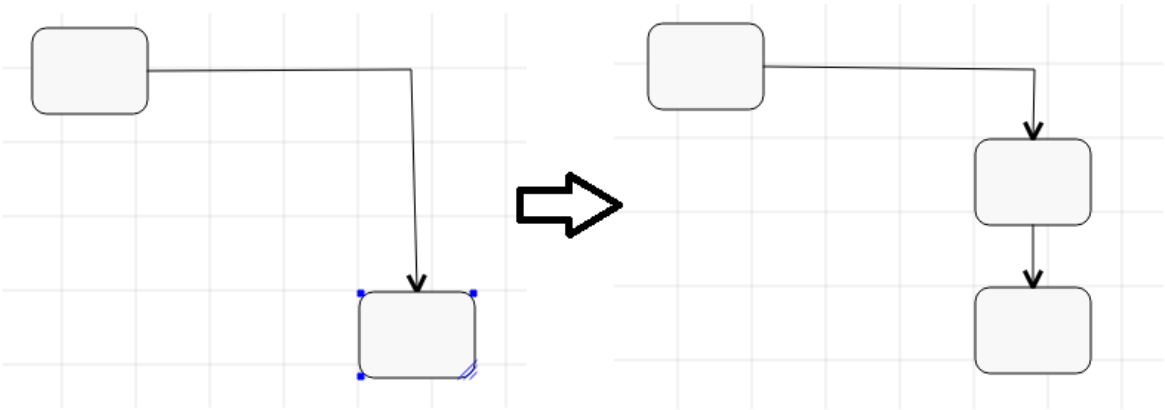

Рис. 17. Корректное разделение изогнутой ассоциации при добавлении внутрь нее элемента: диаграмма до (слева) и после вставки (справа).

В рамках решения такой задачи была модернизирована функция вставки.

2.2.1.2.Сдвиг части диаграммы при вставке элемента

В языке ДРАКОН вставка элемента внутрь линии потока управления программы предполагает неизменность удобной для визуального восприятия раскладки элементов на сцене: не только ассоциация разделяется на две, но и все прочие существующие на диаграмме элементы осуществляют сдвиг, освобождая место для нового элемента.

Первоначальная реализация разделяемых ассоциаций допускала возможность наложения узлов друг на друга при вставке, что особенно заметно для вставки групп элементов (см. рис. 18).

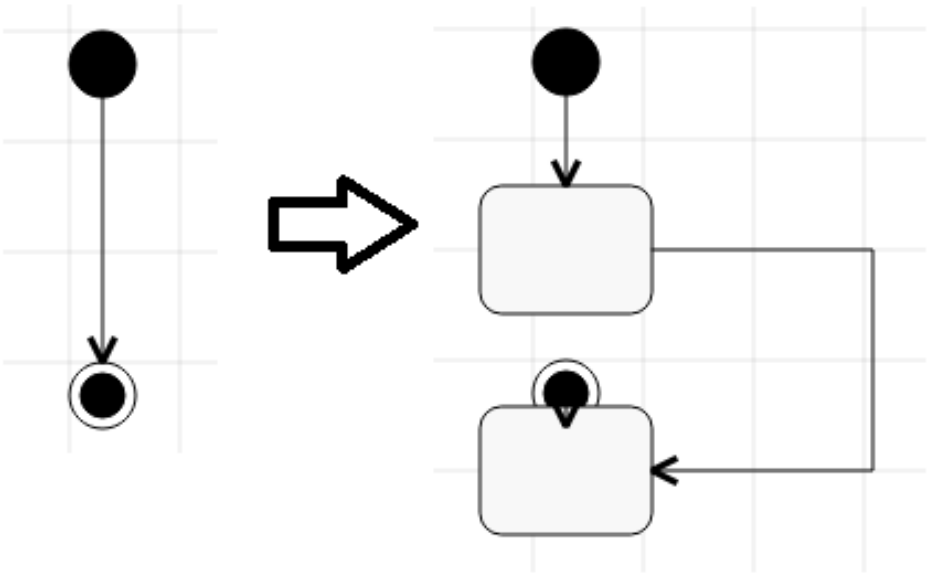

Рис. 18. Пример операции вставки элемента внутрь ассоциации: часть диаграммы до (слева) и после (справа) вставки элемента.

