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

Реализация эвристик языка ДРАКОН в metaCASE-средстве QReal

Курсовая работа студентки 361 группы
Колантаевской Анны Сергеевны

Научный руководитель

Т.А. Брыксин

ст. преп.

Санкт-Петербург
2012

Введение

В связи с активным развитием программной индустрии в последние десятилетия, все более актуальным становится вопрос как об оптимизации процесса разработки программного обеспечения в целом, так и о повышении эффективности работы каждого программиста в отдельности. Среди прочих средств и подходов можно выделить совершенствования инструментария разработчиков, от парадигм программирования и возможностей переиспользования готовых решений до улучшения или расширения диапазона средств разработки. В спектре подобных решений подробнее нами будут рассмотрены средства визуального программирования: языки и соответствующие среды разработки.

Считается, что средства визуального программирования, как возможность повышения уровня абстракции, способны существенно упростить процесс создания программ и программных комплексов. Они могут быть использованы на всех этапах жизненного цикла разработки программного обеспечения, от анализа предметной области и формализации требований до проектирования, реализации, генерации кода и отладки. Главное преимущество визуальных языков программирования - 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 этот слой абстракции, позволяя программисту, непосредственно наблюдать и манипулировать сложными программными структурами.

Однако в реальном промышленном программировании подобные средства практически не используются. Почему не происходит массового перехода на визуальные средства программирования? По мнению экспертов и разработчиков, существующие на текущий момент реализации графических языков неудобны – а потому не могут служить адекватной альтернативой привычному интерфейсу разработки – текстуальному программированию.

Постановка задачи.

Таким образом, возникает вопрос – какие существуют возможности совершенствования визуальных средств разработки с точки зрения удобства их использования разработчиком? Чем определяется такое удобство?

Специалист в области дизайна программных систем Джоэль Спольски писал (см. [7]), что система является удобной в использовании, если различные аспекты ее взаимодействия с пользователем соответствуют пользовательским ожиданиям. 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. Именно такие интуитивно понятные свойства языков и нуждаются в изучении.

Попытки создавать в самом деле удобные графические средства разработки имели место быть. Мной был рассмотрен язык **ДРАКОН** (*Дружелюбный Русский Алгоритмический язык, Который Обеспечивает Наглядность*) - визуальный алгоритмический язык программирования, разработан в рамках космической программы “Бурам” в 80-е годы. Одна из особенностей - язык создавался как средство, должное быть доступным к использованию в том числе непрофессиональными программистами - инженерами, учеными. Разработчики ДРАКОН’а полагают, что правила языка по созданию диаграмм оптимизированы для восприятием алгоритмов человеком. Таким образом, язык предлагается в качестве инструмента, способствующего повышению производительности работы.

Итак, было принято решение проанализировать семантику написания программ на языке ДРАКОН, выделить отдельные аспекты и по возможности апробировать их реализацию на некоем произвольно взятом языке. На кафедре системного программирования математико-механического факультета СПбГУ уже несколько лет разрабатывается CASE-система QReal, чьей особенностью является возможность по описаниям предметно-ориентированных визуальных языков создавать редакторы. Задачей стало расширение визуального метаредактора QReal с редактором графического представления новыми, выделенными из спектра средств языка ДРАКОН эвристиками.

1. Язык Дракон

ДРАКОН - визуальный алгоритмический язык, разработанный в конце 80-х – начале 90-х гг. под руководством Владимира Пароннджанова при участии Российской Академии Наук в рамках космической программы Буран. Задачей ставившихся 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 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"

1.1. Удобство

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 специальным образом приспособлены к восприятию, делая, таким образом, представление программы *еще более* понятной, а программирование - еще более удобным.

Как и все прочие языки, ДРАКОН опирается на математику и логику. Однако сверх того, он самым тщательным образом учитывает когнитивные вопросы. 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 когнитивно-эргономических методов ДРАКОН приобрел уникальные эргономические характеристики.

1.2. Определение ДРАКОН-схем.

Дракон-схему можно вывести путём исчисления над алфавитом вершин-операторов(*икон*) и словарём подграфов-операторов(*шаблонов или атомов*) из аксиомы-заготовки.

