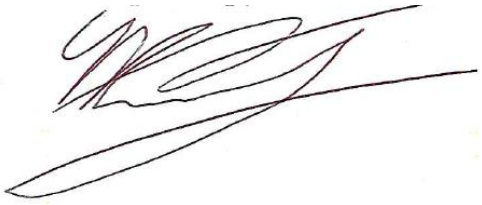
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

На правах рукописи

Степанов Павел Алексеевич



Модели, алгоритмы и программные средства описания визуальных языков на основе вычислительных моделей

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация
на соискание ученой степени кандидата технических наук

Научный руководитель:
Д.т.н., профессор Охтилев М.Ю.

Санкт-Петербург - 2019

ОГЛАВЛЕНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ 4
ВВЕДЕНИЕ 5
1. Модели определения визуальных языков 16
1.1. Задачи, стоящие перед визуальными языками при представлении результатов обработки данных при оценивании ТС РКТ 16
1.2. Системный анализ существующих математических моделей визуальных языков 19
1.3. Формальная постановка задачи исследования 26
Выводы по разделу 1. 30
2. Модель визуального языка для задач РКТ 33
2.1. Вычислительная модель диаграммы 33
2.2. Алгоритм расчета вычислительной модели визуального языка 37
2.3. Визуальный синтаксис диаграммы 47
2.4. Логические объекты диаграммы 51
2.5. Интерфейс взаимодействия с пользователем 55
2.6. Циклические вычисления 57
2.7. Описание визуального языка 59
2.8. Трансляция диаграмм в вычислительной модели 60
Выводы по разделу 2. 64
3. Прикладная вычислительная модель визуального языка и инструментальное средство поддержки прикладных визуальных языков 67
3.1. Типы данных и операции над ними 67
3.2. Визуальные примитивы 73
3.3. Математическое ядро редактора диаграмм 75
3.4. Преобразования вычислительной модели визуального языка и интерфейс редактора диаграмм 77
3.5. Транслятор диаграмм 81
Выводы по разделу 3. 82
4. Результаты применения модели в задачах оценивания состояний объектов РКТ 84
4.1. Решение задачи поддержки общеупотребительных диаграмм в условиях импортозамещения в РКТ на примере ER-диаграмм 85
4.1.1. Обоснование выбора ER-модели и постановка задачи 85
4.1.2. Реализация визуальной части объекта «сущность» 87
4.1.3. Реализация визуальной части объекта «связь» 96

4.1.4. Невизуальные атрибуты и сценарий трансляции 99
4.2. Решение задачи графического представления телеметрической информации при контроле технического состояния объектов ракетно-космической техники 103
4.2.1. Обоснование выбора моделируемой подсистемы и постановка задачи 103
4.2.2. Реализация мнемосхемы тракта наддува топливных баков 104
4.2.3. Отображение телеметрической информации на мнемосхеме 114
4.3. Технико - экономическая эффективность 117
Выводы по разделу 4. 120
Заключение 122
Список литературы 126
ПРИЛОЖЕНИЕ П1. Обоснование преимуществ визуальных языков перед текстовыми 156
ПРИЛОЖЕНИЕ П2. Обзор моделей представления визуальных языков 160
ПРИЛОЖЕНИЕ П3. XML Схема дескриптора визуального языка 196
ПРИЛОЖЕНИЕ П4. XML Схема дескриптора данных диаграммы 206
ПРИЛОЖЕНИЕ П5. Описание ER-Диаграммы средствами вычислительной модели визуального языка (фрагмент) 209
ПРИЛОЖЕНИЕ П6. Внешний вид и результаты трансляции ER- диаграммы 231

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ER	Entity-Relationship (Сущность-связь)
IT	Информационные технологии
MVC	Model-View-Controller
UML	Unified Modelling Language
АГС	Абстрактный граф синтаксиса
АГТ	Алгебраические графовые трансформации
ГГ	Графовая грамматика
ИИ	Искусственный интеллект
ПГО	Пространственный граф отношений
ПО	Программное обеспечение
РКТ	Ракетно-космическая техника
РПО	Разработка программного обеспечения
СПО	Специальное программное обеспечение
ТС	Техническое состояние

ВВЕДЕНИЕ

Актуальность. В современной IT-индустрии повсеместно применяются разнообразного рода машинные языки, предназначенные для восприятия компьютером, а не человеком. Наиболее известным классом таких языков являются текстовые языки программирования, для которых существует хорошо развитая теория, позволяющая создавать трансляторы в машинные команды [1,2]. В последнее время для решения разнообразных задач достаточно широко применяются также и визуальные языки, например, такие, как язык моделирования UML [3], языки на основе диаграмм «сущность-связь» [55], языки моделирования Simulink [25] и LabView [48], а также другие визуальные языки.

Считается, что визуальные языки намного нагляднее и удобнее для восприятия человеком, нежели текстовые. В частности, недавнее исследование с применением магнитно-резонансной томографии показывает, что постановка задач с использованием графических схем приводит к значительному ускорению в получении ответа [232].

Ввиду вышесказанного, визуальные языки внедряются повсеместно как в задачах разработки и проектирования информационных систем различных назначений, так и в задачах визуализации данных. Интерес к визуальным языкам крайне высок. Достаточно сказать, что существуют ежегодные международные конференции, посвященные визуальным языкам и визуальному программированию, например, IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) или The International Conference on the Theory and Application of Diagrams.

Применительно к средствам ракетно-космической техники (РКТ), таким как ракеты космического назначения, разгонные блоки, космические аппараты, элементы наземной космической инфраструктуры, визуальные языки используются не только при проектировании прикладного

программного обеспечения, но также и при выполнении процедур по оценке их технического состояния (ТС) [56]. При этом они применяются на всех этапах жизненного цикла РКТ. Используемые в настоящее время средства далеки от совершенства и не обеспечивают в полной мере эффективного решения названных задач. Достаточно актуальной также является задача разработки и сопровождения специального программного обеспечения (СПО), на долю которого приходится большая часть возникающих при проведении оценивания ТС ошибок, неточностей и пр., что значительно снижает результативность рассматриваемых процессов, часто приводит к различным по масштабам авариям и даже катастрофам и в целом не обеспечивает достаточного уровня эффективности применения средств РКТ [20]. Кроме того, в свете принятого правительством Российской Федерации курса на импортозамещение, в том числе программного обеспечения, крайне важно создавать отечественные аналоги средств визуального проектирования и разработки, используемых в РКТ.

Необходимо отметить, что круг задач, решаемых с помощью визуальных языков, не ограничивается задачами проектирования ПО. Визуальные языки используются в музыке, медицине, генетике, образовании, в том числе для обучения детей (в том числе и через Интернет), в изобразительном искусстве, а также моделировании трехмерных сцен, поиске, распознавании и обработке изображений, поиске информации, интеллектуальном анализе данных (data mining), интерпретации математических выражений и теорий, телекоммуникациях, описаниях бизнес-процессов и алгоритмов, проектировании устройств и программного обеспечения, программировании устройств на низком уровне, цифровых схем, ведении архивов мультимедиа, веб-дизайне, управлении роботами, архитектуре, банковской деятельности, обработке текстов на естественных языках, управлении безопасностью и даже для создания интерфейсов типа “командная строка”. Можно отметить языки для программирования веб-приложений и SOA, а также большое количество графических подходов к

работе с базами данных, равно как и разнообразных расширений языка UML. Все эти приложения требуют разработки соответствующих средств редактирования и преобразования диаграмм. Все это говорит о том, что имеется крайне актуальная задача описания синтаксиса и семантики произвольных визуальных языков.

Ввиду наличия большого количества хорошо известных подходов к работе с текстовыми языками наиболее очевидным способом определения визуального языка является адаптация формализмов текстовых языков. И действительно, доминирующим подходом в данное время является подход на основе так называемых графовых грамматик [214, 215], то есть формализм, по аналогии с текстовыми языками[1,2], использующий правила свертки графов в некоторый начальный граф либо синтез диаграммы из этого начального графа. Однако, данный подход приводит к специфическим проблемам, которые отсутствуют у текстовых языков, в частности:

- в отличие от текста, у графического представления (изображения, диаграммы) обычно нет начала и конца, точка начала анализа не определена;
- в отличие от текста, который в большинстве случаев может быть представлен в виде последовательности символов на бесконечной ленте, диаграммы и изображения всегда не менее, чем двумерны;
- элементы изображения могут содержать в себе другие изображения, синтаксически или семантически связанные с основным; возможно наличие сложной иерархии изображений.
- изображения могут быть динамическими.

Несмотря на перечисленные проблемы, подход на основе графовых грамматик широко применяется; существует адаптированная классификация визуальных языков по Хомскому [94,234] и адаптированные к визуальным языкам алгоритмы трансляции, в том числе алгоритмы LALR [208], Эрлея [105, 122] и Кока-Янгера-Касами (CYK) [218]. Таким образом, графовые

грамматики, как минимум, применимы к любым контекстно-свободным визуальным языкам [102].

В то время как основным преимуществом формализма графовых грамматик является большая выразительная мощность и проработанные алгоритмы, взятые из теории текстовых языков, основным их недостатком является высокая сложность. Действительно, даже для текстовых языков, многие популярные компиляторы не используют формализмы грамматик по причине слишком высокой сложности. Для визуальных языков сложность построения грамматики еще выше из-за перечисленных ранее специфичных проблем, поэтому другим популярным подходом к задаче построения средства поддержки визуального языка является описание его через объекты предопределенного поведения. Вероятно, наиболее известным средством такого рода является Microsoft Visio [26], крайне простое для пользователя, однако и поддерживающее только очень ограниченные синтаксис и семантику.

Таким образом, налицо **противоречивая ситуация** – либо используемые формализмы универсальны, но очень сложны, что ограничивает их применимость для решения прикладных задач, либо просты, но обладают недостаточными выразительными средствами. Разрешение этого противоречия является сущностью решаемой **прикладной задачи -** создания моделей и методов определения визуального языка и поддерживающего их программного средства, обладающего простотой для пользователя и, одновременно, выразительными свойствами, сравнимыми с формализмом графовых грамматик, а также проверка применимости разработанных моделей, методов и программных средств к решению задач контроля ТС РКТ в условиях импортозамещения.

Для решения данной задачи в данной работе предлагается оригинальный подход к описанию визуальных языков, который совмещал бы в себе универсальность и гибкость графовых грамматик с простотой таких систем, как Visio [26].

Степень разработанности темы. Современное состояние предметной области исследования визуальных средств можно охарактеризовать как некоторое насыщение, в котором новые формализмы графовых грамматик практически не создаются, зато исследуются свойства и применимость существующих. Фундаментальный вклад в предметную область внесли выдающиеся отечественные и зарубежные ученые, в частности, Ахо А. и Ульман Дж.[1,2], Вирт Н.[4], Гладкий А.В. [5], Городецкий В.И.[7], Ершов А.П.[9,10], Котов В.Е. [16,17,18], Ляпунов А.А. [19], Майерс Б.[171,189], Непейвода Н.Н.[22], Терехов А.Н. [46,47], Тыугу Э.Х.[49,50,51,52], Эхриг Х. [121], Хомский Н.[53,94], Янов Ю.И.[57,58], и другие, развившие базовые элементы теории автоматов, теории алгоритмов, теории искусственного интеллекта, математической логики, теории формальных языков и грамматик.

Нужды индустрии, появление и внедрение в практику парадигмы “цифрового общества” привели к развитию разнообразных прикладных теорий, решающих задачи создания универсальных визуальных редакторов и формализмов, стреди которых, в частности, можно отметить

- схемы представления алгоритмов анализа информации на основе G-сетей, научная школа проф. Охтилева М.Ю. и проф. Соколова Б.В., СПИИРАН, Россия [23,24];
- многоагентные, иерархические, P2P модели, научная школа Городецкого В.И., СПИИРАН, Россия [7];
- формализм алгоритмических сетей, научная школа проф. Иванищева В.В., проф. Морозова В.П. и проф. Марлея В.Е; СПИИРАН, Россия [11,12];
- технологию REAL и генератор графических редакторов QReal, научная школа проф. Терехова А.Н., Санкт-Петербургский Государственный Университет [46,47];
- формализм расширенных позиционных грамматик, научная школа проф. Дж. Костаглиолы, университет Салерно, Италия [101,102,103,104,105,106];

- формализм грамматик упорядочивания изображений, разработанный в Университете Брауна, Род Айленд, США [143,144,145,146];
- формализм графических функциональных грамматик К. Коджимы и Б. Майерса, Университет Джона Монаша, Австралия [171];
- различные решения производителей средств CASE и RAD, предназначенные для расширения возможностей их графических редакторов пользователем, в частности, Eclipse GMF и Microsoft DSL Tool;
- и многие другие.

Исходя из всего вышесказанного, **целью диссертационной работы** является сокращение сроков и трудоемкости проектирования программных средств наглядного отображения и оценивания технического состояния сложных технических объектов с использованием визуальных языковых моделей предметной области.

Объектом исследования в данной работе являются визуальные языки, используемые при проектировании прикладного ПО наглядного отображения и оценивания ТС СТО, **предметом исследований** являются модели, методы и алгоритмы, а также разработанные на их основе программные средства, предназначенные для визуализации и оценивания ТС СТО и поддержки жизненного цикла прикладного ПО.

На основании результатов анализа существующих технологий представления и применения визуальных методов в диссертационной работе необходимо решить следующие **основные задачи**:

1) провести анализ проблемных аспектов существующих моделей и методов формализации визуальных языков;
2) разработать модели и алгоритмы, обеспечивающие поддержку синтаксиса и семантики визуальных языков, доступные широкому кругу лиц;
3) разработать универсальный комплекс программ, реализующий отмеченные алгоритмы и модели;
4) исследовать применимость моделей, алгоритмов и разработанного комплекса программ средств разработки программного обеспечения (ПО)

для задач проектирования и визуального контроля технического состояния СТО.

Методология и методы исследований базируются на использовании элементов теории искусственного интеллекта, теории множеств, теории графов, теории категорий, теории алгебраических графовых трансформаций и теории формальных языков, а также методов математической логики и теории программирования.

Научная новизна диссертационной работы состоит в следующем:

1) разработан подход к созданию редакторов диаграмм, используемых в задачах разработки и контроля технического состояния сложных технических объектов, основанный на описании визуального языка пользователем, отличающийся от аналогичных подходов тем, что объединяет описание предметной области и описание визуализации в единую математическую модель;
2) разработаны модели и алгоритмы, обеспечивающие поддержку синтаксиса и семантики визуальных языков, доступные широкому кругу лиц, отличающиеся тем, что за основу описания синтаксиса визуального языка взята вычислительная модель Тыугу;
3) разработан программный комплекс - универсальный редактор диаграмм, для работы с вышеописанными моделями и алгоритмами, отличающийся использованием для описания синтаксиса визуального языка вычислительных моделей;
4) для языка диаграмм “сущность-связь” разработано описание синтаксиса, отличающееся тем, что за основу описания взята вычислительная модель, и продемонстрирована возможность создания диаграмм и их трансляции в сценарии создания схем баз данных, а также для мнемосхемы тракта наддува топливных баков ракеты “Союз-2”, разработано описание, отличающееся тем, что за основу описания взята вычислительная модель, и продемонстрирована возможность использования полученной мнемосхемы при контроле технического состояния системы.

Методология и методы исследования. При выполнении диссертационных исследований использовались элементы теории искусственного интеллекта, теории множеств, теории графов, теории категорий, теории алгебраических графовых трансформаций и теории формальных языков, а также математической логики.