Было необходимо реализовать такую перераскладку элементов, которая, с одной стороны, оставляла бы неизменной (по возможности) пользовательскую логику расположения элементов на сцене, а с другой стороны, обеспечивала бы неналожение вставляемого элемента на уже существующие. Задача реализации такой перераскладки потребовала расширения функциональности функции вставки. Теперь часть диаграммы, лежащая ниже вставляемого элемента, сдвигается вниз в направлении разделяемой ассоциации на размер пересечения ее с вставляемым элементом (см. рис. 19).

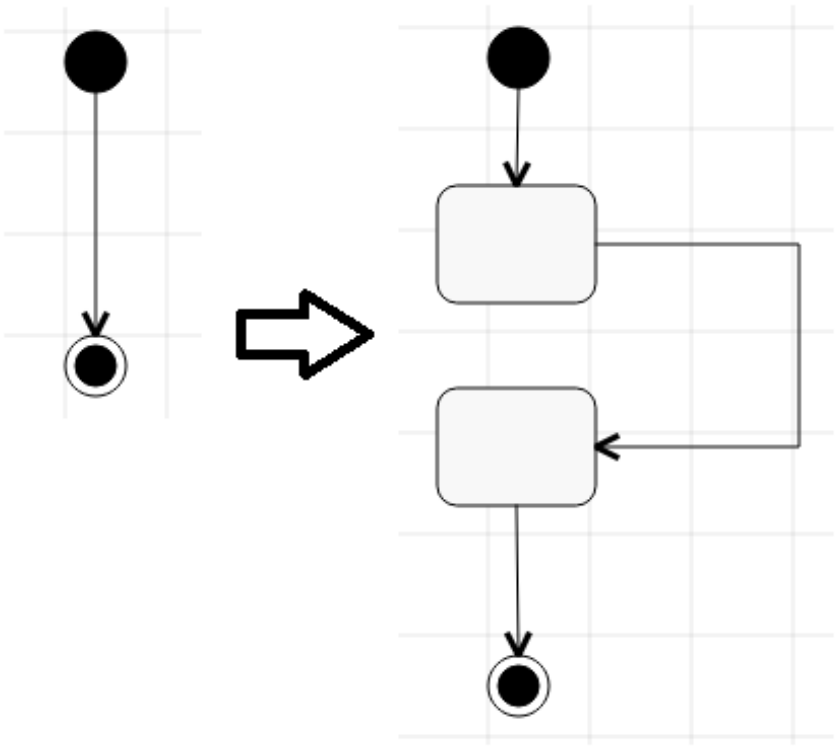

Рис. 19. Пример вставки элемента внутрь ассоциации с раздвижением
диаграммы: часть диаграммы до (слева) и после (справа) вставки.

2.2.1.3. Реализация объединяемых ассоциаций

По аналогии с возможностью расширения потока управления добавлением некоторого элемента, естественным продолжением стратегии инкрементальности подхода к моделированию является возможность удаления элемента диаграммы без нарушения целостности программы. Задачей стала реализация эвристики “объединяемых” ассоциаций для линейных участков программы: когда участок программы линеен, можно предположить, что после удаленного оператора управление должно быть передано следующему за ним.

Линейными участками можно считать цепочки непосредственно (без ветвлений) последовательно выполняющихся операторов, с возможным началом цепочки в узле схождения ветвлений и конце ее в узле, из которого начинается ветвление включительно.