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

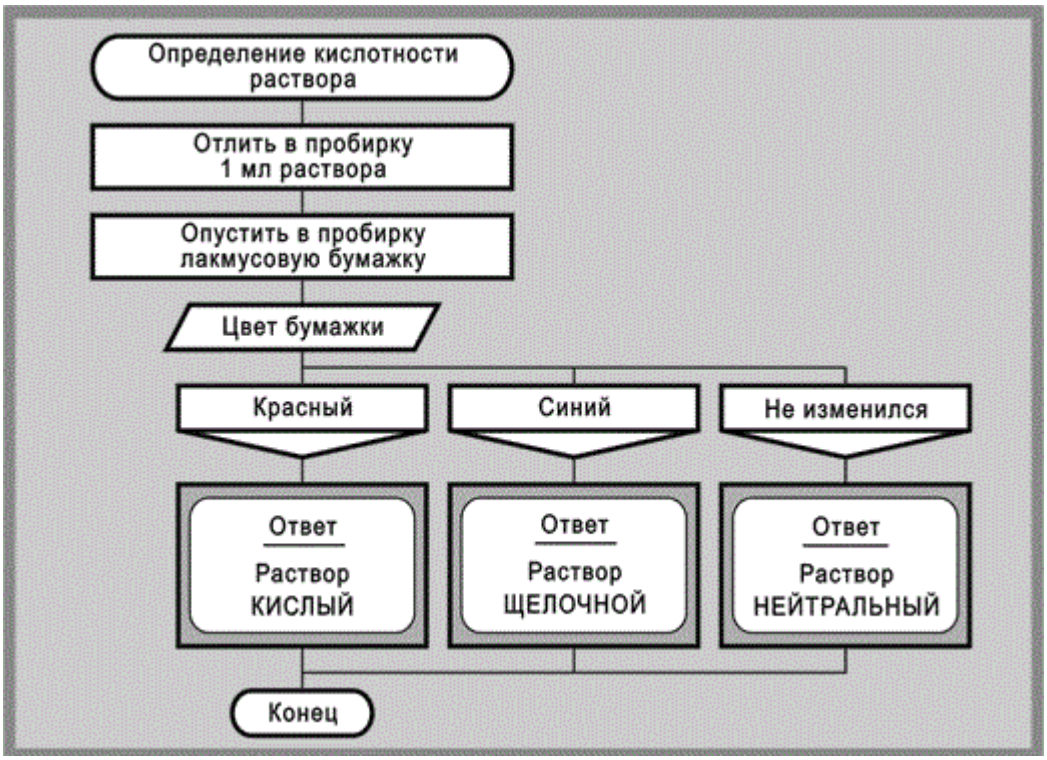

Рис. 1. Пример ДРАКОН-схемы.

Исчисление, разработанное для ДРАКОНа, называется *шампур-методом*. Оно даёт возможность строить «слепыш» алгоритма, последовательно применяя правила построения к базовым заготовкам-шаблонам. В основе метода исчисления лежит небольшое число базовых когнитивно-оринтированных, т.е. упрощающих восприятие, принципов:

- *шампура* — иконы-операторы, следующие в алгоритме *безусловно* друг за другом, т. е. представляющие собой прямую и непосредственную последовательность действий, упорядочиваются по вертикали, так что первая икона всегда лежит сверху, последняя – снизу, и все они лежат на одной оси;
- *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*;
- *вложения* — схема наращивается вводом элементов или групп элементов, построенных из знаков алфавита, в специально указанные точки на уже существующей схеме;
- *силуэта* — принцип разбиения программы на этапы, каждый из которых представляет собой выделенную шампур-ветку, соединенные друг с другом последовательно через особую структуру — петлю силуэта — и вершины-соединители.
- *операций с лианой* — возможность без изменения в прочем структуры программы переноса концов побочных маршрутов (изменение конца блока условного оператора и т.п.).

1.3. Создание ДРАКОН-схем

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

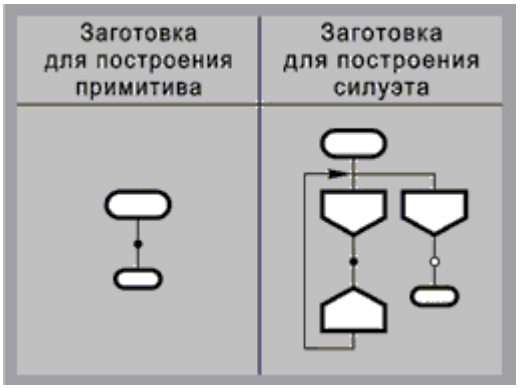

Рис. 2. Заготовки для построения ДРАКОН-схемы.

Первая заготовка – самая простая – может использоваться для создания коротких, одноэтапных программ. Возможность создания заготовки “силует” суть разделение одноэтапной программы, представленной “примитивом” на несколько этапов. Каждый этап представляется отдельной веткой в отдельном шампуре. Название этапа объявляется в заголовке шампура.