Теоретическая значимость полученных в диссертационной работе результатов заключается в получении единой математиченской модели описания функционирования и графического представления сложного технического объекта, при этом имеется возможность определения правил визуализации ограниченным математическим аппаратом, а также возможность преобразования в различные артефакты.

Практическая значимость полученных результатов состоит в возможности внедрения в модель СТО правил ее визуализации, что приводит к возможности ее использования не профессиональными программистами, а экспертами в области контроля качества и эксплуатации СТО. За счет этого стоимость разработки ПО автоматизированных систем оценки качества СТО снижается на 20-30%, а время разработки на 10-15%.

В целом, все полученные в работе результаты направлены на формирование единого системного подхода к разработке систем визуализации информации, характерных для телеметрических систем передачи, приема, преобразования и обработки информации.

Основные положения, выносимые на защиту:

1) Расширенная вычислительная модель визуального языка для поддержки человеко-машинных интерфейсов.
2) Алгоритмы расчета вычислительной модели визуального языка и приведения ее к стационарному виду.
3) Метод трансляции диаграмм, заданных в вычислительной модели визуального языка, в программы на других языках.
4) Метод построения визуальных интерфейсов диаграмм, заданных в вычислительной модели визуального языка, позволяющий, в частности,

построение диалогов с пользователем и сопряжение с телеметрической информацией.

Реализация. Основные теоретические положения и практические результаты работы были использованы при выполнении работ по гранту РФФИ 00-07-90344 “Разработка метамоделей, методов, инструментальных средств и технологии конвертации проектов информационных систем, созданных в соответствии с различными методологиями в различных CASE-системах”, при проектировании ПО автоматизированной системы управления подготовки к пуску ракеты-носителя Союз-2 в АО “РКЦ Прогресс”, АО “СКБ Орион”, а также в учебном процессе в Санкт-Петербургском Государственном университете аэрокосмического приборостроения, при чтении курсов по теории искусственного интеллекта и функциональному и логическому программированию.

Степень достоверности и апробация результатов работы. Достоверность сформулированных научных положений, основных выводов и результатов диссертации обеспечивается за счет анализа состояния исследований в данной области, согласованности теоретических выводов с результатами экспериментальной проверки разработанных алгоритмов в ходе испытаний и штатной эксплуатации семейства ракет-носителей типа «Союз-2». Основные результаты диссертационной работы докладывались и обсуждались:

1) на шестой международная научно-техническая конференция студентов и аспирантов “Радиотехника, электроника и энергетика” (Москва, МЭИ. 1999) [27];
2) на 8й международной студенческой школе-семинаре “Новые информационные технологии” (Москва, МГИЭМ. 2000) [28];
3) на 3ей научной сессии аспирантов ГУАП (Санкт-Петербург, ГУАП, 2000) [29];
4) на пятой Санкт-Петербургской Ассамблее молодых учёных и специалистов (Санкт-Петербург, СПбГУ, 2000) [30];

5) на седьмой международной научно-технической конференции студентов и аспирантов “Радиотехника, электроника и энергетика” (Москва, МЭИ, 2000). [31];
6) на 19й научной сессии ГУАП (Санкт-Петербург, ГУАП, 2016) [32].
7) на 20й научной сессии ГУАП (Санкт-Петербург, ГУАП, 2017) [37,38];
8) на 19й международной конференции “Проблемы управления и моделирования в сложных системах” ИПУСС РАН (Самара, ИПУСС РАН, 2017) [36];
9) на Восьмой Всероссийской научно-практический конференции по имитационному моделированию и его применению в науке и технике “Имитационное моделирование. Теория и практика” (ИММОД-2017) [39].

Публикации. Основные результаты и положения диссертационной работы опубликованы в 18 печатных работах. Из них опубликовано 6 статей в рецензируемых научных журналах из Перечня ВАК [33,34,35,40,41,42] и 9 докладов в материалах научно-технических конференций различного уровня, в том числе международных и всероссийских. Кроме того, получено 3 свидетельства о государственной регистрации программ для ЭВМ [43,44,45].

Структура и объем работы

Работа состоит из введения, четырех разделов, заключения, библиографического списка из 274 наименований и шести приложений. Объем основной части работы составляет 110 страниц машинописного текста.

Во введении обоснована актуальность проводимых исследований, определены цели работы, основные задачи исследования, отражена научная новизна и практическая значимость работы, сформулированы результаты, выносимые на защиту.

В первой главе дается обоснование преимуществ визуальных языков над текстовыми, анализируются и систематизируются существующие способы формальных описаний визуальных языков, проводится формальная постановка задач исследования.

Во второй главе вводится вычислительная модель визуального языка, показывается ее применимость к задачам визуализации диаграмм, вводятся алгоритмы для вычисления модели и приведения модели в стационарный вид, демонстрируется метод описания визуального языка на основе модели.

В третьей главе дается описание прикладной реализации вычислительной модели визуального языка и программного средства для ее поддержки.

В четвертой главе приводится описание прикладных результатов, а именно разработанного редактора ER-диаграмм, который может быть использован для проектирования баз данных, и средства визуализации телеметрической информации подсистемы наддува топливных баков ракеты-носителя Союз-2.

1. Модели определения визуальных языков

В данном разделе рассматриваются основные задачи, стоящие перед визуальными языками, при оценивании технического состояния ракетно-космической техники; анализируются и систематизируются существующие способы формальных описаний визуальных языков; демонстрируются недостатки существующих моделей и проводится формальная постановка задач исследования.

1.1. Задачи, стоящие перед визуальными языками при представлении результатов обработки данных при оценивании ТС РКТ

Задача оценивания технического состояния ракетно-космической техники в общем случае заключается в сравнении измеренных и эталонных характеристик оцениваемого объекта. При этом оцениваемый объект может быть как математической моделью узла или агрегата (на начальном этапе разработки), так и непосредственно этим узлом или агрегатом (на позднем этапе разработки). Такая оценка производится в большинстве случаев автоматически на основании имеющейся базы знаний и расчетов (рис. 1.1).

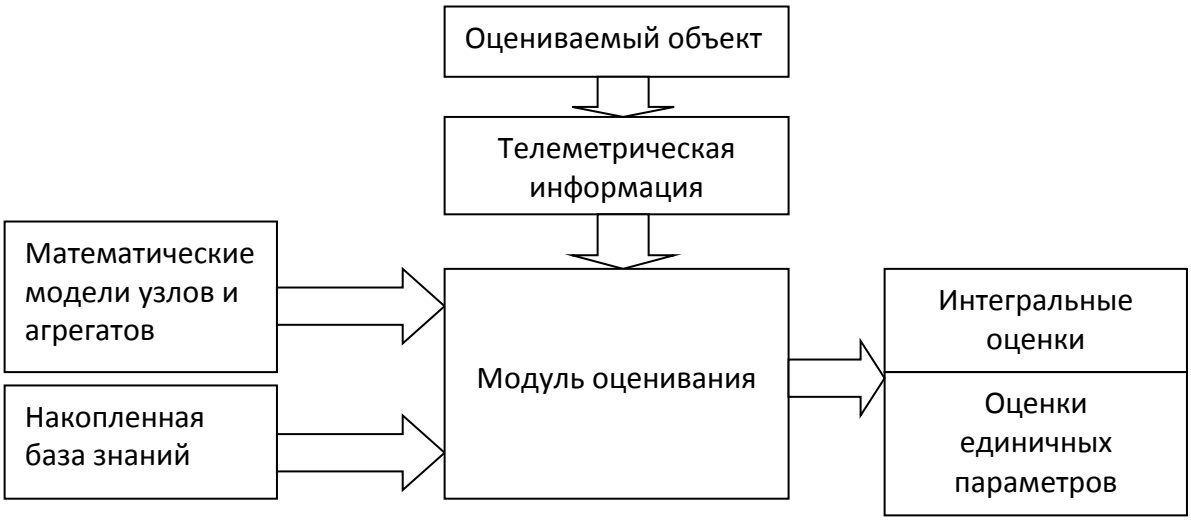


Рис.1.1. Схема процесса оценивания ТС объекта РКТ

В качестве средства представления пользователю оценки технического состояния, как правило, используются мнемосхемы. Мнемосхемы представляют собой графические схематические изображения узлов и агрегатов, на которых тем или иным образом (обычно цветом или текстом) представляется текущая оценка технического состояния оцениваемого оъекта. Таким образом, частью задачи оценивания ТС является разработка соответствующей оцениваемому объекту мнемосхемы для выполнения визуального контроля технического состояния различных подсистем и объектов РКТ [23].

Мнемосхемы представляют собой схематические изображения деталей и узлов космических средств, ракеты-носителя и пр, на которых непосредственно отображается информация об их состоянии. Подобный подход имеет достаточно глубокие исторические корни, в частности, на заре развития кибернетики человеко-машинные интерфейсы зачастую выполняли в виде аппаратных мнемосхем (индикаторных ламп, изображений логических связей и устройств на фронтальных панелях и пр.). С развитием индустрии программного обеспечения стало возможным использовать мнемосхемы произвольной сложности на экране компьютера, что повсеместно используется при человеко-машинном взаимодействии при контроле технического состояния объектов космической техники [14,24].

Другой важной областью применения визуальных языков в задачах РКТ является разработка программного обеспечения, в рамках которой широко используются диаграммы сущность-связь, схемы алгоритма, UML. В настоящее время программные средства, поддерживающие указанные визуальные нотации, в основном производятся за рубежом (Power Designer компании Sybase, Rational Rose компании IBM и др.), поэтому целесообразным является их импортозамещение. Таким образом, возникают две связанные задачи – поддержка стандартных диаграмм в целях импортозамещения и поддержка настраиваемых пользователем произвольных диаграмм для тестирования и улучшения качества СПО.

Помимо преимуществ высокой наглядности результата и возможности быстрого вмешательства при некорректной работе оцениваемого объекта системы графической визуализации также обладают и недостатками, главным из которых является необходимость традиционного программирования при их разработке или изменении. Эта проблема может быть в какой-то степени устранена путем создания библиотек стандартных правил визуализации, например, в случае ракетно-космической техники в такую библиотеку могут входить топливные и воздушные баки, трубопроводы, вентили и другие элементы РКТ. Однако, появление нового правила визуализации или уточнение уже существующего немедленно ставит задачу переработки программного кода, отвечающего за его реализацию. Учитывая тот факт, что контролем технического состояния зачастую занимаются инженеры, имеющие другую специализацию, нежели программирование, вышеописанный недостаток приводит к необходимости найма дополнительного персонала либо нахождения субподрядчиков, готовых поддерживать соответствующее программное средство. Таким образом, возникает задача ухода от традиционного программирования при визуализации технического состояния оцениваемого объекта.

Решением вышеописанной задачи может являться объединение модели функционирования объекта и модели его визуального представления. Описав правила визуализации объекта тем же математическим аппаратом, что и его модель, можно добиться существенного упрощения модификации правил визуализации оцениваемого объекта и избавиться от необходимости наличия штата программистов. Более того, на основе того же самого подхода могут быть реализованы многие визуальные инструменты, используемые при проектировании программных систем. Таким образом, возникает задача выбора математического аппарата, позволяющего описать как математические модели эталонного функционирования оцениваемых объектов, так и их визуальное представление.

1.2. Системный анализ существующих математических моделей визуальных языков

Необходимо отметить, что круг задач, решаемых с помощью визуальных языков, не ограничивается задачами проектирования ПО. Они находят свое применение в самых различных сферах деятельности, таких как распознавание изображений [61] и их обработка [70], обработка мультимедиа данных [112, 113, 114, 115, 116], в том числе, в реальном времени [98], описание интерфейсов пользователя в различных предметных областях, таких как медицина [66, 68, 163, 199, 200, 220], химия [180], финансы [67] и работа с компьютером [134, 210], программирование микропроцессоров [64], описание электрических цепей [242, 243] и трехмерных сцен [248], телекоммуникации [255], интеллектуальный анализ данных [69, 92], поиск информации [206], в том числе в сети Интернет [196, 241], управление компьютерной безопасностью [88] и процессами сместной деятельности [104, 142, 252], управление роботами [110, 216, 217], веб-дизайн [157], описание систем документооборота [166], архитектурный дизайн [224], ообработка естественных языков [230]. В виду того, что в целом визуальные языки считаются более простыми к пониманию людьми (см. приложение П1) они нашли широкое применение в обучении [78, 84, 117, 148, 158, 169, 202, 207, 245, 256], в том числе при описании алгоритмов [80, 81, 122, 138, 175, 193], а также при разработке учебных курсов [65]. Все эти применения требуют разработки соответствующих средств редактирования и преобразования диаграмм. Все это говорит о том, что имеется крайне актуальная задача описания синтаксиса и семантики произвольных визуальных языков, частью которой и является задача, поставленная в предыдущем разделе.

Вышеуказанную задачу было бы проще всего решить на основе существующих математических аппаратов, описывающих визуализацию объектов. Широкое распространение визуальных языков обуславливает крайнюю востребованность разработки универсального инструментального средства их поддержки. Для решения сформулированной задачи необходимо

построить формальную модель визуального языка на основе уже имеющегося формализма, используемого для представления информации о штатном функционировании оцениваемого объекта. Как показано в приложении П2, для создания универсального формализма, описывающего визуальные языки, предпринималось большое количество попыток. Прежде всего, было введено содержательное определение диаграммы [86].

Определение 1.1. Диаграммой называется структурированная система связанных графических объектов.

Как следует из этого определения, диаграмма состоит из сущностей двух видов – графических объектов и отношений между ними, что может быть математически выражено следующим образом:

Определение 1.2. Диаграммой называется пара множеств

$$D = \{I, R\}, \tag{1.1}$$

где I – множество графических объектов;

R – множество отношений между этими изображениями.

Множество графических объектов I может содержать в себе произвольные изображения, в то время как для отношений R основные категории определены, а именно, отношения позиционного синтаксиса, размерного синтаксиса, временного синтаксиса и синтаксиса правил [235]. Дополнительно вводятся понятия синтаксиса взаимодействия и структуры, предназначенные для описания взаимодействия диаграммы с пользователем через редактор. Более подробно вопрос рассмотрен в приложении П2. Таким образом, множество отношений R из (1.1) представляет собой

$$R = \{R_p, R_d, R_t, R_r, R_i, R_s\}, \quad (1.2)$$

где R_p – отношения позиционного синтаксиса;

R_d- отношения размерного синтаксиса;

R_t – отношения временного синтаксиса;

R_r - отношения синтаксиса правил;

R_i - отношения синтаксиса взаимодействия;

R_s - отношения синтаксиса структуры.

Далее необходимо определить понятие языка, на котором описываются диаграммы (визуального языка). Базовым содержательным определением визуального языка, по аналогии с текстовым языком, является следующее:

Определение 1.3 Визуальным языком называется множество всех диаграмм, допустимых для этого языка.

Так же, как и для текстовых языков, для абсолютного большинства практически используемых языков перечислить все диаграммы невозможно. Кроме того, специфические аспекты визуальных языков, такие как наличие позиционного синтаксиса, делают определение языка через перечисление диаграмм практически невозможным. В связи с этим были предложены разнообразные модели визуальных языков, наиболее существенные из которых будут рассмотрены ниже.

Существует пять основных подходов к формализации: подходы на основе графовых грамматик (ГГ), теории категорий, λ-исчисления, искусственного интеллекта и объектов с шаблонами поведения.