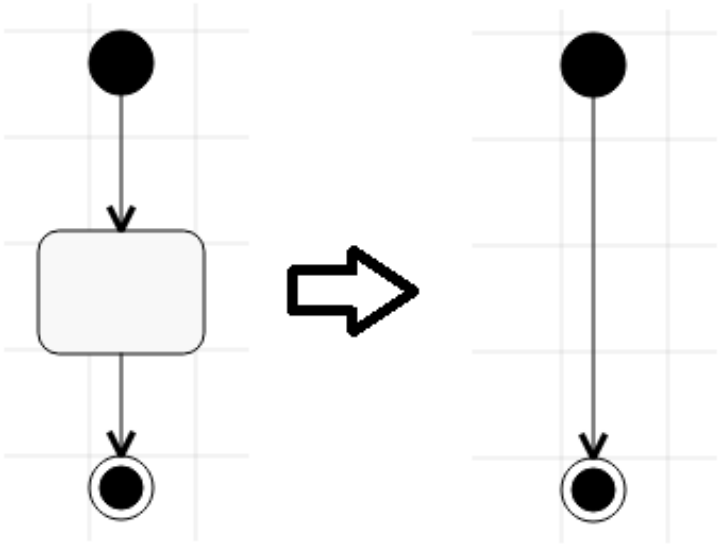

Рис. 20. Пример операции объединения ассоциации при удалении второго в потоке управления элемента: диаграмма до (слева) и после удаления элемента (справа).

Таким образом функция удаления элемента была расширена дополнительной функциональностью: объединением ассоциации при условии линейности участка программы. Пример операции объединения см. на рис. 20.

2.2.2. Создание группы элементов

Существенно упрощает разработку возможность оперирования в процессе создания схемы, помимо элементов атомарных, шаблонами — группами элементов. В QReal существует возможность описания в модели редактора, кроме одиночных элементов, простых логических единиц-шаблонов с последующим созданием по ним шаблонов для палитры и раскладкой их при добавлении на сцену.

Модернизации этой возможности дополнительными эвристиками потребовали расширения функциональности системы QReal на всех уровнях архитектуры:

1. расширение языка описания метамоделей языков;
2. добавление дополнительной обрабатывающей функциональности компилятора xml-метамоделей;
3. расширение интерфейса взаимодействия классов диаграмм с редактором;
4. расширение слоя, отвечающего за поэлементный “разбор” группы элементов;
5. корректировка обработки добавления элемента-группы с разложением на сцене.

2.2.2.1 Создание безадресных ассоциаций в группе

В существующей к моменту начала текущей работы над эвристиками реализации создания элементов-групп в QReal для описания ассоциации в группе было необходимо указывать помимо типа и названия связи ее узел-начало и узел-конец, т.е. ассоциация должна была исходить из некоторой вершины и в некоторую вершину входить. Однако в действительности в шаблонах часто есть необходимость в создании так называемых “безадресных” ассоциаций, не имеющих узла-начала и/или узла-конца.

Для реализации такой эвристики в соответствии со схемой раздела 1.3 система была расширена на всех уровнях архитектуры. В частности язык метамодели был дополнен конструкциями вида:

```
<fromPoint pointX = "200" pointY = "0"> </fromPoint>
<toPoint pointX = "200" pointY = "100"> </toPoint>
```

Теги <fromPoint> и <toPoint> содержат описание относительных (относительно начала координат группы) координат точек начал или концов. Теперь если не указан узел-начало для адресата или узел-конец, необходимо добавить внутри описания ассоциации описание координаты точки начала или конца соответственно. Если не задан ни узел источник, ни узел-сток, необходимо описать обе точки.

Примеры раскладки на сцене групп, содержащих безадресные ассоциации, см. на рис. 21.