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

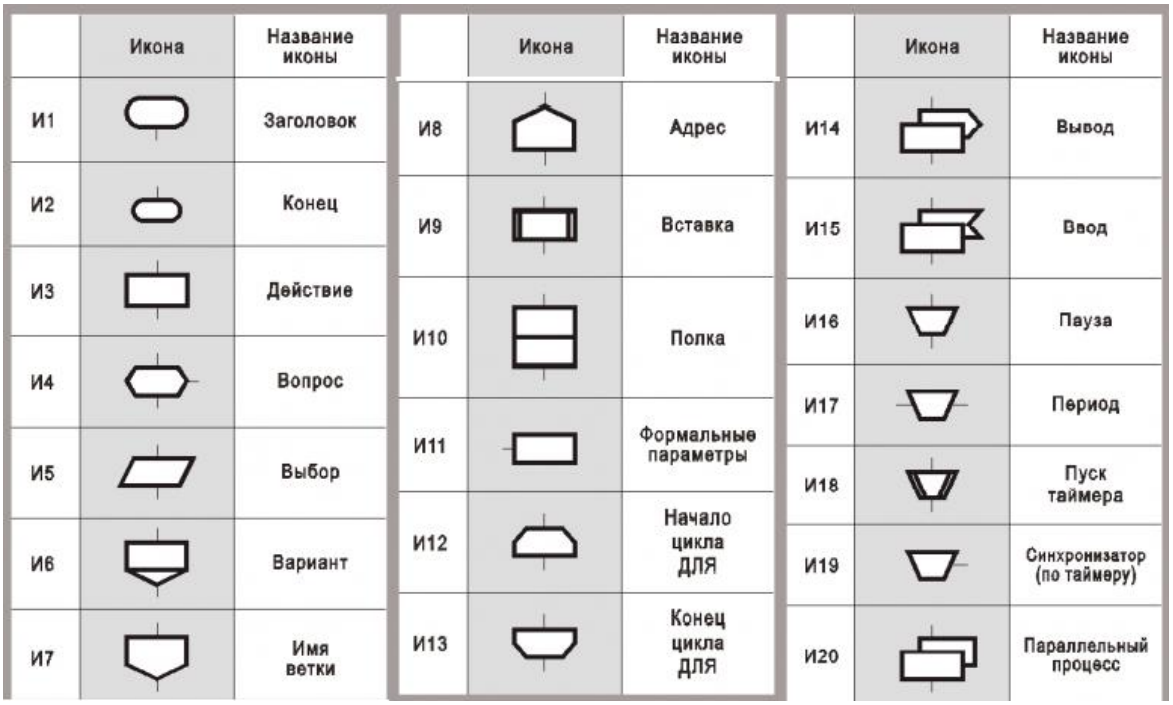

Рис. 3. Базовые иконы языка ДРАКОН.

Атомом называется элемент языка, который за одну транзакцию мы можем добавить в схему. Это может быть одна из базовых икон языка (рис. 3) или один из элементов-шаблонов. Полный список атомов см. рис. 4.

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

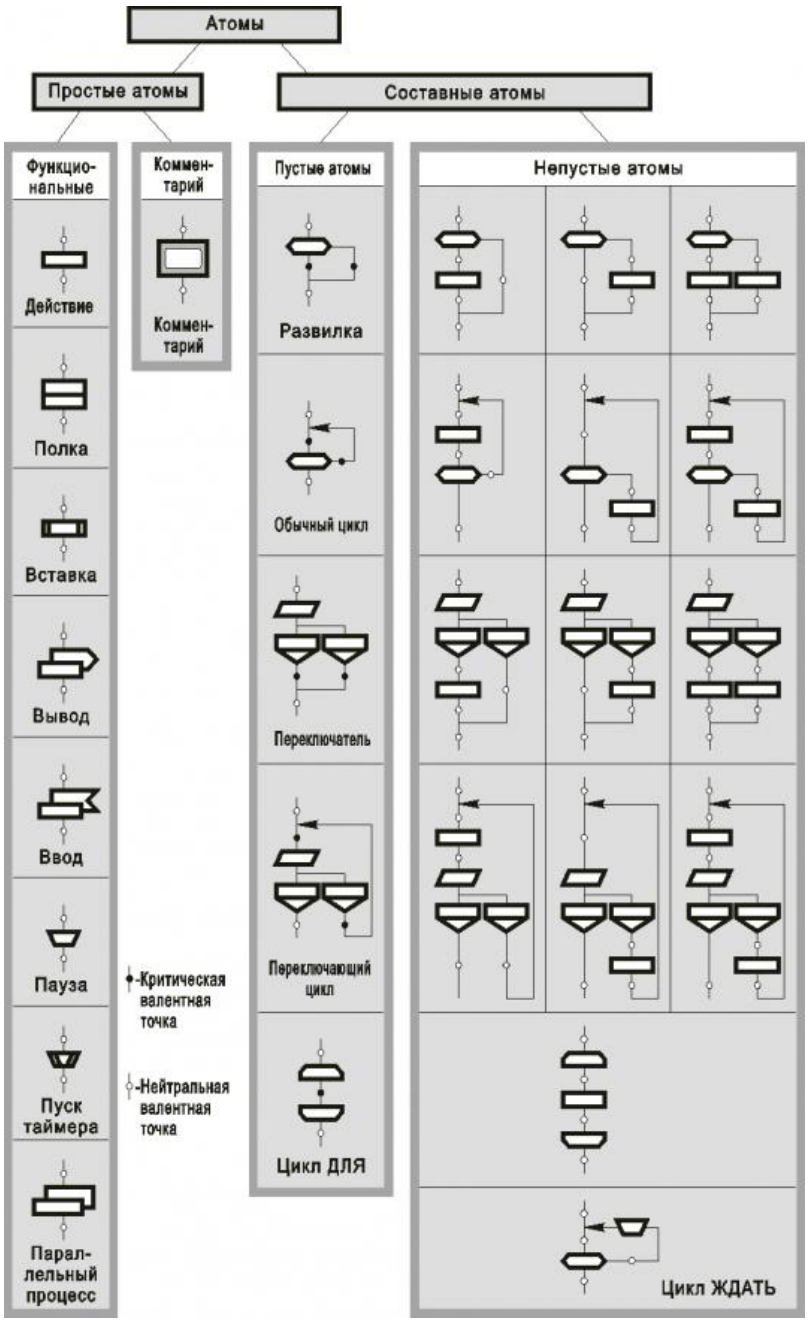

Рис. 4. Атомы языка ДРАКОН.

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

- валентные точки заготовок;
- входы и выходы атомов.

Ввод атома производится так: происходит разрыв соединительной линии в выбранной пользователем валентной точке, после чего в место разрыва вставляется атом, как показано на рис. 5.