А. Графовые грамматики представляют собой обобщение развитого формализма грамматик текстовых языков на визуальные языки путем представления диаграммы в виде графа и определения алфавита (в котором

терминальные символы представляют вершины, а нетерминальные символы – фрагменты графа) и продукционных правил, трансформирующих граф. Это наиболее крупный класс моделей, подробно рассмотренный в приложении П2.

Определение 1.4 Графовой грамматикой VL называется множество

$$VL = \{VT, VN, VP, \varepsilon\} \quad (1.3)$$

где VT – множество визуальных терминалов или визуальный алфавит (множество всех элементарных изображений);

VN – множество всех нетерминалов;

VP – множество продукционных правил (правил преобразования графов, составленных из VT и VN);

ε - начальный граф языка.

Как показано в приложении П2, при рассмотрении свойств графовых грамматик возникает ряд специфических, по сравнению с текстовыми грамматиками, проблем, а именно:

- у диаграмм обычно отсутствует начало и конец, нет единой точки для начала анализа;
- обычно диаграмма многомерна, в связи с этим алгоритмы имеют большую вычислительную сложность;
- расположение элементов диаграммы в пространстве или по отношению друг к другу может быть частью его семантики;
- обычно большая часть диаграммы предназначена для создания дружественного пользователю интерфейса и не обладает релевантной для задачи семантикой, поэтому многие формализмы ГГ разделяют диаграмму на абстрактный граф синтаксиса (АГС), содержащий в себе часть диаграммы, обладающую семантикой, и пространственный граф

отношений (ПГО), содержащий визуальные элементы, предназначенные для человека, с последующим игнорированием последнего.

Все это существенно усложняет формализм ГГ по сравнению с текстовыми языками. Как показано в приложении П2, ряд исследователей вводят существенные допущения и ограничения на ГГ для того, чтобы сложность моделей не оказалась чрезмерной. Тем не менее, подходы с использованием ГГ имеют в своей основе хорошо проработанные формализмы и для них адаптированы основные алгоритмы.

Б. Подход на основе теории категорий является обобщением графовых грамматик с помощью более общей теории. Основным представителем являются теория алгебраических графовых трансформаций (АГТ), существующая в двух вариантах – на основе одинарного кодекартова квадрата и двойного кодекартова квадрата [121]. Принципиальная разница между двумя подходами заключается в том, что в первом случае применение одного и того же морфизма может моделировать как удаление, так и добавление вершин в граф, в то время как во втором случае для каждой операции требуется отдельное применение морфизма, то есть большинство преобразований требует применения двух морфизмов.

Подход на основе теории категорий является наиболее общим и универсальным, пригодным практически для любых языков. Компенсацией за универсальность является его крайняя сложность – подход оперирует графами, в которых возможны дуги не только между вершинами, но и между другими дугами. Преобразование конкретных визуальных языков в подобные графы является отдельной проблемой.

В. Подход на основе λ-исчисления трактует диаграмму (1.1) через конструирование графа из начального пустого графа путем применения к нему узлов и ребер [122,123,124,125]. Наиболее проработанным представителем

является индуктивный граф, первоначально предназначенный для описания синтаксиса визуального языка VEX (см. приложение П2). Формализм обладает серьезными преимуществами с точки зрения применения теории вычислимости и формальных доказательств свойств визуального языка. Сдругой стороны, в виду того, что графовые трансформации предполагают только добавление новых узлов, выразительные свойства данного подхода весьма ограничены.

Г. Подходы на основе искусственного интеллекта (ИИ) предполагают обучение графических редакторов на основе некоторых шаблонов поведения, выбранных программистом. Графические редакторы при этом ориентируются на разнообразные методы искусственного интеллекта – нечеткую логику, генетические алгоритмы, семантические сети и пр. в попытке сгененировать такой редактор, который будет удовлетворять поставленной задаче. Подход является преимущественно интерфейсным и при этом требует от программиста понимания работы нижележащих алгоритмов для правильного выбора шаблонов для обучения системы [181,185,249,250,270].

Д. Подход на основе объектов с шаблонным поведением используется в простых редакторах схем, таких, как Visio[26]. При этом подходе создается богатая библиотека разнообразных объектов, обладающих предопределенным поведением, но при этом отсутствует или крайне ограничен синтаксис, связанный с отношениями между объектами. Сильной стороной подхода является удобная возможность создания иллюстративного материала, в то время как главный недостаток – отсутствие проверки синтаксической корректности, ограниченное взаимодействие с пользователем и невозможность трансляции.

Рассмотренные выше формализмы (А-Д) сведены в таблицу 1.1. Для каждого из них указаны достоинства и недостатки применительно к задаче

визуализации измерительной информации при оценивании ТС объектов оценки ТС РКТ.

Таблица 1.1. Математические модели представления визуальных языков

Основа подхода	Преимущества	Недостатки
1	2	3
А. ГГ	Группа хорошо проработанных формализмов, использующих развитый инструментарий текстовых языков. Формализмы поддерживают вплоть до контекстнозависимых языков, к ним адаптированы основные алгоритмы.	Высокая сложность, необходимость идти на ограничения выразительных средств ради упрощения модели. Игнорирование конкретного визуального изображения в пользу абстрактного графа синтаксиса.
Б. АГТ	Универсальный формализм, пригодный для описания любых визуальных языков	Требует разработки взаимного преобразования прикладных языков в графы, которыми оперирует формализм
В. λ-исчисление	Формализм хорошо работает при доказательстве формальных свойств модели. Вычисление λ-выражения, описывающего граф, является естественной реализацией трансляции.	Класс поддерживаемых языков существенно ограничен, так как каждая трансформация связана только с внесением новых узлов и дуг, но не с их удалением.
Г. ИИ	Формализм хорошо работает на простых визуальных языках, не слишком отличающихся от имеющихся в базе знаний стереотипов.	Формализм требует хорошего знания алгоритма ИИ для выбора оптимальных шаблонов для обучения. Формализм в наибольшей степени предназначен для реализации интерфейсов, семантика языка не рассматривается.

Д. Шаблоны объектов	Богатые визуальные библиотеки предопределенных объектов, гибкость при создании диаграмм	Слабая или отсутствующая поддержка синтаксиса и семантики, фокус исключительно на визуальное представление.

Поддержка стандартных диаграмм является задачей, которая может быть решена с помощью однократной настройки. Таким образом, для ее решения можно использовать любую и вышеперечисленных хорошо формализованных моделей (ГГ, АГТ). С другой стороны, требования к диаграммным языкам, применяемым для оценки ТС объектов РКТ, существенно отличаются. При выполнении моделирования параметров РКТ может потребоваться заранее неизвестный набор визуальных элементов и диаграмм, при этом нецелесообразно ожидать, что инженеры, выполняющие моделирование, владеют теорией формальных языков. Из таблицы 1.1 следует вывод, что существующие формализмы либо обладают широкими выразительными свойствами, но крайне сложны в прикладном использовании, либо очень просты, но не обладают нужной гибкостью.

Таким образосм, проведенный системный анализ демонстрирует существенное противоречие между существующими математическими аппаратами для описания визуальных языков и задачей описания модели функционирования оцениваемого объекта, а также заявленной целью обеспечить простоту в унифицированность описания правил визуализации оцениваемых объектов. Все это приводит к выводу о необходимости разработки промежуточного формализма, обладающего большей простотой, по сравнению с графовыми грамматиками, и одновременно поддерживающего сложный синтаксис.

1.3. Формальная постановка задачи исследования

Естественным процессом рисования большинства диаграмм пользователем является замыкание различных контуров и сопоставление

точек. Например, в случае диаграммы Сущность-Связь пользователь рисует две сущности, после чего соединяет их связью. После этого полученная система при перемещении одной из сущностей обязана хранить информацию о том, что она соединена связью с другой сущностью. Видится естественным, что система, управляющая диаграммами, должна оперировать теми же самыми понятиями – например, «начало связи находится на правом ребре сущности 1», «конец связи находится на левом ребре сущности 2» и т.п., в то время как в настоящее время основным средством определения визуальных языков являются продукционные правила. Как было показано ранее, существующие формализмы либо оперируют значительно более сложными понятиями, либо в недостаточной степени поддерживают синтаксис.

Пользователь, создающий элементы визуального языка обычно оперирует понятиями вида “объект представляет собой круг, внутри которого расположен текст” или “объект представляет собой прямоугольник, над которым расположено его название”. Таким образом, было бы удобно описывать визуальный язык в дружественных пользователю правилах вида “объект состоит из круга и текста, причем для центра круга, радиуса круга и размеров текста устанавливаются определенные математические соотношения”. Подобный описатель диаграммы на визуальном языке является семантической сетью, концептами которой являются параметры графических и неграфических элементов, а также математических отношений между концептами; языком будет являться набор подсетей и правил их вставки в основную сеть.

Для представления знаний о штатном функционировании оцениваемого объекта часто используется вычислительная модель Тыугу [49,50,51,52]. Эта модель представляет собой формализованную семантическую сеть, в узлах которой находятся параметры системы и формулы, а связи представляют входные параметры и выходные значения формул. Модель в первую очередь ориентирована на вычислительные задачи,

однако, будучи дополнена визуальной составляющей, может быть применена к описанию визуального языка. В этом случае эксперт, модифицирующий программное средство, поддерживающее визуализацию технического состояния оцениваемого объекта, может формулировать изменения в математической модели поведения объекта и в средстваах его визуализации пользуясь одной и той же моделью. Таким образом, возникают подзадачи расширения модели визуальной составляющей, реализации транслятора входного языка модели и реализации средства визуализации на основе расширенной модели.

Вычислительная модель Тыугу в своем оригинальном определении является нестационарной, что означает, что вычислительные процессы могут идти разными путями и потенциально давать разные результаты. Тыугу считал это свойство модели ее преимуществом и либо рссматривать в качестве решения любой произвольный путь, либо множество всех путей.

К сожалению, применительно к любым системам взимодейтствия с человеком, подобный недетерминированный подход применяться не может – оператор программного средства не ожидает, что при взаимодействии с диаграммой он может получить множество диаграмм, из которых будет необходимо выбрать правильную. Таким образом, возникает еще одна подзадача – приведения модифицированной вычислительной модели к стационарному виду (в своей визуальной части).

Наконец, еще одним важным вопросом является получение новых артефактов на основе взаимодействия со средством визуализации. Например, используя средства визуального проектирования программного обеспечения оператор ожидает получить на выходе некоторые шаблоны кода, при работе со схемами баз данных на выходе должен быть сценарий загрузки базы на языке SQL и так далее. Отсюда возникает еще одна подзадача – трансляции вычислительной модели. Необходимо обеспечить, чтобы состояние модели, полученное в результате взаимодействия с пользователем, можно было

обработать и преобразовать (транслировать) в соответствующие предметной области артефакты.

Подытоживая вышесказанное, создание или модификация нового визуального средства контроля может быть представлена следующей схемой:

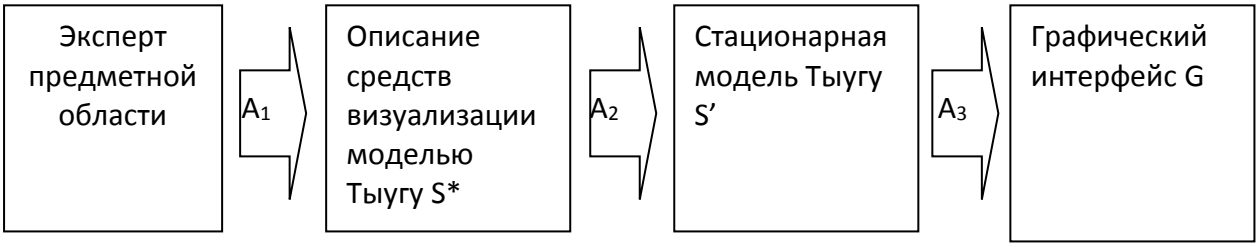


Рис. 1.2. Схема решения задачи.

Где A_1 – средства представления правил визуализации,

S* - модель Тыугу, адаптированная для визуальных языков,

A_2 – алгоритм преобразования модели в стационарную модель

S’ – стационарная модель Тыугу, адаптированная для визуальных языков

A_3 – алгоритм преобразования модели в графический интерфейс (рассчета модели)

G –графический интерфейс, визуальное представление оцениваемого объекта.

В ряде случаев может требоваться получение артефактов на основе взаимодействия с графическим интерфейсом G, в такой ситуации неолбходимо добавить еще два шага:

A_4 – алгоритм трансляции модели

D – артефакт трансляции.

Таким образом, для решения задачи объединения знаний о штатном поведении оценниваемого объекта и правил его визуализации необходимо последовательно описать A_1, S*, A_2, S’, A_3, G, A_4 и D.

В разделе 2 будет дано описание формализма и определены A_1, S*, A_2, S’ и A_3. В разделе 3 будет дано описание инструментального средства, поддерживающего формализм и реализующего G, A_4 и D. В разделе 4 будет

показано, как формализм применим к языкам общего назначения и задачам РКТ.

Выводы по разделу 1.

1. Средства визуализации результатов обработки информации широко используются в двух областях: проектирование и разработка программного обеспечения и визуализация результатов процессов технического состояния узлов и агрегатов, в том числе как при анализе телеметрируемой информации, так и при имитационном моделировании.
2. Визуальные языки являются обобщением текстовых языков на диаграммы. Визуальными называют языки, в которых данные могут представляться в более чем одномерном виде. В таких языках элементами синтаксисам могут являться направление, цвет и звук, кроме того, они могут быть динамическими. Визуальные языки обладают преимуществом над текстовыми, так как в силу психологических или функциональных (ослабленное зрение) причин диаграмма может лучше восприниматься читателем, нежели текст.
3. Существует большое количество разнообразных методов формализации визуальных языков. Наиболее распространенный из них – сведение формализма визуальных языков к формализму текстовых языков, то есть графовые грамматики. Этот метод не свободен от существенных недостатков, приводящих к существенному усложнению всех основных алгоритмов. Поэтому для визуальных языков только очень ограниченное количество входит в классы LL (1) и LR (1), имеющие наибольшее значение для трансляции, большинство визуальных языков требует для трансляции алгоритмы Эрли или Кока-Янгера-Касами.
4. Одним из наиболее проработанных формализмов является алгебраическая трансформация графов, построенная на теории категорий. Этот формализм пригоден для описания любых визуальных

языков, но оперирует специальными графовыми структурами, для которых необходимо отдельно разрабатывать методы взаимного преобразования в целевую диаграмму.

5. Существуют и другие формализмы, не связанные с графовыми грамматиками, в частности, описание визуальных языков через лямбда-исчисление (как применение некоторых операций конструирования к исходному пустому графу) а также через средства искусственного интеллекта.
6. Для хорошо формализованных подходов характерно игнорирование визуальной составляющей диаграммы путем ее деления на абстрактный граф синтаксиса и пространственный граф отношений. Для слабо формализованных подходов, наоборот, основной является визуальная составляющая. Кроме того, строгие математические формализмы обычно весьма сложны и недоступны человеку без специального образования, в то время как круг лиц, которые могут являться авторами графических формализмов, не ограничен.
7. Применительно к задачам РКТ нецелесообразно ожидать, что инженеры, проводящие оценку ТС РКТ, владеют теорией формальных языков. Это приводит к необходимости разработки нового формализма, который позволил бы объединить знания о штатном функционировании оцениваемого объекта и о правилах его визуального представления, взяв за основу формализм, уже широко применяющийся в предметной области.
8. Вычислительные модели Тыугу представляют собой формализованную семантическую сеть, в концептах которой находятся значения и операции над ними, вычисляемые по определенным правилам. Визуальные языки могут быть описаны через набор значений координат и атрибутов объектов, их графических свойств и соотношений между этими значениями. Формализовав эти объекты и отношения, а также

правила их обработки, можно получить адаптированную вычислительную модель для диаграмм.