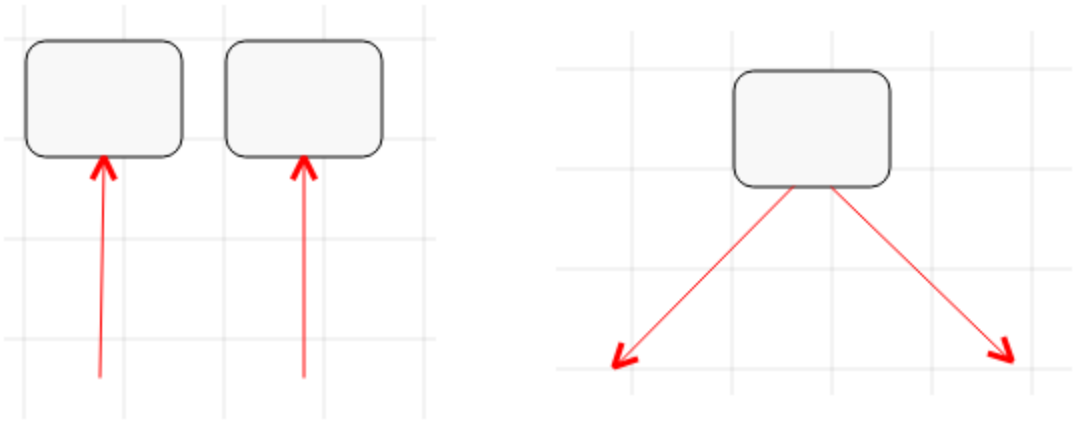

Рис. 21. Пример создаваемых групп, содержащих безадресные ассоциации.

2.2.2.2. Создание изогнутых ассоциаций в группе

Как уже было сказано, изогнутые ассоциации — это связи, представляющие собой не отрезок прямой линии, а ломанную линию. Существование таких ассоциаций является хорошей эвристикой, упрощающей зрительное восприятие.

В QReal уже существовала возможность добавления сгиба для ассоциации на сцене. Задачей стала реализация возможности описания таких изогнутых связей в модели редактора для групп и создания их в палитре.

Реализация потребовала внесения изменений на все уровни архитектуры в соответствии с разделом 1.3. В частности, метамодель была расширена тегом <point>, содержащим описание положения точки относительно начала координат группы. Указываемое внутри тега <groupEdge> множество тегов <point> задает последовательность точек сгиба связи.

Пример описания группы, содержащей изогнутую ассоциацию, и ее раскладки при добавлении на сцену см. ниже.

```
<group name = "atom3_1" inNode = "start" outNode = "end">
<groupNode type = "DragonActionNode" name = "start" xPosition = "0"
yPosition = "0"</groupNode>
<groupNode type = "DragonActionNode" name = "end" xPosition = "0"
yPosition = "100"> </groupNode>
<groupEdge type = "DragonFlow" from = "start" to = "end">
<point pointX = "100" pointY = "0"> </point>
        <point pointX = "100" pointY = "100"> </point>
  </groupEdge>
</group>
```

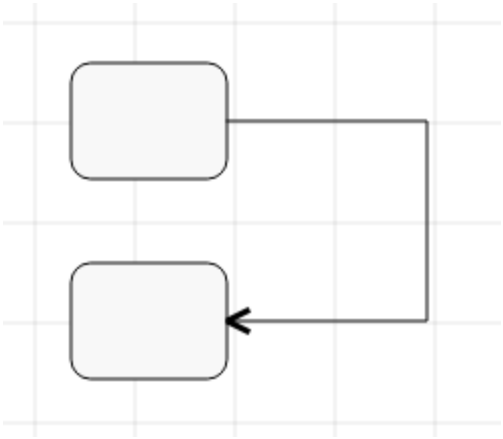

Рис. 22. Пример разложенной на сцене группы, содержащей изогнутую ассоциацию.

2.2.2.3. Создание множественных элементов в группе

Иногда в разработке требуется создание группы-шаблона, содержащей некоторое количество однотипных элементов. Ранее в QReal такая структура потребовала бы описания каждого элемента отдельно, что могло бы быть слишком громоздким для большого числа таких элементов. Еще одной выбранной к реализации эвристикой было обеспечение возможности простого описания в группе нескольких однотипных элементов.

Как и предыдущие, задача также потребовала расширения системы в соответствии со схемой на всех уровнях архитектуры. Отдельно следует сказать о расширении языка метамодели и раскладке шаблона на диаграмме. Теперь в модели метаредактора для описания множественного узла группы необходимо в его свойствах указывать необходимое количество элементов и величину сдвига элементов относительно друг друга:

```
<groupNode type = "DragonActionNode" name = "start" xPosition = "0"
  yPosition = "0" quan = "2" shiftX = "200" shiftY =  "0"> </groupNode>
```

По умолчанию количество считается равным 1, сдвиг: x = 100, y = 0.