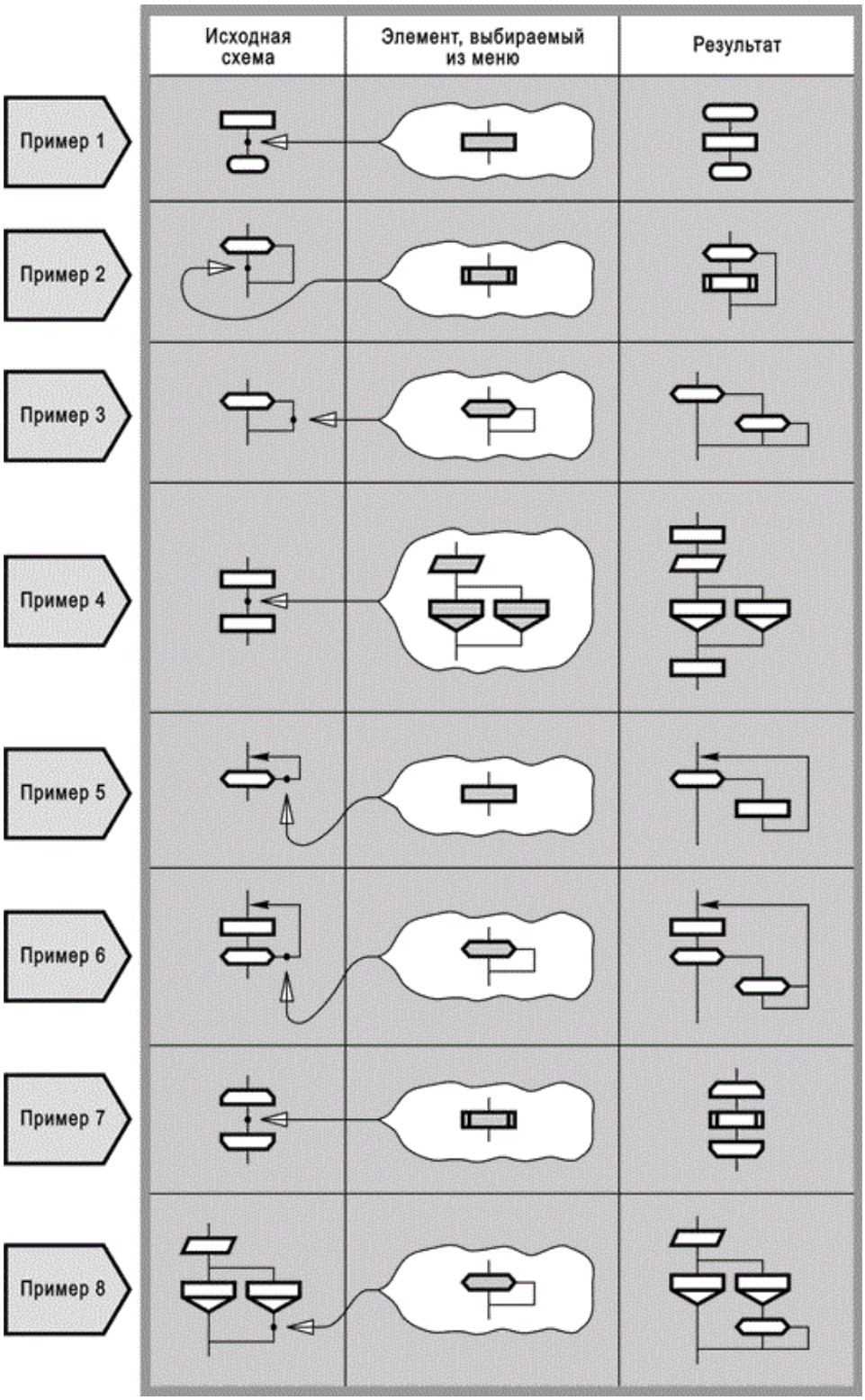

Рис. 5. Примеры операций ввода атома.

Ниже приведены примеры конструирования ДРАКОН-схем на базе заготовок “примитив” и “силует” (рис. 6 и 7).

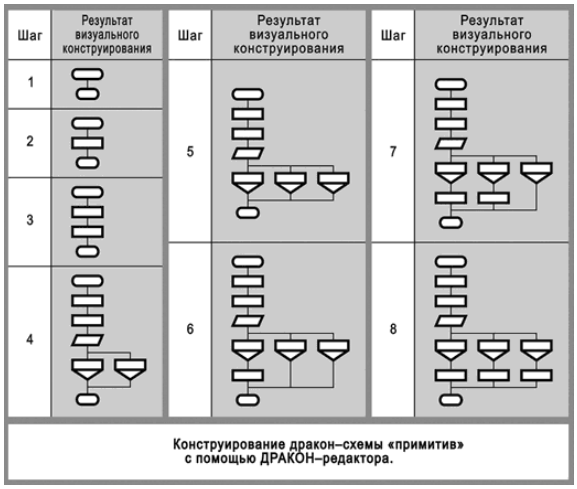

Рис. 6. Конструирование ДРАКОН-схемы на базе заготовки “примитив”.