9. Схема решения задачи состоит из последовательного определения и описания следующих элементов: средств представления правил визуализации A_1, модели Тыугу, адаптированной для визуальных языков S*, алгоритма преобразования модели в стационарную модель A_2, стационарной модели Тыугу, адаптированной для визуальных языков S’, алгоритма преобразования модели в графический интерфейс (рассчета модели) A_3, графического интерфейса G, алгоритма трансляции модели A_4 и артефакта трансляции D.

2. Модель визуального языка для задач РКТ

2.1. Вычислительная модель диаграммы

Как было указано в разделе 1, поставлена задача создания метода описания визуальных языков через набор значений координат и атрибутов объектов, их графических свойств и соотношений между этими значениями. Для ее решения и создания адаптированной вычислительной модели для диаграмм необходимо формализовать эти объекты и отношения, а также правила их обработки, можно будет получить. В рамках этой цели, прежде всего, необходимо ввести ряд определений:

Определение 2.1. Атрибутом называется пара

$$a = \{n, v\},$$

где

n - имя атрибута или $\emptyset$, если оно не задано;

v- значение атрибута или $\emptyset$, если значение не определено.

Определение 2.2. Визуальным примитивом называется

$$v = \{A, f_g(A)\},$$

где

A – множество атрибутов примитива;

$f_g(A)$- графическая функция (функция, преобразующая значения атрибутов в изображение).

Определение 2.3. Диаграммой называется множество

$$D = \{A, G, V, R\}, \quad (2.1)$$

где

A - множество атрибутов, такое, что $A = A_n \cup A_v$, причем A_n - множество невизуальных атрибутов и A_v - множество визуальных атрибутов, то есть существует однозначное отображение из A_vв V;

G- алгебраическая система, заданная на множестве атрибутов;

V – множество визуальных элементов диаграммы, $v \in V \vdash v \in V_T$ (символом $\vdash$ обозначается логическое следование, т.е. в данном примере из принадлежности v множеству V следует также его принадлежность V_T), где V_T – предопределенное множество известных визуальных элементов, для которого верно, что $\forall a \in v, v \in V \vdash a \in A_v$, $\forall a \in A_v \vdash a \in v \in V$, т.е. множество A_v состоит из тех и только тех атрибутов, которые принадлежат визуальным элементам из V;

R – множество правил вида $\mathrm{cond}(c_i): a_i = f_j(A);\ a_i \in A$. f_j задано в рамках алгебраической системы G, где c_i – условие, при истинности которого выполняется присвоение $a_i = f_j(A)$ (для краткости в случае, если условие всегда истинно, правило записывается как $a_i = f_j(A)$).

Пример 2.1. Рассмотрим следующую ситуацию. Пусть имеются два объекта, соединенных связью. Предположим, что каждый объект имеет свойства X и Y, указывающие координаты его левого верхнего угла и свойства ширина и высота. Отношение пусть имеет свойства (X_1,Y_1), указывающее на координаты его начала, и (X_2,Y_2), указывающие на координаты конца. Допустим, диаграмма выглядит так: (рис. 2.1).

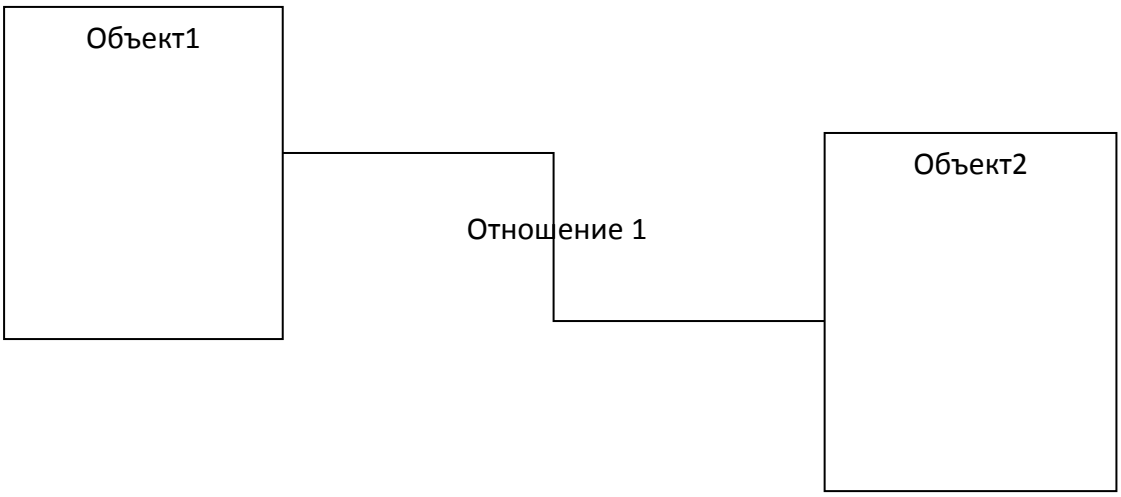


Рис. 2.1. Пример простой диаграммы

Диаграмма на рис .2.1 построена из трех элементов, два из которых принадлежат к типу объекта, а один – к типу отношение (рис. 2.2).

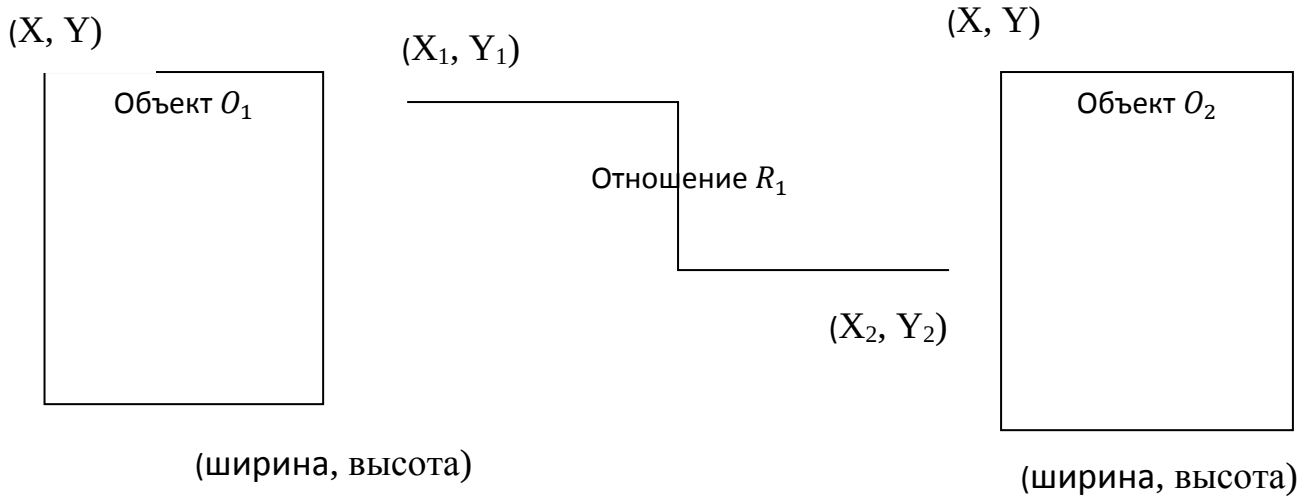


Рис. 2.2 Объекты простой диаграммы

Как видно на рис. 2.2, элемент изображения, представляющий “объект”, содержит четыре атрибута – координаты левого верхнего угла, ширину и высоту. Элемент изображения, представляющий “отношение”, также имеет четыре атрибута – координаты начала и координаты конца.

Для простоты примера примем существенное допущение, что в визуальном словаре имеются графические элементы, эквивалентные сущности и отношению. Назовем их “прямоугольник” и “соединительная_линия”. Очевидно, эти элементы могут быть собраны из более простых элементов – линий, но это существенно увеличит систему соотношений и лишит пример 2.1 простоты и наглядности.

Таким образом, мы можем составить следующую систему соотношений:

$$R_1.X_1 = O_1.X + O_1.\text{ширина} \quad (2.2)$$

$$R_1.Y_1 = O_1.Y + \frac{O_1.\text{высота}}{2} \quad (2.3)$$

$$R_1.X_2 = O_2.X \quad (2.4)$$

$$R_1.Y_2 = O_2.Y + \frac{O_2.\text{высота}}{2} \quad (2.5)$$

При этом диаграмма из примера 2.1 описывается следующим соотношением:

$$D = \{A, G, V, R\}, \quad (2.6)$$

где

$\mathrm{A} = \{\{\mathrm{O}_1.\mathrm{X},\ \text{значение}\},\ \{\mathrm{O}_1.\mathrm{Y},\ \text{значение}\},\ \{\mathrm{O}_1.\text{ширина},\ \text{значение}\},\ \{\mathrm{O}_1.\text{высота},\ \text{значение}\},\ \{\mathrm{O}_2.\mathrm{X},\ \text{значение}\},\ \{\mathrm{O}_2.\mathrm{Y},\ \text{значение}\},\ \{\mathrm{O}_2.\text{ширина},\ \text{значение}\},\ \{\mathrm{O}_2.\text{высота},\ \text{значение}\},\ \{\mathrm{R}_1.\mathrm{X}_1,\ \text{значение}\},\ \{\mathrm{R}_1.\mathrm{X}_2,\ \text{значение}\},\ \{\mathrm{R}_1.\mathrm{Y}_1,\ \text{значение}\},\ \{\mathrm{R}_1.\mathrm{Y}_2,\ \text{значение}\}\}$;

G- алгебраическая система, заданная на множестве атрибутов (ввиду очевидности необходимости и применения оставлена за рамками данного рассмотрения);

$\mathrm{V} = \{\mathrm{O}_1.\text{прямоугольник}, \mathrm{O}_2.\text{прямоугольник}, \mathrm{R}_1.\text{соединительная\_линия}\}$;

$\mathrm{R} = \left\{\mathrm{R}_1.\mathrm{X}_1 = \mathrm{O}_1.\mathrm{X} + \mathrm{O}_1.\text{ширина},\ \mathrm{R}_1.\mathrm{Y}_1 = \mathrm{O}_1.\mathrm{Y} + \frac{\mathrm{O}_1.\text{высота}}{2},\ \mathrm{R}_1.\mathrm{X}_2 = \mathrm{R}_1.\mathrm{Y}_2 = \mathrm{O}_2.\mathrm{Y} + \frac{\mathrm{O}_2.\text{высота}}{2}\right\}$.

Естественно, в реальной диаграмме система соотношений будет значительно сложнее, кроме того, сами исходные объекты V могут строиться по тому же принципу из символов визуального алфавита.

Одним из преимуществ подхода является возможность отказа от разделения диаграммы на абстрактный граф синтаксиса (АГС) и пространственный граф отношений (ПГО), которые часто используются в формализмах, связанных с графовыми грамматиками. АГС представляет синтаксическое наполнение, имеющее смысл для трансляции диаграммы, в то время как ПГО описывает интерфейс пользователя [227]. АГС для приведенного примера определяет, что у нас есть два объекта с

соответствующими атрибутами, соединенные некоторым отношением, ПГО содержит информацию о том, как именно оно изображено на экране. Этот подход появился, прежде всего, из-за высокой сложности формализмов графовых грамматик, в виду которой описывать собственно изобразительную часть диаграммы на тех же грамматиках, что и синтаксис, было бы чересчур тяжело. Вычислительная модель диаграммы совершенно свободна от этого недостатка – как описание, так и смысловое наполнение задаются в ней практически единообразно.

Тем не менее, из приведенного описания возникает большое количество проблем. Приведем здесь основные.

Во-первых, требуется формальный алгоритм вычисления указанной семантической сети при взаимодействиях с пользователем.

Во-вторых, пользователь не привык взаимодействовать с диаграммами как с набором элементарных блоков. Вместо этого он оперирует объектами и отношениями.

В-третьих, необходимы четко формализованные правила добавления и удаления частей диаграммы при редактировании.

В-четвертых, требует исследования вопрос конечного графического отображения диаграммы – независимо от универсальности бизнес-логики, ограничения могут возникать из-за отсутствия подходящих возможностей визуального отображения.

Эти проблемы будут рассмотрены подробнее в последующих разделах.

2.2. Алгоритм расчета вычислительной модели визуального языка

В работах [49, 50, 51, 52] Э.Х. Тыугу предложил концепцию вычислительной модели – формализованной семантической сети, в узлах которой находятся переменные, связанные между собой отношениями – математическими операциями. Данная модель также иногда называется алгебраической семантической сетью.

Данная сеть представляет знания о возможностях вычислений. Те или иные операторы должны выполняться в зависимости от определенных условий, а именно – известны значения их параметров, и они вычисляют еще невычисленные переменные. Путем определения пути в сети от цели, которую необходимо рассчитать, до начальных узлов можно выяснить, какой именно порядок операций должен при этом выполняться.

Как было показано Тыугу, подобные модели хорошо описывают простые математические и физические системы, в которых заданы соотношения внутри множеств переменных. Кроме того, в простых случаях вычислительная модель позволяет ответить на вопрос – разрешима ли задача вообще, и, если да, то каков минимальный алгоритм для ее разрешения. Кроме того, вычислительные модели легко комбинируются, в результате чего управление знаниями о предметной области весьма упрощается.

Основным недостатком оригинальной модели является невозможность строить алгоритмы с ветвлением. Для решения этой проблемы Тыугу предложил так называемые расширенные вычислительные модели, дополненные управляющими отношениями, а именно, конкатенацией (вычисления выполняются строго друг за другом), условное отношение – выполняющееся, только если некоторая переменная истинна, а также отношения вызова подзадачи.

Основным преимуществом модели является то, что она прекрасно позволяет представлять знания в тех случаях, когда это знания о математических соотношениях в рамках некоторой предметной области. Именно это свойство моделей Тыугу послужило мотивацией для создания вычислительной модели визуального языка. С другой стороны, важнейшим недостатком указанной модели является недетерминированность ее вычисления и потенциальная противоречивость (Тыугу предложен термин “нестационарность” [52]). Действительно, далеко не обязательно существует только один путь вычисления результата, кроме того, разные пути могут давать разные результаты. Тыугу писал, что “*Введение требований полноты и*

непротиворечивости было бы сильным ограничением при использовании моделей для описания сложных объектов и явлений. Исследование или проектирование объекта часто начинается с незначительного количества сведений, которые в процессе работы изменяются и дополняются. Вычислительные модели вполне подходят для представления таких изменяющихся, неполных и, быть может, даже противоречивых сведений, которыми исследователь или проектировщик располагает. Далее требование однозначности решения задачи ставиться не будет. Однозначность решения может обеспечиваться, с одной стороны, содержанием отношений, образующих модель, и с другой — непротиворечивостью самих задач"[52]. Однако, то, что хорошо для описанной ситуации, не подходит для систем, связанных с взаимодействием с пользователем, в особенности, если это взаимодействие рассматривается как побочный эффект вычисления.

Оригинальная модель Тыугу сравнительно хорошо работает для случаев, когда вычисления всегда идут в одну сторону, от некоторых переменных, образующих входной интерфейс модели, к значениям, которые являются результатом вычислений. В случае вычислительной модели визуального языка, зависимости носят циклический характер, поэтому оригинальная модель Тыугу нежелательна к использованию ввиду недетерминированности вычислений – для редактора диаграмм было бы крайне неудобно, если бы после взаимодействия с пользователем одна диаграмма превращалась в несколько и пользователь должен был бы выбирать ту, с которой он хочет далее продолжить работу. Необходимо предложить правила вычисления семантической сети такие, чтобы они позволяли при правильном проектировании фрагментов сети детерминированные вычисления диаграмм.