Для ассоциации теперь необходимо указывать тип соединения. Вычисляемым по умолчанию типом считается “everyToEvery”, предполагающий соединения ассоциациями каждый узел-источник и каждый узел-адресат. Если вместо источника или адресата окончанием

должна являться точка, то приводится описание списка таких точек, либо описание положения первой точки, количества точек и сдвига каждой очередной.

Если указан тип “oneToOne”, тогда каждый узел-адресат либо точка будет связан только с одним узлом или точкой-адресатом. Если их несколько, то первый соединяется с первым, второй со вторым и т.д., таким образом необходимым условием корректности такой группы является равенство единиц-источников и единиц-адресатов.

Для безадресных ассоциаций с типом “oneToOne” нет необходимости описывать все точки: вместо координат начала (в случае отсутствия узла-источника) либо вместо координаты конца (для остальных случаев) указывается относительный сдвиг точки от узла-окончания, либо точки или узла конца — для каждой такой точки.

Ниже представлены примеры созданных по описаниям групп, содержащие ассоциации с разными типами связей.

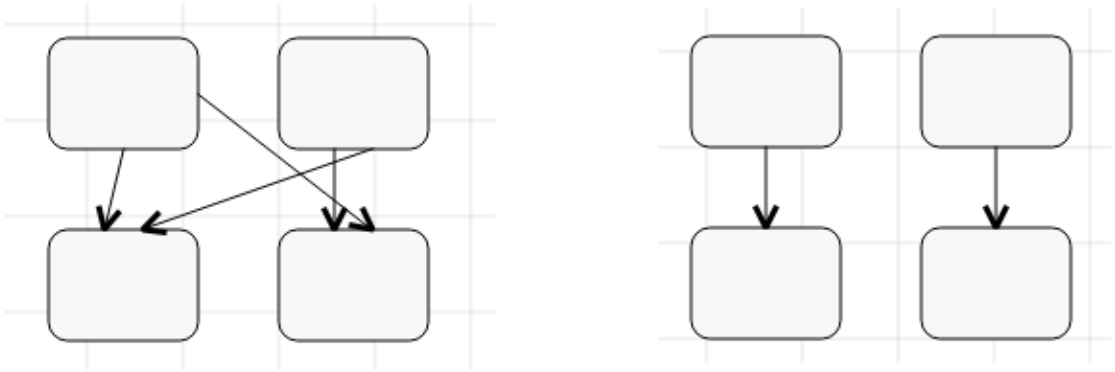

Рис 23. созданные по описаниям группы, тип ассоциации:
слева - “everyToEvery”, справа -“oneToOne”

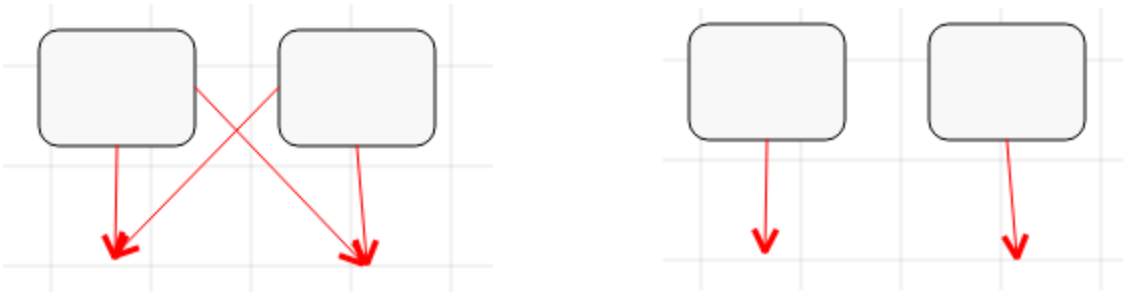

Рис 24. созданные по описаниям группы, тип ассоциации:
слева -“everyToEvery”, справа - “oneToOne”

Все координаты, указанные в описании группы являются относительными (отсчитывая от начала координат группы). Раскладка на экране осуществляется относительно точки вставки на вычисленные координаты.