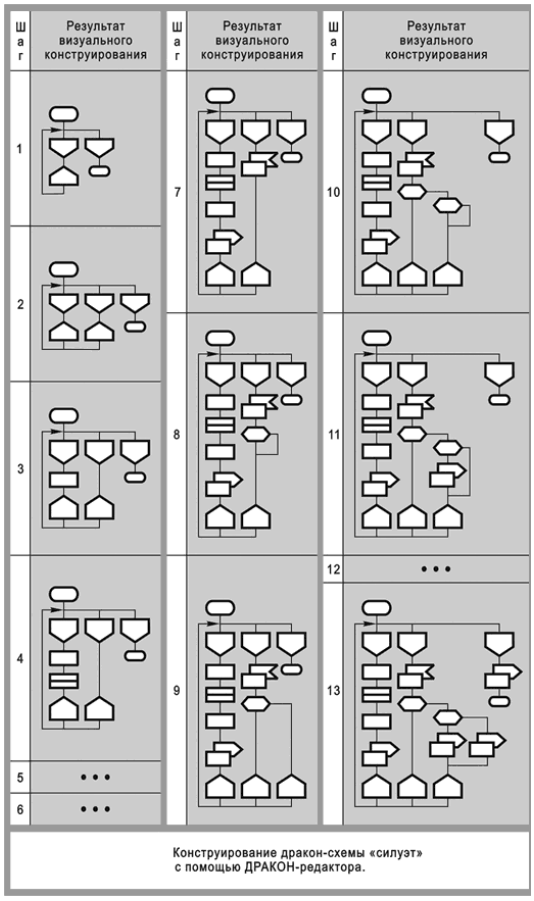

Рис. 7. Конструирование ДРАКОН-схемы на базе заготовки “силуэт”.

2. Конкретизация направлений разработки и используемых средств

Первоочередной задачей моей курсовой работы являлось изучение языка ДРАКОН, как представителя семейства визуальных языков программирования, ориентированного на оптимизацию процесса разработки. В дальнейшем следовало отобрать некоторые отделяемые от контекста ДРАКОНА свойства языка и попробовать сделать некоторое обобщение их на более широкий пласт языков.

2.1. Отбор эвристик

Возникает предположение о том, что введение в произвольный язык возможностей аналогичного ДРАКОН'у способа создания программ, могло бы улучшить его характеристики. Среди всего многообразия облегчающих когнитивные процессы свойств языка дракон были отобраны некоторые, используемые, в процессе создания схем языка. Прослеживая способы создания программ на ДРАКОН'е, следует отметить, что процесс, начинающийся с использования заготовок, продолжается в основном за счет наличия некоторого набора других заготовок и возможности расширять ими уже существующую схему.

Таким образом, появляется решение: из спектра наличествующих эвристик языка выбрать и реализовать следующие две:

- возможность расширять поток управления существующей программы непосредственной в него вставкой допустимого элемента;
- возможность оперирования в процессе создания схемы, помимо элементов атомарных, шаблонами – группами элементов.

2.2. QReal

Для реализации обобщения выбранных эвристик на более широкий класс языков необходима возможность, во избежание “ручной” реализации для каждого языка с прямым перебором некоторого множества языков, использования единообразного, обобщенного описания языков и средства, способного генерировать редакторы языков по эти описаниям.

Хорошим решением, таким образом, было бы расширение арсенала возможностей описания языков некоторым metaCASE-средством, в качестве такого средства была выбрана система QReal - это кроссплатформенный свободно распространяемый под лицензией GNU GPL инструмент с открытым исходным кодом, предназначенный для создания специализированных сред визуального программирования. Её важной особенностью является возможность с помощью метамоделирования быстро и удобно реализовывать новые предметно-ориентированные визуальные языки. Для этого системой предоставляются такие инструменты, как визуальный метаредактор с редактором графического представления элементов модели, возможность автоматической компиляции созданной метамодели языка в модуль для среды QReal, реализующий графический редактор этого языка. Также QReal предоставляет возможности написания генераторов исходного кода по визуальным моделям.

3. Реализация

3.1. Об архитектуре QReal.

QReal – сложная система с многоуровневой архитектурой. Она предоставляет возможность быстрого и удобного создания новых редакторов за счет генерации их по метамоделям языка. 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 (см. рис. 8).

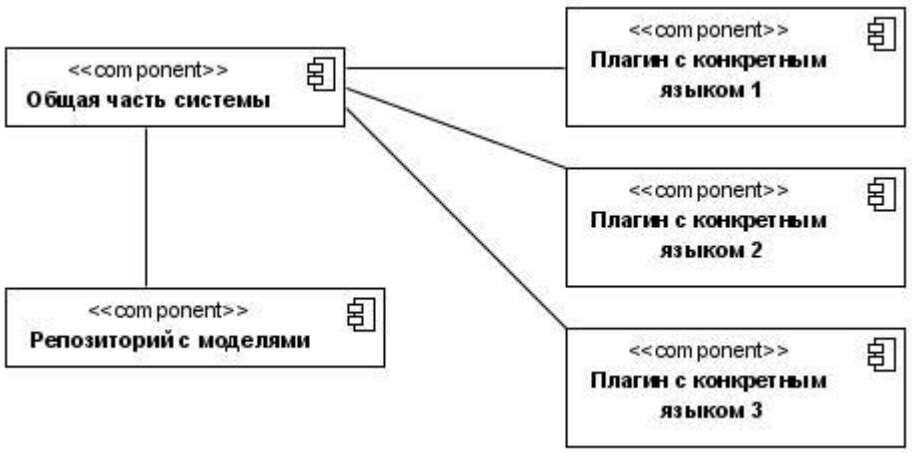

Рис. 8. Представление архитектуры QReal как metaCASE-системы.

Генерация редактора по описанию проходит в несколько этапов, проиллюстрированных на рис. 9.

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 .dll файл (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.