Пример 2.2. Рассмотрим типовой пример простого фрагмента диаграммы, имеющий место для большинства диаграммных языков и иллюстрирующий проблему. Пусть задан прямоугольный блок, над которым можно производить следующие действия – менять его размер, перемещая углы

блока. Такой фрагмент может быть описан следующей семантической сетью (рис. 2.3):

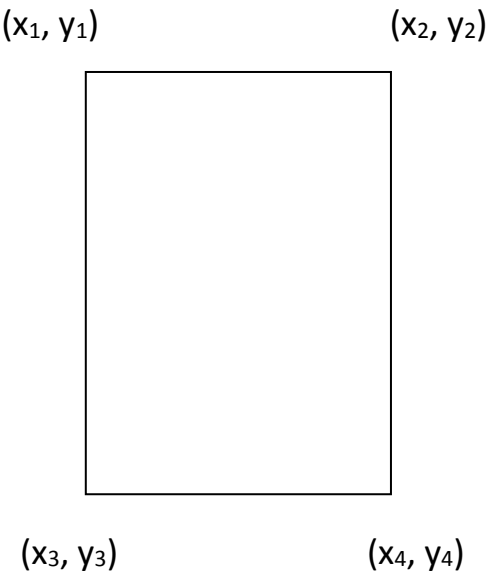


Рис.2.3. Пример типовой фигуры

$x_1=x_3$ (2.7)	$y_1=y_2$ (2.11)	$x_2=x_1+width$ (2.15)	$y_3=y_1+height$ (2.21)
$x_3=x_1$ (2.8)	$y_2=y_1$ (2.12)	$x_1=x_2-width$ (2.16)	$y_1=y_3-height$ (2.22)
$x_2=x_4$ (2.9)	$y_3=y_4$ (2.13)	$x_4=x_3+width$ (2.17)	$y_4=y_2+height$ (2.23)
$x_4=x_2$ (2.10)	$y_4=y_3$ (2.14)	$x_3=x_4-width$ (2.18)	$y_2=y_4-height$ (2.24)
		$width=x_2-x_1$ (2.19)	$height=y_3-y_1$ (2.25)
		$width=x_4-x_3$ (2.20)	$height=y_4-y_2$ (2.26)

Здесь width – ширина прямоугольника, а height – высота.

Построим соответствующую этим правилам семантическую сеть, фрагмент которой приведен на рис. 2.4.

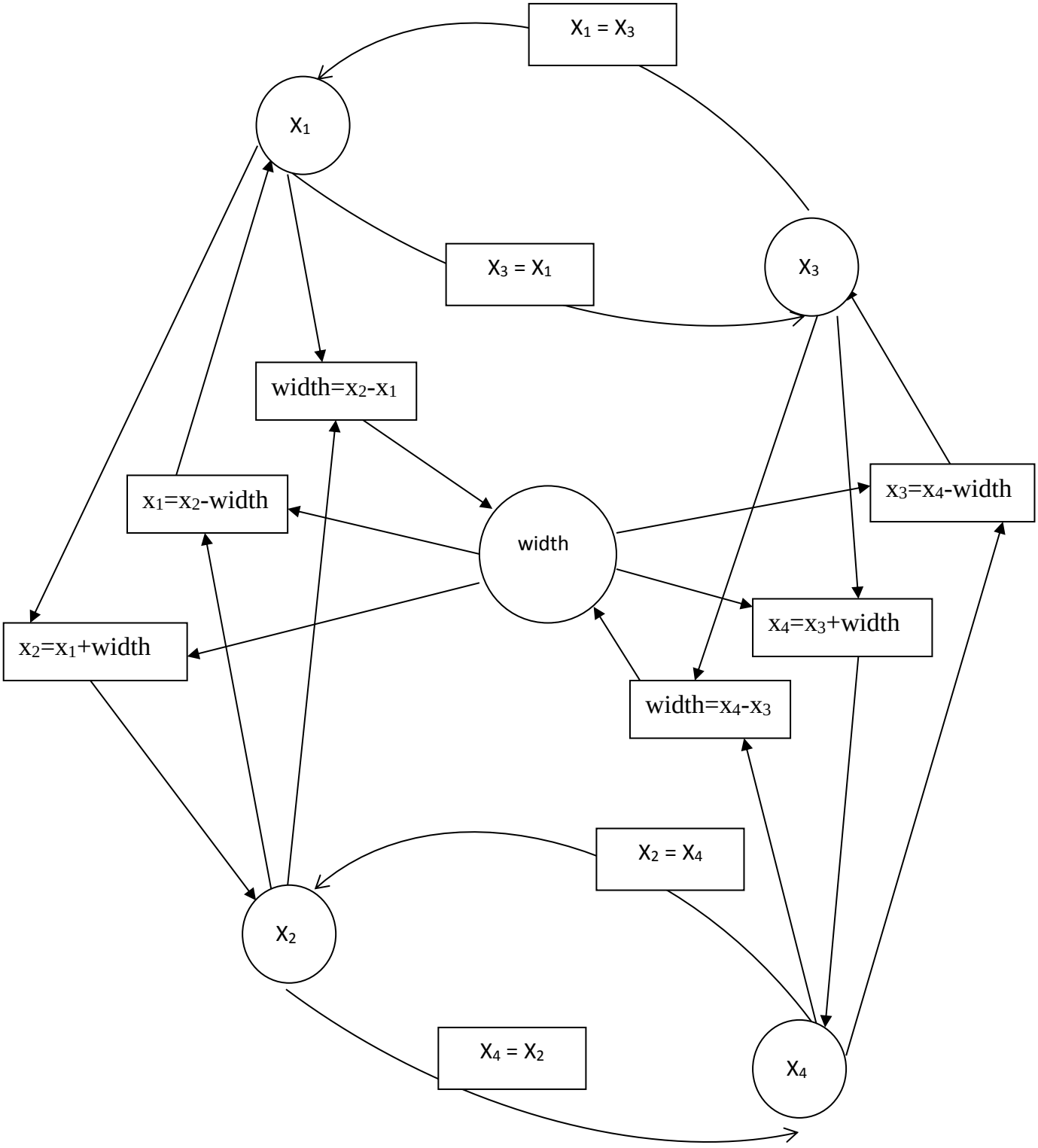


Рис. 2.4. Фрагмент семантической сети, описывающей типовую фигуру.

На приведенном рисунке изображена часть семантической сети, соответствующая правилам с (2.7) по (2.10) и с (2.15) по (2.20) из примера 2.2.

Узлы, обозначенные окружностями, представляют значения переменных. Узлы – прямоугольники представляют математические соотношения. Связи представляют участие в математических соотношениях, причем связь может соединять только узлы разного типа (т.е. граф двудольный); если связь выходит из переменной и входит в соотношение, то эта переменная входная, если наоборот – переменная выходная. Очевидно, из каждого соотношения может выходить только одна связь.

Предположим, пользователь передвинул левый верхний угол прямоугольника. На рис. 2.4. видно, что существуют различные пути пересчета модели, потенциально дающие различные результаты. Например, вычисление может в начале пойти по пути (2.19) → (2.17), а может пойти по пути (2.8) → (2.15). В первом случае фигура изменит свой размер, а во втором – нет.

Таким образом, очевидно, что необходимы детерминированные правила вычисления получившейся семантической сети.

Для дальнейшего изложения введем несколько определений.

Определение 2.4. Воздействием на диаграмму $D = \{A, G, V, R\}$ называется подмножество множества

$A^* = \{a_1^*, a_2^*, \dots a_n^*\}$,

где для любого $a_i = \{n_i, v_i\} \in A$ верно, что $a_i^* = \{n_i, {v_i}^*\}$, т.е. атрибуты с совпадающими именами имеют, возможно, различные значения.

Со смысловой точки зрения воздействие – это множество атрибутов с новыми значениями, которые получили эти значения в процессе взаимодействия с диаграммой. Теперь можно дать описание алгоритма, вычисляющего атрибуты диаграммы при возникновении воздействия на нее.

Алгоритм 2.1 (алгоритм вычисления диаграммы при воздействии). Вычисление состоит из двух этапов. На первом этапе осуществляется

первоначальное изменение некоторых переменных (взаимодействие с моделью). На втором этапе применяются правила до тех пор, пока не обнаруживается, что больше не осталось применимых правил. Любое правило из множества R (см. формулу 2.6) может быть выполнено при соблюдении следующих условий:

1. v_i еще не получило новое значение;
2. $v_n ... v_m$ содержат хотя бы одну переменную, получившую новое значение;
3. Условие *cond* истинно.

Определение 2.5. Назовем детерминированным вычисление диаграммы, при котором алгоритм 2.1, независимо от воздействия, генерирует только одну последовательность применения правил.

Определение 2.6. Назовем непротиворечивым вычисление диаграммы, при котором алгоритм 2.1, независимо от воздействия, приводит к одному и тому же состоянию диаграммы.

Вычисления семантической сети происходят в общем случае недетерминировано (нестационарно), что является проблемой, в частности, приводящей к комбинаторному взрыву, а также к непредсказуемому поведению визуальной составляющей. Для того, чтобы модель получила прикладное значение, ее необходимо привести к детерминированному или хотя бы непротиворечивому виду. Стандартные алгоритмы для приведения к детерминированному виду конечных автоматов [1] применены здесь не могут быть, так как исходные узлы могут быть связаны с визуальными элементами и их преобразования разорвут эту связь. В связи с этим возникает настоятельная необходимость применения алгоритма 2.2, описание которого приведено ниже.

Алгоритм 2.2 (алгоритм приведения вычислительной модели визуального языка к детерминированному виду)

Пусть даны три функции:

$P_i(a)$, возвращающая истину, если x принадлежит воздействию и ложь в противном случае;

$P(a)$, возвращающая истину, если x уже получил новое значение при выполнении алгоритма 2.1, и ложь - в противном случае и

$F(a_1, a_2, \dots a_i)$, возвращающая для каждого возможного воздействия I_j последовательность вычисления узлов $n_j = \left\{a_{n_j,1}, a_{n_j,2}, \dots a_{n_j,k_j}\right\} \epsilon N$, где N - множество всех последовательностей, k_j- длина последовательности n_j, а $a_{n_j,m}$- атрибут, вычисляемый на шаге m этой последовательности. Для простоты можно считать, что функция возвращает индекс j последовательности из множества N. Эта функция может быть преобразована в функцию $F'(a_1, a_2, \dots a_i, j) \equiv F(a_1, a_2, \dots a_i) = j$, а F' уже выражена через $P_i(a)$ и $P(a)$ а также функции математической логики.

Пусть также задано множество правил $R' = \left\{r_{n_j,m}\right\}$, где $r_{n_j,m}$ - правило, которое должно быть применено для вычисления атрибута $a_{n_j,m}$ на шаге m последовательности n_j в соответствии с условиями алгоритма 2.1.

Для каждого значения n_j выполним следующие действия:

Введем в диаграмму атрибуты $a'_{n_j,0}, a'_{n_j,1}, \dots a'_{n_j,k_j}$ логического типа,

Каждое правило $r_{n_j,m} : cond\left(c_{n_j,m}\right) : a_{n_j,m} = f_{n_j,m}(A);\ a_{n_j,m} \in A, m \in 1..k_j$ модифицируем в форму $r_{n_j,m} : cond\left(c_{n_j,m} \wedge F'(a_1, a_2, \dots a_i, j) \wedge \bigwedge_{i=1}^{m-1} P(a'_{n_j,i})\right) : a_{n_j,m} = f_{n_j,m}(A);\ a_{n_j,m} \in A, m \in 1..k_j.$

В результате выполнения алгоритма будет получена новая семантическая сеть, вычисление по которой будет происходить детерминированно.

Продемонстрируем приведение модели к детерминированному виду. Решим предыдущий пример, изображенный на рис.2.4, и определяющий прямоугольник с перемещаемыми углами. Формулы (2.7) –(2.26) построены так, что при перемещении угла фигуры в рамках алгоритма 2.1 допустимо как произвести пересчет остальных углов, а потом ширины и высоты, так и произвести эти действия в обратном порядке. В результате фигура может как изменить размер, так и переместиться на другое место без изменения размера, либо изменить размер по одной оси и переместиться по другой. Подобный недетерминизм неприемлем для пользователя. Поэтому необходимо четко определить порядок, в котором будут выполняться вычисления, например, следующим образом:

$$n_1 : P_i(x_1) \wedge P_i(y_1) \vdash \begin{cases} x_3 = x_1; \\ y_2 = y_1; \\ width = x_2 - x_1; \\ height = y_3 - y_2; \end{cases} \quad (2.27)$$

$$n_2 : P_i(x_2) \wedge P_i(y_2) \vdash \begin{cases} x_4 = x_2; \\ y_1 = y_2; \\ width = x_2 - x_1; \\ height = y_3 - y_2; \end{cases} \quad (2.28)$$

$$n_3 : P_i(x_3) \wedge P_i(y_3) \vdash \begin{cases} x_1 = x_3; \\ y_4 = y_3; \\ width = x_2 - x_1; \\ height = y_3 - y_2; \end{cases} \quad (2.29)$$

$$n_4 : P_i(x_4) \wedge P_i(y_4) \vdash \begin{cases} x_2 = x_4; \\ y_3 = y_4; \\ width = x_2 - x_1; \\ height = y_3 - y_2. \end{cases} \quad (2.30)$$

В этом случае функция F может иметь следующий вид:

$$F'(a_1, a_2, \dots a_i, j) = P_i(x_1) \wedge P_i(y_1) \wedge j \equiv 1 \vee P_i(x_2) \wedge P_i(y_2) \wedge j \equiv 2 \vee P_i(x_3) \wedge P_i(y_3) \wedge j \equiv 3 \vee P_i(x_4) \wedge P_i(y_4) \wedge j \equiv 4 \quad (2.31)$$

Однако, для простоты и без потери общности для каждого n_j может быть использована функция

$$P_i(x_j) \wedge P_i(y_j) \quad (2.32)$$

Наконец, построим множество правил R (см. формулу 2.6)

$$r_{n_1,1} : cond(P_i(x_1) \wedge P_i(y_1)) : x_3 = x_1 \quad (2.33)$$

$$r_{n_1,2} : cond(P_i(x_1) \wedge P_i(y_1) \wedge P(x_3)) : y_2 = y_1 \quad (2.34)$$

$$r_{n_1,3} : cond(P_i(x_1) \wedge P_i(y_1) \wedge P(x_3) \wedge P(y_2)) : width = x_2 - x_1 \quad (2.35)$$

$$r_{n_1,4} : cond(P_i(x_1) \wedge P_i(y_1) \wedge P(x_3) \wedge P(y_2) \wedge P(width)) : height = y_3 - y_2 \quad (2.36)$$

$$r_{n_2,1} : cond(P_i(x_2) \wedge P_i(y_2)) : x_4 = x_2 \quad (2.37)$$

$$r_{n_2,2} : cond(P_i(x_2) \wedge P_i(y_2) \wedge P(x_4)) : y_1 = y_2 \quad (2.38)$$

$$r_{n_2,3} : cond(P_i(x_2) \wedge P_i(y_2) \wedge P(x_4) \wedge P(y_1)) : width = x_2 - x_1 \quad (2.39)$$

$$r_{n_2,4} : cond(P_i(x_2) \wedge P_i(y_2) \wedge P(x_4) \wedge P(y_1) \wedge P(width)) : height = y_3 - y_2 \quad (2.40)$$

$$r_{n_3,1} : cond(P_i(x_3) \wedge P_i(y_3)) : x_1 = x_3 \quad (2.41)$$

$$r_{n_3,2} : cond(P_i(x_3) \wedge P_i(y_3) \wedge P(x_1)) : y_4 = y_3 \quad (2.42)$$

$$r_{n_3,3} : cond(P_i(x_3) \wedge P_i(y_3) \wedge P(x_1) \wedge P(y_4)) : width = x_2 - x_1 \quad (2.43)$$

$$r_{n_3,4} : cond(P_i(x_3) \wedge P_i(y_3) \wedge P(x_1) \wedge P(y_4) \wedge P(width)) : height = y_3 - y_2 \quad (2.44)$$

$$r_{n_4,1} : cond(P_i(x_3) \wedge P_i(y_3)) : x_2 = x_4 \quad (2.45)$$

$$r_{n_4,2} : cond(P_i(x_3) \wedge P_i(y_3) \wedge P(x_2)) : y_3 = y_4 \quad (2.46)$$

$$r_{n_4,3} : cond(P_i(x_3) \wedge P_i(y_3) \wedge P(x_2) \wedge P(y_3)) : width = x_2 - x_1 \quad (2.47)$$

$$r_{n_4,4} : cond(P_i(x_3) \wedge P_i(y_3) \wedge P(x_2) \wedge P(y_3) \wedge P(width)) : height = y_3 - y_2 \quad (2.48)$$

Результирующая сеть вычисляется алгоритмом 2.1 детерминированно.