2.2.2.4. Создание параметризованных количеством элементов в группе

Часто заранее нельзя определить необходимое количество какого-то элемента в группе заранее, как, например, в случае оператора “switch-case” нельзя угадать количество необходимых пользователю веток case. Хорошей модернизацией вышеописанной эвристики является добавление возможности задания пользователем количества создаваемых элементов некоторого типа при добавлении группы на сцену.

В метамодель в описание узлов или безадресных ассоциаций добавлено необязательное свойство “parameterized”:

<groupNode type = "DragonActionNode" name = "start" xPosition = "0" yPosition = "0" quan = "2" parametrized = "true"> </groupNode>

При указании значения “true” этого свойства, можно не описывать количество или относительный сдвиг точки или узла — они должны быть введены пользователем в поле диалогового окна.

Заключение

В ходе выполнения выпускной квалификационной работы был сделан обзор существующих исследований, касающийся использования CASE-инструментов в промышленном программировании. Также была сделана попытка выделить основные черты, определяющие неудобство процесса моделирования посредством таких инструментов.

Кроме того, исследованы отдельные подходы к повышению удобства процесса моделирования. Рассмотрен язык ДРАКОН как пример удобного в использовании визуального средства. Отдельные когнитивно-ориентированные приемы ДРАКОН’а были выделены и обобщены для более широкого класса языков.

Часть эвристик языка ДРАКОН была реализована для DSM-платформы QReal и апробирована группой разработчиков инструмента.

Список используемой литературы

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

[2] Паронджанов В. Д., Графический синтаксис языка ДРАКОН, 1995

[3] Паронджанов В.Д., Как улучшить работу ума, М.: Дело, 2001

[4] Паронджанов В.Д., Перспективы информационных технологий и повышение продуктивности интеллектуального труда // НТИ. Сер. 1., 1993

[5] Aaen, I., CASE Tool bootstrapping (how little strokes fell great oaks) // Next Generation CASE Tools. K. Lyytinen, and V-P. Tahvanainen, Eds. IOS, Netherlands, 1992

[6] Bandinelli S., Fuggetta A., Grigolli S., Process Modeling in the Large with SLANG, 1993

[7] Damm, C., Hansen, K., Thomsen, M., Tyrsted, M., Creative object-oriented modelling: Support for intuition, flexibility and collaboration in CASE tools // In Proceedings of ECOOP2002, 2000

[8] Dutta, S., M. Lee, L. V. Wassenhove, Software Engineering in Europe: A study of best practices // IEEE Software, 16(3), 1999

[9] Elshazly, H., Grover, V., A Study on the Evaluation of CASE Technology // Journal of Information Technology Management (4:1), 1993

[10] Forte, G., Norman, R.J., A self-assessment by the software engineering

community // Communications of the ACM, Vol. 35, No. 4 (April), 1992

[11] Fowler, Lynne, Armarego, Jocelyn. Allen, Maurice, CASE tools: constructivism and its application to learning and usability of software engineering tools // Computer Science Education, 2001

[12] Greene, S. L., Characteristics of applications that support creativity // Communications of the ACM, 45(10), 2002

[13] Heena, Ranjna, A Comparative Study of UML Tools // Association for Computing Machinery, 2011

[14] Holt, J.D., Current Practice in Software Engineering - A survey // Computing and Control Engineering Journal, 8(4), 1997

[15] Iivari, J., Why are CASE Tools Not Used? // Communications of the ACM (39:10), 1996

[16] Jarzabek, S., Huang, R., The case for user-centered CASE tools // Communications of the ACM, Volume 41 Issue 8, ACM New York, NY, USA, 1998

[17] Kermerer, C.F., How the learning curve affects CASE tool adoption // IEEE Software, 9(3), 1992

[18] Kelter, U., Monecke, M., Schild, M., Do we need 'agile' software development tools? // In Proceedings of the Net.ObjectDays, 2002

[19] Lending, D., Chervany., N. L., The use of CASE tools // In Proceedings of the 1998 ACM SIGCPR Conference, 1998

[20] Necco, C.R., Tsai, N.W., Holgeson, K.W., Current Usage of CASE Software // Journal of Systems Management, 1989

[21] Spolsky, J., User Interface Design for Programmers // Apress, 2001