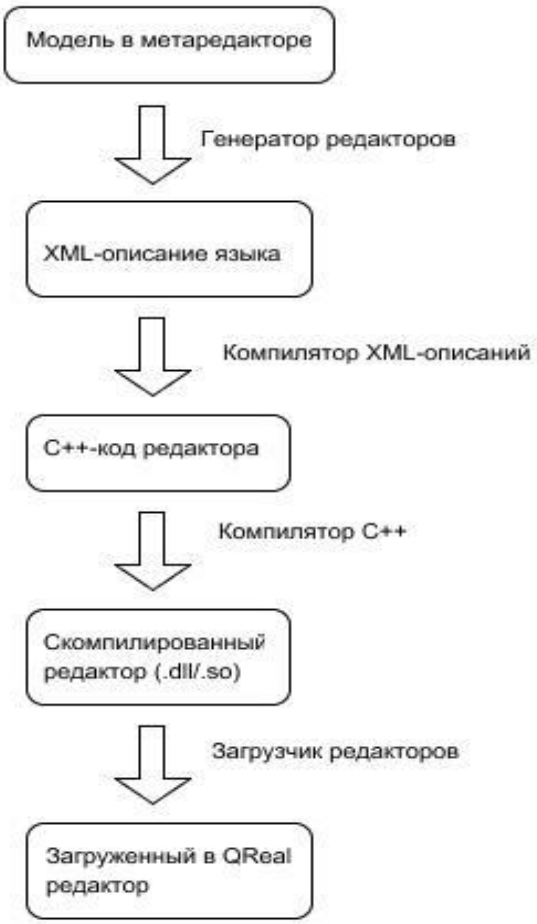

Рис. 9. Этапы генерации нового подключаемого модуля.

3.2. Реализация раздвигаемых ассоциаций

Первой из выбранных эвристик была реализована возможность расширять поток управления существующей программы непосредственной в него вставкой допустимого элемента. В языке ДРАКОН реализация этого свойства осуществляется за счет так называемых *точек входа* на связывающих операторы-вершины линиях потока управления. Точка входа – это точка на ассоциации, на которую можно добавить новую вершину (рис. 10).

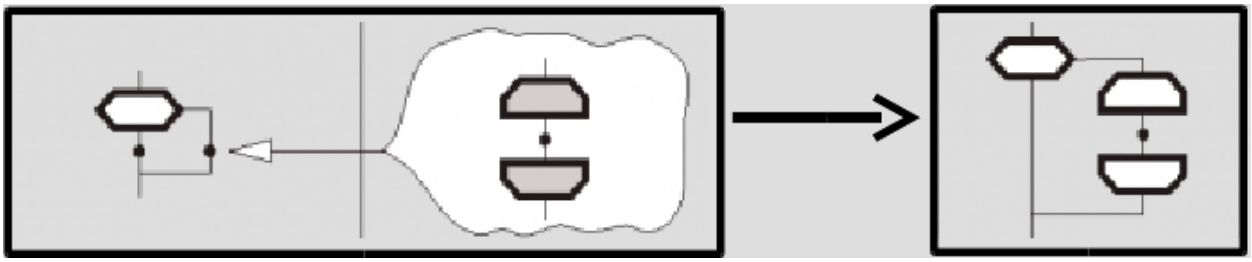

Рис.10. Пример точки ввода.

По аналогии с точками входа было принято решение добавить в редакторы возможность создавать новый вид связей-ассоциаций, обладающий той же функциональностью. Итак, задача: реализовать возможность создания ассоциаций, обладающих свойством “раздвигаемости”.

В соответствии с архитектурой, внесение изменений в спектр возможностей описания метамодели требует расширения функциональности системы на всех уровнях.

1. **Расширен язык описания метамоделей языка**

Языки описываются посредством перечисления свойств составляющих их объектов в xml файле в соответствии с определенными правилами. Совокупность этих правил описания составляет язык описания метамоделей. В соответствии с уже имеющимися правилами, ассоциации и их свойства описываются в теле тега <edge>. Среди свойств связи описываются имя, графическое представление и различные логические свойства, среди которых, например, свойство наследуемости.

Ко множеству возможных графических свойств было добавлено свойство <dividability>, содержащее аргумент логического типа isDividable, при придании которому истинного значения, ассоциация должна считаться раздвигаемой.

```
<logic>
        <dividability isDividable="true" />
        …
</logic>
```

2. **Добавлена обрабатывающая функциональность компилятора xml-моделей**

За компиляцию метамоделей языков отвечает утилита qrxc — QReal Xml Compiler, которая обрабатывает XML-описания редакторов и генерирует код соответствующих подключаемых модулей. В рамках реализации раздвигаемых ассоциаций, были

внесены изменения в класс EdgeType, занимающийся разбором XML-описаний связей и генерацией соответствующего кода на C++.

3. Расширен интерфейс взаимодействия классов-диаграмм с редактором

Взаимодействие элементов языка с их внешним представлением в редакторе обеспечивает интерфейс EditorInterface, расширенным для ассоциаций в соответствии с их новым свойством.

4. Добавлен новый способ взаимодействия диаграмм и ассоциаций на сцене.

- При создании нового узла-оператора, его иконка перетаскивается из палитры и добавляется в заданное место на сцену. Для появления связей со свойством “раздвигаемости” необходимо проверять, не находится ли на позиции, на которую добавляется узел, ассоциация, а если да – не является ли ассоциация раздвигаемой. В случае положительного ответа, необходимо перенаправить существующую ассоциацию ко вновь созданному узлу и создать новую (того же типа, что и уже существующая), направленную от вновь добавленного узла к старому “адресату” раздвигаемой ассоциации. Процесс проиллюстрирован рисунком 11.
- При перемещении уже существующего узла на новую позицию, проверки происходят аналогичным образом.