Пример 2.3. Для приведения диаграммы к непротиворечивому виду могут использоваться значительно более простые преобразования. Например, рассмотрим модифицированный фрагмент диаграммы с рис. 2.4.

$$\left.\begin{array}{llll} x_1=x_3 & y_1=y_2 & P(width):x_2=x_1+width & P(height):y_3=y_1+height \\ x_3=x_1 & y_2=y_1 & P(width):x_1=x_2-width & P(height):y_1=y_3-height \\ x_2=x_4 & y_3=y_4 & P(width):x_4=x_3+width & P(height):y_4=y_2+height \\ x_4=x_2 & y_4=y_3 & P(width):x_3=x_4-width & P(height):y_2=y_4-height \\ & & width=x_2-x_1 & height=y_3-y_1 \\ & & width=x_4-x_3 & height=y_4-y_2 \end{array}\right\} \quad (2.49)$$

Можно убедиться, что все варианты пересчета сети дадут одинаковый результат, т.е. диаграмма непротиворечива.

Таким образом, было дано формальное определение вычислительной модели диаграммы, введены алгоритмы расчета этой модели и приведения модели к детерминированному виду. Следующим необходимым шагом является определение визуального синтаксиса диаграммы, заданной в вычислительной модели, т.е. непосредственного способа изображения ее на экране.

2.3. Визуальный синтаксис диаграммы

В предыдущем разделе было показано, как параметры, на основе которых формируется изображение диаграммы, могут быть преобразованы в вычислительную модель визуального языка. Следующим шагом необходимо определить способ изображения диаграммы на основе этих параметров. Собственно изображение диаграммы формируется за счет некоторых визуальных примитивов с преопределенным поведением, имеющих атрибуты, указывающие, как их рисовать, и описываемые семантической сетью. Набор

этих атрибутов образует визуальный алфавит диаграммы, а взаимодействие с остальной семантической сетью – ее визуальный синтаксис.

Визуальный примитив фактически представляется фрагментом семантической сети, которому сопоставлены некоторые знания о том, как он взаимодействует с пользователем. Набор таких примитивов не обязательно ограничен чисто графическими элементами – в этой роли могут выступать объекты, издающие звук или, например, взаимодействующие с файловой системой.

Пример 2.4. В качестве примера рассмотрим примитив – отрезок прямой, который характеризуется четырьмя координатами, вступающими в отношения с атрибутами более общей семантической сети. Прямоугольник строится из четырех таких примитивов и т. д. На рис. 2.5 показано, как прямоугольник строится из четырех объектов – отрезков, соединенных соотношениями равенства своих координат.

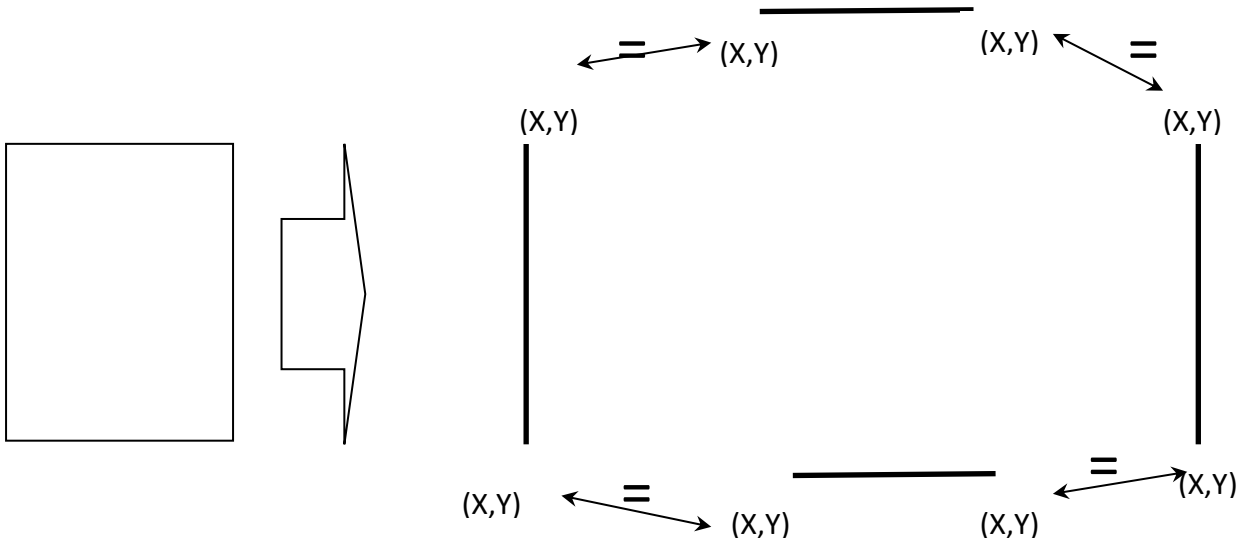


Рис. 2.5. Пример построения графического элемента из примитивов

Данный подход приводит к трем ключевым проблемам. Первой из них является выбор множества визуальных примитивов. Очевидно, что создать исчерпывающее множество невозможно. Таким образом, если представление части, связанной с абстрактных графом синтаксиса, не требует для своего описания ничего, кроме структур данных, то необходимость визуального

отображения приводит к вопросу о том, где установить черту, за которой дальнейшее наращивание визуального алфавита становится нецелесообразным. Иными словами, потенциальная возможность нарисовать диаграмму ограничивается количеством и характером визуальных элементов, которые могут присутствовать в инструменте, рисующем эту диаграмму, и их количество счетно. Возвращаясь к примеру с отрезком прямой, в самом простом случае он может иметь четыре атрибута. В более сложном случае мы можем учитывать толщину линии, цвет и стиль, наличие стрелок на концах и т.д. Фактически, потенциальное количество требующихся атрибутов не ограниченно.

Возможно несколько очевидных подходов к решению этой задачи. Во-первых, это классификация известных диаграмм по типам объектов и попытка выделения базового набора визуальных примитивов. Во-вторых, это подходы на основе перечня примитивов визуальных редакторов. Подход на основе классификации известных диаграмм обладает тем преимуществом, что гарантирует возможность изображения того множества диаграмм, которые были проанализированы, однако оставляет вопрос относительно диаграмм, которые не были рассмотрены на момент классификации. С другой стороны, опираясь на возможности визуальных редакторов можно считать, что все, что может быть нарисовано визуальным редактором, может быть также использовано как визуальный алфавит диаграммы. Таким образом, второй подход, хотя и сопряжен с большими трудностями (потенциально огромное количество атрибутов), является более универсальным.

Вторая важная проблема, появляющаяся из-за самой природы модели – привязка к графическим устройствам, фактически, делающая диаграмму зависимой от отображения (что нарушает шаблон Model-View-Controller). Предположим, что у нас есть графический элемент “текст”, для которого задаются атрибуты – координаты левого верхнего угла, шрифт, собственно текст, а также длина и высота, вычисляемые на основе предыдущих параметров. Однако для того, чтобы рассчитать длину и высоту текста,

недостаточно знать шрифт и сам текст, необходим также графический контекст. Таким образом, в зависимости от того, на каком устройстве рисуется диаграмма, она должна пересчитываться. Важной проблемой, возникающей в результате указанного недостатка, является невозможность одновременного отображения диаграммы на двух различных устройствах.

Проблема зависимости от графического контекста может быть решена одним из двух способов. Первый – это создание некоторого промежуточного абстрактного графического контекста, который бы преобразовывал визуальную модель в аппаратно-независимое изображение, которое в дальнейшем уже, собственно, и изображалось в конечном устройстве. Данный способ достаточно прост, но не универсален, так как не позволяет передавать назад в модель данные о настройках оконечного устройства что, в свою очередь, делает графическое представление неоптимальным для наблюдателя.

Эта же проблема может быть решена путем созданий копий графических элементов для каждого из графических устройств. В таком случае семантическая сеть должна быть разбита на две части и запрещены любые функции, вычисляющие значения атрибутов из невизуальной части по атрибутам из визуальной (но не наоборот). Преимуществом такого решения является большая универсальность, а очевидным недостатком - исключительная сложность.

Наконец, третья проблема заключается в поддержке взаимодействия между пользователем и визуальным элементом. Именно визуальные элементы являются для пользователя теми сущностями, через которые возможно осуществить то или иное воздействие на диаграмму – например, перемещение объекта, его удаление или вызов контекстного меню. Необходима поддержка взаимодействия с пользователем именно на этом уровне.

Для решения третьей проблемы предлагается создавать невидимые графические примитивы, предназначенные для обработки определенных событий интерфейса. Таким образом, возможно сочетать произвольное

визуальное представление с произвольным событием интерфейса. В качестве таких событий можно рассматривать, например, нажатие кнопки мыши.

Таким образом, было показано, что визуальное отображение диаграммы может быть обеспечено за счет набора визуальных примитивов, связанных с ее моделью через атрибуты. При использовании такого подхода возникают три проблемы. Первая – выбор множества графических примитивов – может быть решена либо с помощью классификации декомпозиций визуальных элементов известных диаграмм, либо с помощью синтеза на основе возможностей современных графических редакторов. Вторая проблема – невозможность отображения диаграммы в отрыве от конкретного графического устройства – может быть решена либо с помощью введения промежуточного уровня – абстрактного графического устройства, либо путем разделения семантической сети на невизуальную и визуальную, специфичную для каждого устройства в отдельности. Наконец, третья проблема – взаимодействия с пользователем – решается с помощью введения невидимых графических примитивов, предназначенных для обработки соответствующих событий интерфейса.

2.4. Логические объекты диаграммы

Пользователь, оперируя с диаграммой, привык работать с объектами, а не с их элементарными частями. В частности, задача удаления объектов для пользователя представляется атомарной, что требует хранения в модели информации о том, какие именно ее части образуют удаляемый объект. Таким образом, необходимо обеспечить композицию графических примитивов и атрибутов семантической сети в более крупные элементы – логические объекты, которыми можно было бы оперировать как единым целым.

Диаграмма может быть представлена пользователю тремя различными способами, два из которых уже были рассмотрены. Во-первых, диаграмма – это изображение (визуальный уровень). Во-вторых, диаграмма имеет определенное информационное наполнение, на основании которого с диаграммой можно взаимодействовать (уровень поведения или бизнес-

логики). Наконец, для решения поставленной задачи – взаимодействия с фрагментами диаграммы как с единым целым – предлагается ввести дополнительный логический уровень, состоящий из объектов диаграммы.

Рассмотрим уже приводившуюся систему из двух сущностей и связи между ними. Эта система представима следующим образом (рис. 2.6):

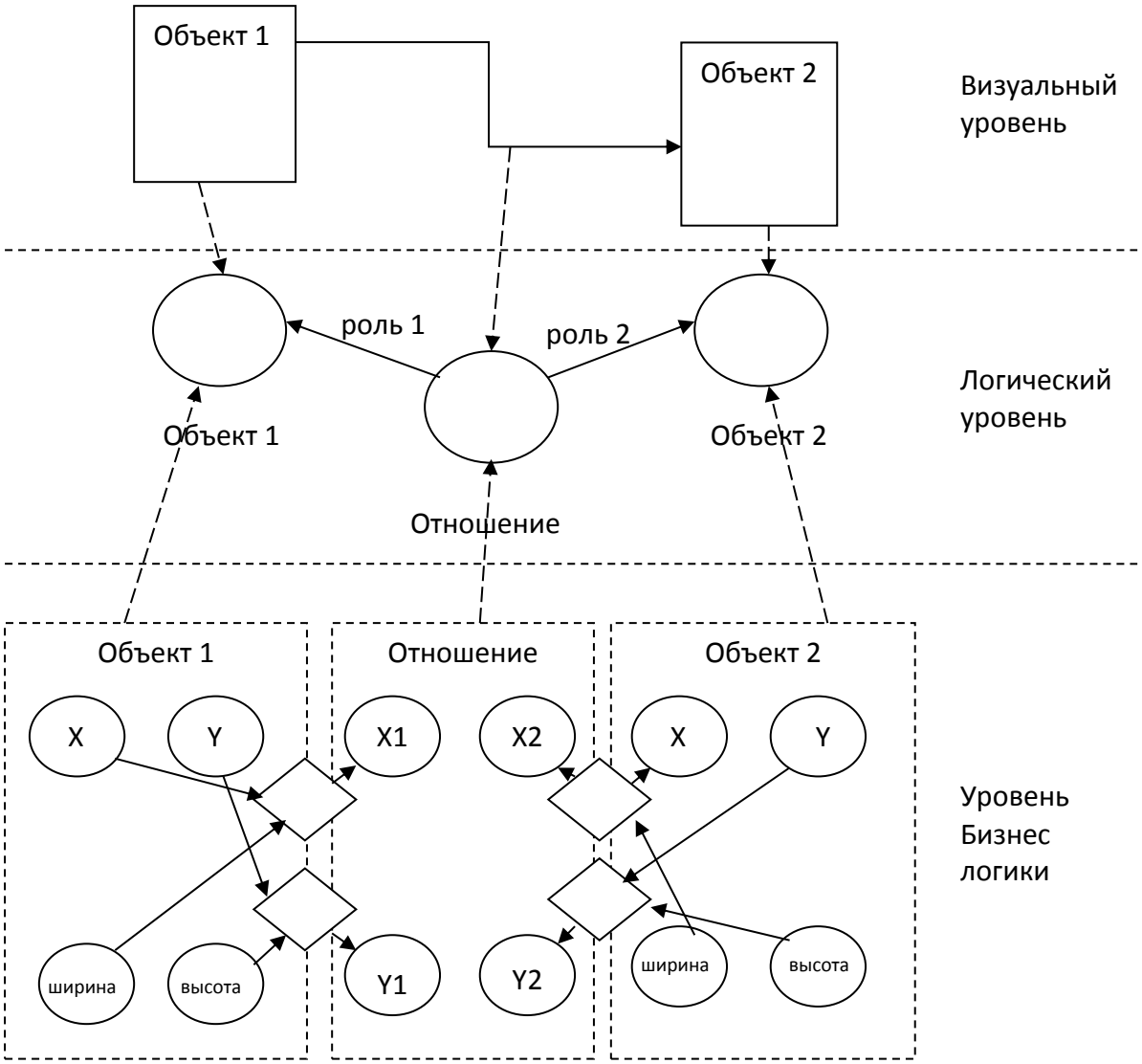


Рис.2.6. Уровни представления диаграммы

Упомянутый подход порождает сразу несколько задач. Во-первых, необходимо обеспечить однозначное соответствие между всеми элементами объекта и самим объектом. Во-вторых, необходимо обеспечить правильную обработку разрыва отношений между объектами при удалении одного из них (например, при удалении сущности в диаграмме “сущность-связь” должны удалиться и все связи, входящие или выходящие из нее).