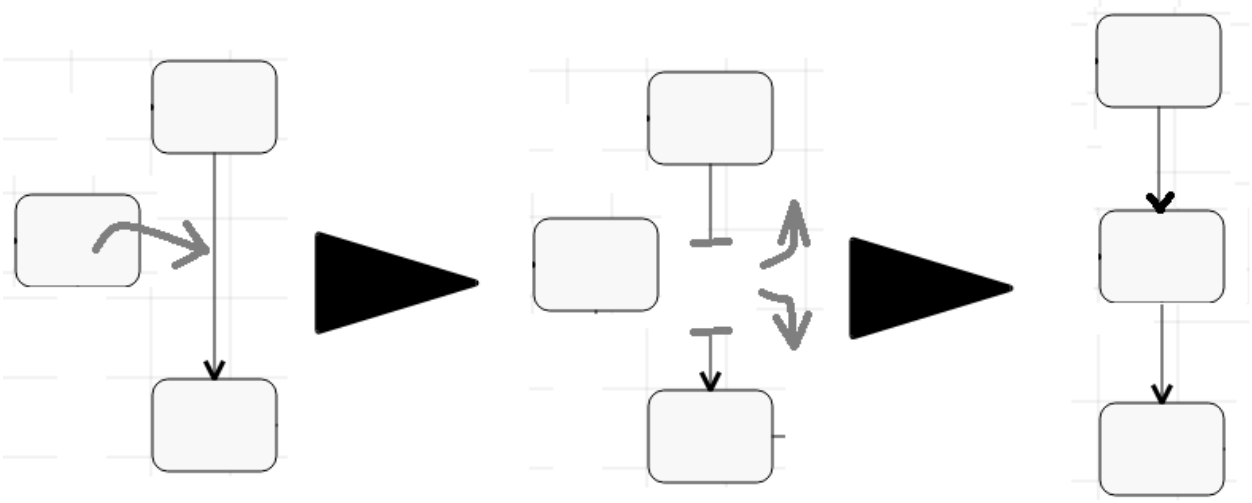

Рис. 11. “Разбиение” связи.

Необходимо также было решить вопрос совместимости новой модели описания редакторов с редакторами уже описанными в рамках старой модели. Было решено сделать любую ассоциацию “нераздвигаемой” по умолчанию, если для не задано свойство раздвигаемости с истинным логическим значением.

3.3. Реализация групп элементов

Второй выбранной для реализации эвристикой была возможность оперирования в процессе создания схемы, помимо элементов атомарных, шаблонами – группами элементов. Необходимость такой возможности очевидна: существуют группы элементов, не использующиеся отдельно друг от друга (как в циклах, рис. 12), а также группы элементов, постоянно использующихся совместно.

Рис. 12. Примеры шаблонов, групп элементов.

К началу работы над курсовой, в QReal существовала возможность генерации палитры, состоящей из одиночных элементов-операторов, описываемых в XML-метамодели языка. Задачей стало реализовать возможность при описании языка неким образом обозначать также логические элементы – являющие собой группы обычных, атомарных элементов, соединенных ассоциациями и определенным образом расположенных – и создавать по ним в редакторе, по аналогии со стандартной палитрой, палитру шаблонов (Рис. 13, палитра справа).

Задача очевидным образом разбивается на две подзадачи:

- генерация палитры шаблонов;

- корректное добавление шаблона на сцену поэлементной раскладкой.

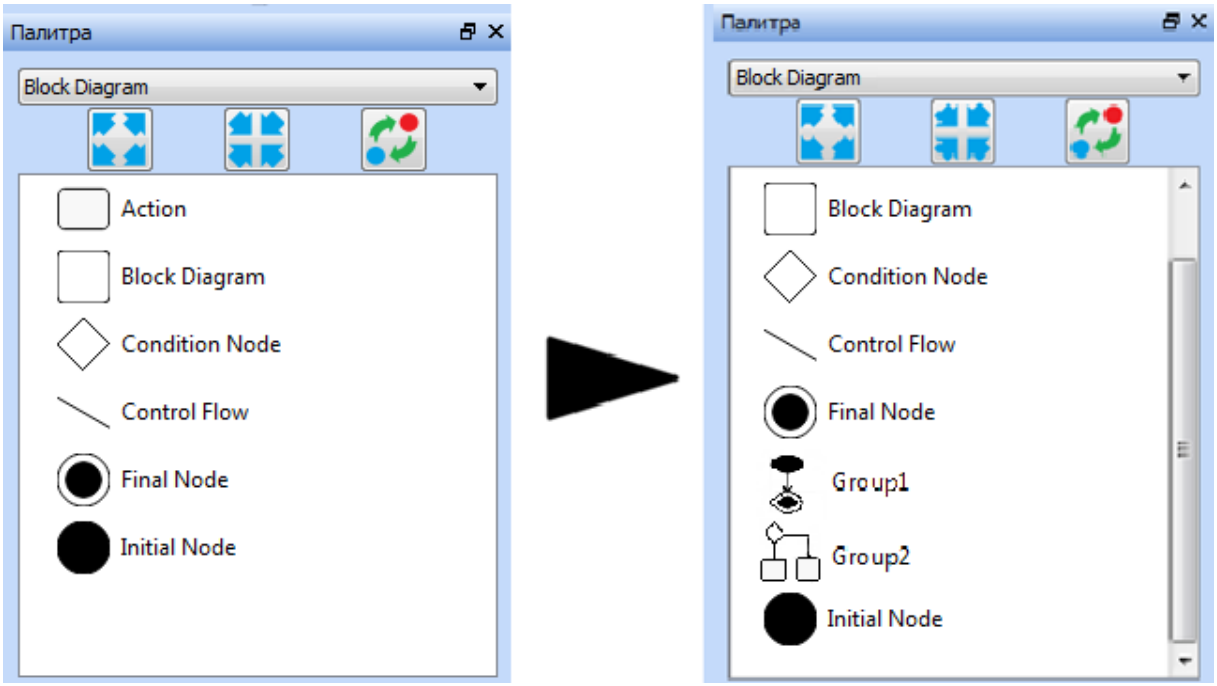

Рис. 13. Палитра, состоящая из атомарных элементов (слева),

палитра, содержащая группы элементов (справа).

По аналогии с добавлением нового свойства для ассоциаций, расширения языка описания редакторов возможностью создания групп элементов также требует расширения функциональности система на всех уровнях архитектуры.

1. Расширен язык описания метамоделей языка

Помимо графических элементов, таких как узлы и ассоциации, XML-метамодель языка содержать описания логических элементов, не имеющих непосредственного графического представления, а относящихся ко всему редактору в целом. К перечню таких логических свойств был добавлен блок свойств <groups>, внутри которого описываются свойства групп элементов, как:

- имя группы;

- список содержащихся в группе элементов с параметрами, такими как тип (тип связи или тип узла), id и относительные координаты (для узлов)и id-узла “отправителя” ассоциации и узла “получателя” (для ассоциации).

<groups >

<group name = “groupName”>

<node type = “type1” id = “1” positionX = “0” positionY = “0” >

<node type = “type2” id = “2” positionX = “150” positionY = “0” >

<edge type = “type3” fromId = “1” toId = “2”>

…

</group>

</groups>

2. Добавлена обрабатывающая функциональность компилятора xml-моделей

В рамках реализации возможности генерации групп элементов, были внесены изменения в классы XMLCompiler, editor и diagram, внесена возможность передачи участка .xml, ответственного за описания групп далее для разбора.

3. Расширен интерфейс взаимодействия классов-диаграмм с редактором

Интерфейс взаимодействия элементов языка с их внешним представлением в редакторе был расширен функцией для передачи участка .xml в другие модули системы.

4. Добавлена обрабатывающей новые логические элементы функциональность

В классе EditorManager была реализована функциональность, позволяющая разбирать участки .xml, ответственные за описания групп, и генерации по ним списка паттернов, каждый из которых отражал бы свойства группы, которую представляет.

5. Добавлена возможность генерации новых элементов в палитре

Конструированием и отображением палитры занимается класс PaletteTree. С помощью уже существующего списка паттернов, была реализована возможность создания палитры, аналогичной уже имеющимся ранее для редакторов, но содержащей шаблоны в качестве элементов.

6. Введена обработка корректного добавления элемента-группы на сцену

Когда элемент уже присутствует в палитре, все, что остается – добавить возможность “бросать” его на сцену и раскладывать на более простые элементы. С этой целью были внесены изменения в метод CreateElement класса editorViewScene, теперь создание атомарных элементов осталось прежним, группа же элементов при добавлении на сцену раскладывается на составляющие ее простые элементы на позиции, рассчитываемые с учетом заданных в описании группы относительных координат элементов, и соединенные между собой, согласно описанию, поданному в XML-модели.

Заключение.

Достигнутые результаты

Итак, в ходе работы над курсовой, были изучены различные вспомогательные средства визуального программирования, способные локально упрощать процесс разработки, процесс конструирования, создания диаграмм, делая его более комфортным. В частности был подробно рассмотрен язык визуального программирования ДРАКОН.

Часть изученного теоретически - а именно такие эвристики языка ДРАКОН, как раздвигаемые ассоциации и возможность автоматической генерации по описаниям шаблонов-групп элементов, были реализованы на практике и апробированы группой разработчиков QReal в частности.

Дальнейшие планы

В дальнейшем планируется продолжать работу в выбранном направлении, а именно – изучать и прочие упрощающие процесс разработки средства визуального программирования. Также планируется реализация некоторых других эвристик языка ДРАКОН, применительно к metaCASE-средсту QReal, а в идеале: добавление в перечень уже существующих редакторов системы редактора языка ДРАКОН, полностью охватывающего весь спектр его характеристик.

Список используемой литературы

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

[2] *Н. В. Видинеев* Природа интеллектуальных способностей человека, М.: Мысль, 1989.

[3] *В. Д. Паронджанов* Перспективы информационных технологий и повышение продуктивности интеллектуального труда // НТИ. Сер. 1. 1993. № 5.

[4] А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов и др. Архитектура среды визуального моделирования QReal. // Системное программирование. Вып. 4. СПб.: Изд-во СПбГУ. 2009, С. 171-196

[5] *А.С. Кузенкова Ю.В. Литвинов, Т.А. Брыксин*, Метамоделирование: современный подход к созданию средств визуального проектирования. // Материалы второй научно-технической конференции молодых специалистов «Старт в будущее», посвященной 50-летию полета Ю.А. Гагарина в космос. СПб. 2011. С. 228-231

[6] *Т.А.Брыксин Ю.В.Литвинов*, Технология визуального предметно-ориентированного проектирования и разработки ПО QReal // Материалы международной конференции "Информационные технологии в образовании и науке", Самара, 2011

[7] *Joel Spolsky* User Interface Design for Programmers, Apress; 1 edition, 2001

[8] qreal *(QReal research group)* https://github.com/qreal (21.05.2012)