Наиболее очевидным решением первой задачи является рассмотрение логического объекта как еще одного узла (концепта) семантической сети специального типа, связанного отношениями вида “входит в” или “состоит из” с другими узлами. Таким образом, по визуальному элементу можно установить, к какому именно логическому объекту он относится, после чего определить все элементы, из которых этот объект состоит. При необходимости удаления этого объекта достаточно просто удалить из семантической сети все концепты, входящие в этот объект (если объект не связан отношениями с другими объектами). Недостатком подхода является некоторое усложнение модели.

Более сложной является задача разрыва отношений между объектами при удалении одного из них. Такая задача может требовать каскадного удаления большого количества объектов. Тем самым, возникает вопрос – как именно следует информировать семантическую сеть о подобных взаимосвязях. Простым решением этой задачи является введение операции связывания узлов - логических объектов отношениями принадлежности. Таким образом, например, в диаграмме “сущность-связь” сущность, вступая в отношение со связью, может устанавливать отношение принадлежности этой связи к сущности и при удалении сущности связь также будет удаляться. Сложные иерархии принадлежности логических объектов могут приводить к любому каскадному удалению, которое необходимо. При этом следует иметь в виду тот недостаток решения, что невозможно выполнить условное удаление в зависимости от значений концептов-атрибутов.

Задача преобразования семантической сети при вставке объекта в диаграмму и удалении объекта из нее является крайне важной. При этом эти действия не обязательно связаны непосредственно с действиями пользователя. Требования предметной области могут быть таковы, что подобные преобразования могут выполняться в ответ не на события вставки или удаления, а на изменение значения некоторого концепта. Простым примером такой диаграммы может быть игра “жизнь” [15], предполагая, что она ведется

на бесконечном поле и каждая ячейка в ней реализована как объект. С функцией времени количество объектов на игровом поле существенно меняется, однако пользователь не участвует в этом процессе напрямую. Вместо этого изменение является частью синтаксиса диаграммы.

Как было уже показано ранее, объект диаграммы представляется фрагментом семантической сети, который необходимо по определенным правилам в эту сеть вставить либо который необходимо из этой сети изъять. Таким образом, логичным видится, что изменение топологии семантической сети может являться побочным эффектом от ее вычисления таким же образом, как это происходит в функциональных языках (что немедленно вызывает предположение о предпочтительности чисто функционального стиля описания диаграмм).

Таким образом, предлагается ввести особого рода функции, выполнение которых приводило бы к изменению топологии диаграммы. Вызов этих функций мог бы происходить в процессе пересчета диаграммы и, как побочное действие этого пересчета, добавлять в диаграмму новые объекты либо удалять существующие. Это позволило бы моделировать добавление или удаление объектов, например, в ответ на перемещения других объектов либо в ответ на внешние воздействия или воздействия таймера.

Еще одним важным аспектом является декомпозиция. Во многих диаграммах существуют объекты, содержащие внутри себя другую диаграмму. В вычислительной модели визуального языка эта задача также имеет естественное решение. Семантическая сеть, представляющая внутреннюю диаграмму, образует единое целое с внешней сетью, будучи связанной с ней через атрибуты декомпозируемого объекта. Таким образом, с одной стороны, внутренняя часть сети может представлять отдельную диаграмму, с другой стороны, данные этой диаграммы естественным образом связаны с основной диаграммой.

Итак, было показано, что за счет введения концептов нового типа – логических объектов и отношений принадлежности между ними и другими

концептами появляется возможность оперировать со сложными фрагментами диаграммы как с единой синтаксической единицей. При этом решена задача разрыва связей между объектами при удалении одного из них и каскадного удаления. Кроме того, показано, что вставка и удаление объектов может естественным образом производиться в ходе обычного пересчета сети и не обязательно требовать действия пользователя. Наконец, продемонстрировано, что декомпозиция логического объекта также имеет естественное решение в рамках вычислительной модели.

2.5. Интерфейс взаимодействия с пользователем

Взаимодействие с диаграммой для пользователя не ограничивается только обработкой событий от мыши, рассмотренной в разделе 2.3. Вообще говоря, источник взаимодействия может быть произвольным. Можно привести следующий список возможных видов взаимодействия:

1. Взаимодействия со стандартными методами ввода операционной системы (в основном, клавиатура и мышь, а также потенциально оконные события).
2. Взаимодействие с диалоговыми окнами.
3. Взаимодействие с функцией времени.
4. Взаимодействие со сторонними программами.

Тем не менее, эти виды взаимодействия также могут быть промоделированы через определение воздействия на диаграмму и применение алгоритма 2.1. В частности, вопросы реакции на взаимодействия стандартными методами операционной системы могут быть решены путем добавления визуальных примитивов - слушателей, поддерживающих реакцию на соответствующие воздействия. Например, для создания прямоугольника, который можно двигать левой кнопкой мыши и выводить контекстное меню правой кнопкой мыши, необходимо добавить к графическим элементами две активные области, реагирующие на соответствующие события от мыши.

Важной задачей является редактирование объектов с помощью диалоговых окон. Диалоговое окно должно изменять значения некоторых атрибутов точно тем же способом, что и остальные виды взаимодействия, после чего сеть должна пересчитываться так же, как при других видах взаимодействия. Иными словами, необходимо описать дескриптор окна, связывающий формат окна с концептами диаграммы (причем, этот дескриптор потенциально может брать свои данные из той же сети, скажем, редактор концепта может быть выполнен в виде выпадающего списка, возможные значения для которого содержатся в другом узле сети).

Кроме того, диаграмма может быть динамической, т.е. меняться как функция времени. Для решения этой задачи предлагается использовать события таймера, связанные со значением некоторого концепта. Соответственно, с течением времени значение концепта будет изменяться и диаграмма пересчитываться. На основе этого подхода возможно демонстрировать динамические изображения вплоть до видео.

Наконец, важной возможностью является взаимодействие со сторонними программами. При наличии некоторого открытого интерфейса, позволяющего устанавливать значения концептов, сторонние приложения могут строить свой интерфейс на основе существующих диаграмм. Например, известны параметры летательного аппарата, эти параметры передаются в диаграмму, после чего по известным формулам вычисляется и рисуется схематическое изображение положения аппарата в пространстве.

Таким образом, все виды взаимодействия с диаграммой естественным образом реализуются с помощью изменения значений некоторого набора атрибутов вычислительной модели и применения алгоритма 2.1. Подобный подход может использоваться для всех видов взаимодействия диаграмм с пользователем, а также для создания динамических диаграмм и визуализации данных сторонних систем.

2.6. Циклические вычисления

При использовании алгоритма 2.1 возникает проблема выполнения циклических вычислений. Так как каждый концепт в рамках одного цикла вычислений может быть вычислен только один раз, то на первый взгляд наиболее прямолинейным решением для организации цикла будет трактовать тот факт, что переменная уже изменила значение в рамках пересчета модели, как некоторый признак (признак вычисленности), который может быть снят в процессе вычисления других концептов. Тем самым, в рамках одной итерации, в зависимости от условия выхода из цикла, признак может сниматься (возврат на итерацию) или устанавливаться (выход из цикла). Например, подобная семантическая сеть, вычисляющая факториал, изображена на рис.2.7.

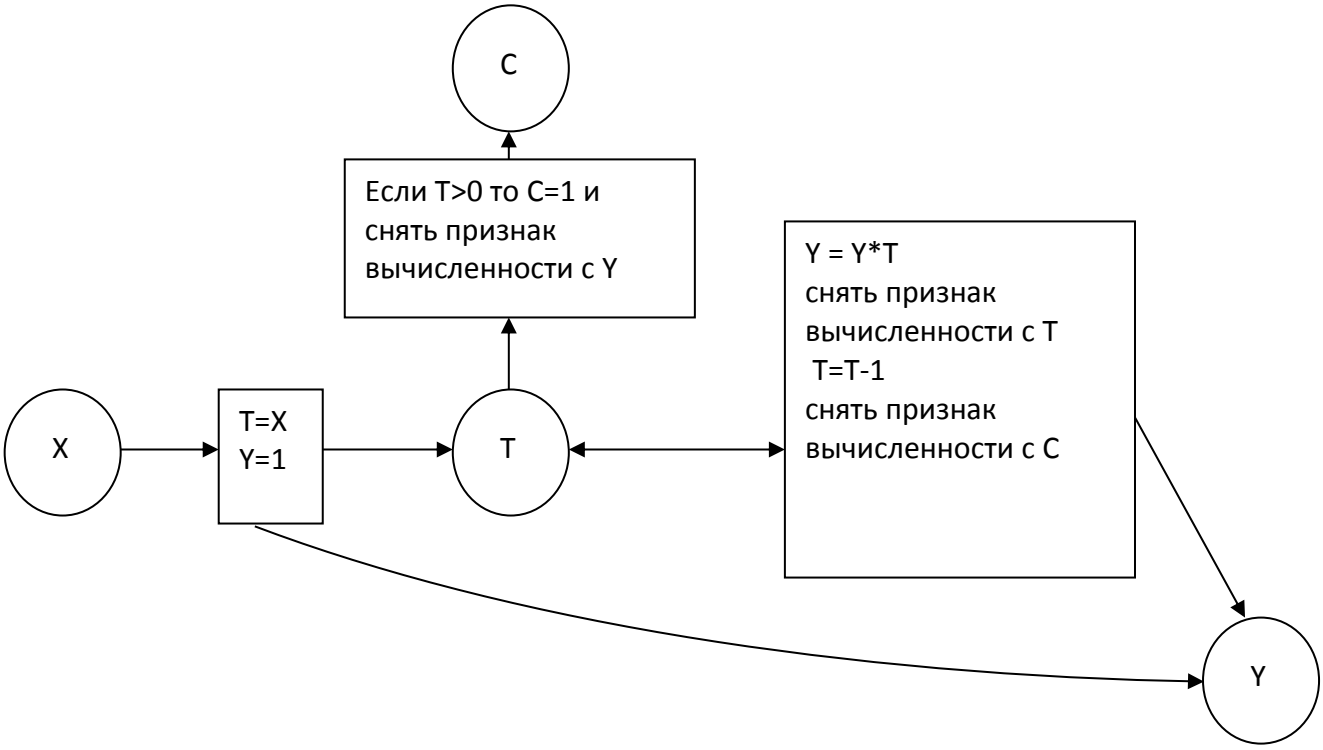


Рис.2.7. Семантическая сеть, вычисляющая факториал

Однако подобный подход обладает рядом недостатков. Прежде всего, как можно увидеть, уже для столь простого алгоритма, как факториал, требуется ведение некоторых искусственных сервисных узлов, побочных эффектом вычисления которых являлось бы снятие нужных признаков

вычисленности (в определенном смысле эти сервисные узлы заменяют итераторов цикла). Во-вторых, так как результат является побочным действием, это затруднило бы автоматический анализ и оптимизацию алгоритма (которая в оригинальной модели возможна по аналогии с чистым функциональным программированием). Таким образом, подход через признаки вычисленности проблематичен.

Наиболее удачным представляется подход через рекурсивное обращение к фрагментам диаграммы. Как было показано в подразделе 2.4, узлы диаграммы могут объединяться в логические объекты. Выделив интерфейс такого объекта и имея функцию вставки объекта в диаграмму, можно описать рекурсивные алгоритмы способами, характерными для функциональных языков, как, например, алгоритм вычисления факториала f=x! , схема которого изображена на рис.2.8.

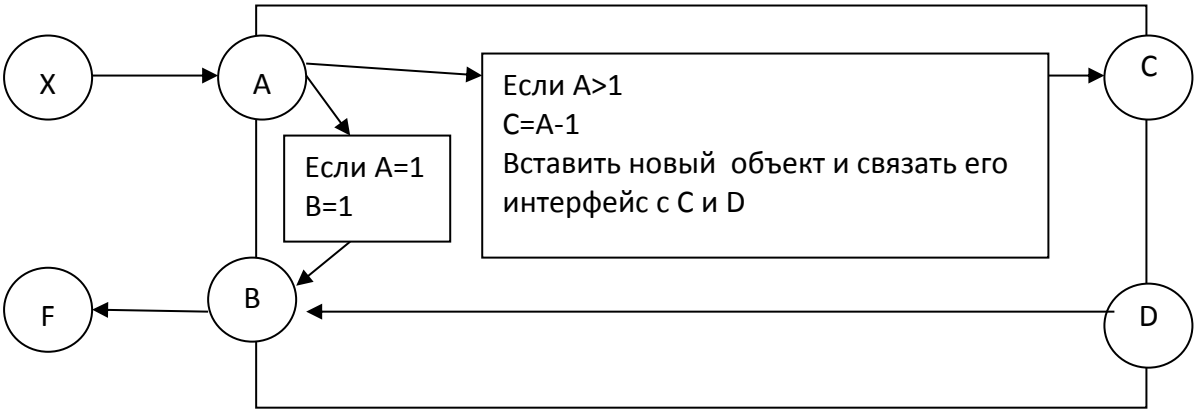


Рис.2.8. Рекурсивное вычисление факториала

Таким образом, сеть меняет топологию по мере вычислений (что подразумевает необходимость удаления ненужных объектов после окончания вычислений). В качестве примера использования можно привести вычисление факториала, например, числа 2 (рис. 2.9).

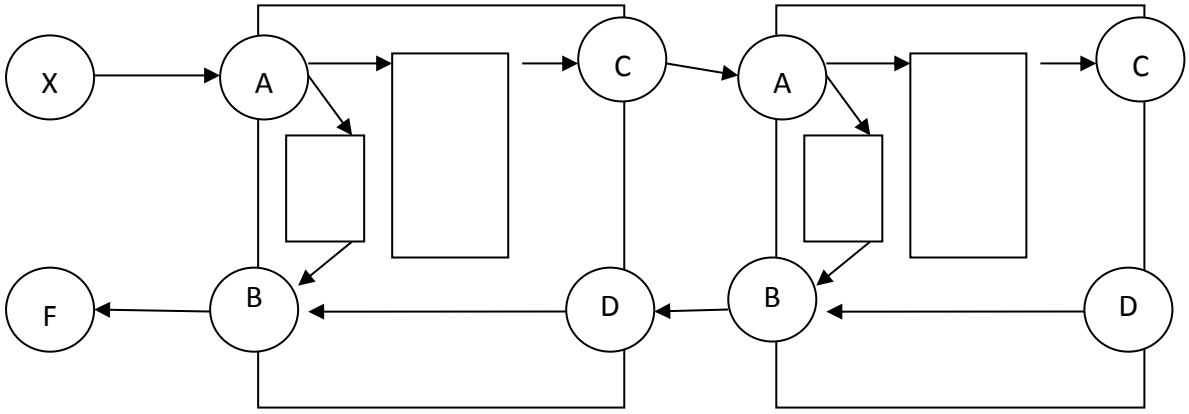


Рис. 2.9. Пример расширения сети для вычисления факториала числа 2

Таким образом, данная модель позволяет производить циклические вычисления.

2.7. Описание визуального языка

До сих пор вычислительная модель рассматривалась здесь применительно к некоторой диаграмме. Однако, конечной задачей данной работы является предложить математическую модель для описания языка диаграмм, а не конкретных диаграмм на этих языках. Таким образом, необходимо предложить способ описания правил, по которым пользователь сможет строить диаграмму.

Пользователь в процессе работы оперирует объектами и правилами трансформации диаграммы (т.е., в некотором приближении, продукционными правилами). Представляется логичным, что язык должен быть описан на основе перечня объектов и правил трансформации, заданных в вычислительной модели.

Объект в вычислительной модели является совокупностью концептов-атрибутов, концептов – функций, концептов – графических примитивов и концептов - других объектов. Таким образом, объект сам является диаграммой, которая может быть описана способом, описанным в предыдущих подразделах.

С другой стороны, правила трансформации фактически состоят из двух частей. Во-первых, это правила соединения концептов объекта с концептами диаграммы. Во-вторых, это некоторое поведение, которое должна продемонстрировать диаграмма после включения в нее нового объекта.

Первая часть решается относительно легко. У объекта выделяется некоторый интерфейс, при добавлении этого объекта в существующую диаграмму указывается, с какими из концептов этой диаграммы он связывается и какими именно функциями. Таким образом, для решения первой задачи необходимо определить для каждого объекта его возможные интерфейсы (так называемые роли) а также для каждой роли перечень других участвующих в отношении ролей и правила установления функциональных зависимостей между ними.

Вторая часть является естественным следствием структуры модели. При добавлении объекта в диаграмму его концепты-атрибуты являются невычисленными и в процессе их вычисления может быть реализовано необходимое поведение. Кроме того, возможно добавить специальный набор функций, выполняющихся в ответ на добавление или удаление объекта.

Таким образом, описание языка в вычислительной модели строится на основе описания фрагментов семантической сети и правил их вставки в диаграмму. При этом возможно поддержать как синтаксис самих объектов, так и специальное поведение диаграммы в ответ на изменение ее топологии.

2.8. Трансляция диаграмм в вычислительной модели

Очевидным образом, графическая среда редактирования является важной, но не единственной функцией поддержки визуальных языков. Другой важной задачей является трансляция. Можно выделить две основные задачи, связанные с трансляцией – преобразование диаграмм в код (причем под кодом можно понимать как текст, так и, например, другие диаграммы) и обратное

преобразование кода в диаграмму (реинжиниринг). Возникает вопрос возможности трансляции для диаграмм, заданных вычислительной моделью.

Задача универсальной трансляции диаграмм в код существует еще с шестидесятых годов [235]. Можно выделить два основных подхода, используемых в универсальных диаграммерах – применение графовых грамматик и сценариев трансляции. Графовые грамматики – суть обобщение механизма трансляции текстовых языков на случай графов. По аналогии с текстовыми языками, графы представляются в виде терминальных и нетерминальных узлов, между которыми заданны ребра. Продукционные правила представляют собой правила трансформации фрагментов графов в другие фрагменты. Таким образом, используя те же элементарные операции, что и для текстовых языков, можно построить транслятор визуального языка.

К сожалению, графовые грамматики обладают рядом специфических недостатков, затрудняющих их применение. Прежде всего, у диаграмм обычно отсутствует понятие начала и конца. Множество достаточно популярных диаграмм, вообще говоря, все равно с какого места начать читать (ER-модель, диаграмма классов и пр.). Ввиду вышесказанного, к наиболее важным для трансляции классам LL(1) и LR(1) [1, 2] относится сравнительно небольшое количество визуальных языков; напротив, зачастую трансляция визуального языка возможна путем использования только адаптированных алгоритмов Эрлея [105, 122] и Кока-Янгера-Касами [218].

Важнейший альтернативный метод представляет собой описание сценария обхода графа и соответствующей ему процедуры генерации кода. Языки описания таких сценариев бывают достаточно сложны и содержат большое количество операторов [164,244]. Краткая суть такого сценария обычно заключается в переборе объектов, удовлетворяющим критерию отбора, обнаружению связей между объектами, следованию этим связям при обходе и собственно генерации кода.

Для случая трансляции в диаграмму либо реинжиниринга существует проблема, специфичная только для визуальных языков. При автоматической

генерации диаграммы можно восстановить ее смысл (абстрактный граф синтаксиса), однако без пространственного графа отношений диаграмма не может быть нарисована, сам же пространственный граф отношений может быть выбран бесконечным количеством способов. Эта проблема, известная как проблема упорядочивания, имеет большое количество разнообразных решений и по сложности, вообще говоря, едва ли не превышает собственно задачу трансляции. В силу своей самостоятельности эта проблема в данной работе не рассматривается.

Так как вычислительная модель представима в виде семантической сети, то есть графа, то диаграмма, заданная через вычислительную модель, может быть транслирована как с помощью сценариев обхода, так и с помощью графовых грамматик. Абстрактный граф синтаксиса при реинжиниринге может также быть восстановлен с помощью обоих методов. При этом пространственный граф отношений может быть заполнен некоторыми значениями по умолчанию, в дальнейшем преобразованными либо оператором, работающим с программой, либо с помощью какого-либо из автоматических методов. При специфическом виде трансляции – трансляции из одной нотации в другую – для вычислительной модели возможен специфический метод преобразования путем замены графического уровня объектов с сохранением их семантического смысла.

Тем не менее в данной работе будет рассматриваться трансляция на основе навигационных сценариев, как наиболее простая для восприятия человеком. Идея метода состоит в том, что существует некоторый язык, на котором описывается сценарий, в соответствии с которым выбираются те или иные узлы диаграммы и генерируется некоторый код. В виду вышесказанного возникает три основных вопроса, а именно:

- как именно должны выглядеть структуры данных, подготавливаемые диаграммером, для выполнения трансляции;
- каким именно образом может выглядеть этот язык для диаграмм, заданных в вычислительной модели;

- каковы области применимости этого языка.

Естественным подходом к решению данной задачи выглядит использование некоторого базового языка, расширенного возможностями, необходимыми для доступа к модели. В виду того, реализующее вычислительную модель визуального языка программное средство, описанное в разделе 3, выполнено на языке Java, наиболее естественным базовым языком выглядит JavaScript (на базе проекта “Nashorn”), транслятор которого доступен через пакет javax.script. Nashorn также может естественным образом обращаться к коду редактора диаграмм, написанному на Java. Использование этого языка решает сразу большое количество проблем – хорошая документированность языка, понятные возможности, отсутствие необходимости доказательства тьюринг-полноты и пр. Таким образом, необходимо решить только две задачи – определить модель данных, с которой будет работать транслятор, и определить функции доступа к ней.

Очевидным образом, так как внутреннее трехуровневое представление, описанное в подразделе 2.4, является достаточным для отображения диаграмм, оно должно быть достаточным и для их трансляции. Поэтому предлагается рассматривать структуру данных как дерево, в корне которого находится объект-диаграмма, из которой выходят объекты этой диаграммы, каждый из которых может быть либо терминальным объектом (листом дерева) с атрибутами, либо тоже диаграммой. При этом специфические операторы языка будут сведены к операциям получения дочерних объектов для данного объекта, его свойств и атрибутов.

Пример 2.5. В качестве иллюстрации вышесказанного рассмотрим пример ER-диаграммы, изображенной на рис. 2.1. Внутренняя структура данных будет выглядеть так (рис. 2.10):

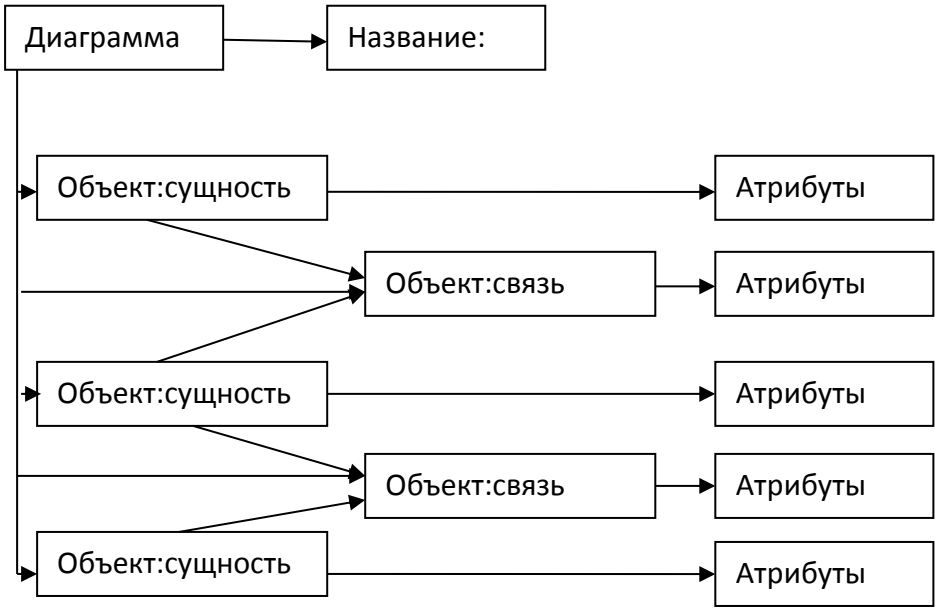


Рис. 2.10. Структура данных диаграммы для трансляции

Соответственно, необходимые операции для обработки этой структуры сводятся к обычным операциям получения доступа к элементам дерева.

Таким образом, трансляция диаграмм, заданных в вычислительной модели визуального языка, возможна обоими основными методами, принятыми в индустрии – через графовые грамматики и сценарии обхода. Использование графовых грамматик является более сложным методом. Так одной из основных мотивация создания вычислительной модели визуального языка является упрощение описания визуальных редакторов пользователем, предлагается в качестве основного метода использовать сценарии обхода дерева, построенного на основе семантической сети, соответствующей вычислительной модели. Таким образом, вычислительная модель визуального языка может описывать визуальный язык, диаграмму на этом языке и использоваться для трансляции диаграммы в текст, то есть является законченной системой, пригодной для описания редакторов диаграмм.

Выводы по разделу 2.

1. В современной индустрии наибольшей популярностью пользуется подход к описанию визуальных языков на основе графовых грамматик. Подход

является весьма сложным и провоцирует разнообразные попытки упрощения, в частности, на основе абстрактного графа синтаксиса и пространственного графа отношений, позволяющих описывать смысл и изображение диаграммы по отдельности.

2. Вычислительная модель визуального языка на основе модифицированных вычислительных моделей Тыугу позволяет описать визуальный язык способом, свободным от вышеуказанного недостатка и дружественным пользователю за счет адаптации широко применяющегося для описания функционирования систем математического аппарата. Модель функционирует в соответствии с алгоритмом пересчета формализованной семантической сети при взаимодействии с ней. Вычислительную модель всегда можно привести к виду, когда вычисления будут приводить к одному и тому же результату независимо от пути. При этом модель пригодна для циклических вычислений при условии введения дополнительных расширений.
3. Графическое изображение диаграммы может быть реализовано путем добавления в семантическую сеть концептов особого вида – графических примитивов. Задача деления диаграммы на объекты также решается добавлением объектов – логических узлов, связанных иерархическими отношениями. Таким образом, семантическая сеть трактует все свои составляющие единообразно и одновременно может представлять себя тремя способами – смысловым наполнением (данными), графически и в виде взаимодействующих объектов. Показано, что подобная сеть поддерживает интерфейс как с пользователем, так и со сторонними приложениями. При этом диаграмма имеет механизмы для динамического изменения без участия пользователя, причем на всех уровнях представления, а также простое решение для декомпозиции логического объекта в другую диаграмму.
4. Визуальный язык может быть определен на основе множества объектов и правил трансформации. При этом каждый объект, в свою очередь, также

является диаграммой, а правила трансформации естественным образом представляются в виде концептов - функций вычислительной модели.

5. Трансляция диаграмм, заданных в вычислительной модели визуального языка, возможна как через графовые грамматики, так и через сценарии обхода графа. С целью снижения сложности определения диаграммного языка пользователем предлагается в качестве основного метода использовать сценарии обхода дерева, построенного на основе вычислительной модели. Таким образом, предложенная модель визуального языка может описывать собственно язык, диаграмму на этом языке и использоваться для трансляции диаграммы в текст, то есть является законченной системой, пригодной для описания редакторов диаграмм.

3. Прикладная вычислительная модель визуального языка и инструментальное средство поддержки прикладных визуальных языков

В соответствии со сформулированной целью работы, рассмотрим возможность применения полученных и изложенных ранее результатов в инструментальном средстве. Вычислительная модель визуального языка была приведена в общем виде. Для решения задачи создания инструментального средства необходимо ее ограничить за счет выбора конкретных поддерживаемых операций, при этом получившееся программное средство не должно накладывать существенные ограничения на визуальные языки, поддержка которых является частью оригинальной задачи, а именно визуальные языки общего и специального назначения, используемые в РКТ. Таким образом, необходимо определить:

- Набор поддерживаемых типов данных;
- Перечень поддерживаемых математических операций и встроенных функций, используемых для установления соотношений внутри семантической сети;
- Перечень и семантику визуальных примитивов.

В этом разделе будет рассмотрено программное средство Diagen, написанное на языке Java, и реализующее редактор диаграмм на основе вычислительных моделей и транслятор, а также прикладная реализация вычислительной модели, использованная в этом редакторе.

3.1. Типы данных и операции над ними

Прикладная вычислительная модель, использованная в программном средстве Diagen, имеет слабую типизацию. Решение о конкретном типе данных результата принимается в процессе вычислений.

Типы данных в прикладной вычислительной модели визуального языка могут быть трех базовых типов – атомарного, спискового и специального типа

“объект”. Кроме того, на основе спискового типа строятся синтетические типы “множество”, “отображение”.

В качестве атомарных типов данных прикладной вычислительной модели выбраны следующие типы:

- логический (Boolean);
- строковый (String);
- числовой (Number) на базе чисел с неограниченной точностью java.lang.BigDecimal.

Для чисел с неограниченной точностью вводится настраиваемая постоянная величина точности при выполнении таких операций, как деление.

Кроме атомарных типов вводится тип – список. Список – упорядоченный набор нуля и более элементов произвольного типа. Таким образом, список может содержать в себе другие списки и элементы различного типа. С помощью очевидных наборов функций список может быть расширен до синтетических типов “множество”, “отображение”.

Дополнительно вводится тип данных “объект”, основным предназначением которого является установление отношения принадлежности между атрибутами и логическим уровнем диаграммы. Необходимость объектного типа проистекает из задачи удаления логически связанных фрагментов из диаграммы – если при вставке можно считать, что одна диаграмма добавляется в другую, то при удалении необходимо уметь определять границы удаляемой области. Объекты семантической сети указывают на иерархию узлов, которые им принадлежат, и могут быть удалены из сети как единое целое.

Для атомарных типов данных вводятся два типа операторов – арифметические и логические, а также набор функций. Для списковых типов дополнительно вводятся специальные списковые операторы. Тип “объект” может быть аргументом в операторах установления отношений принадлежности, а также удаления. Ниже будет приведен перечень всех допустимых операторов для различных типов данных.

Таблица 3.1.

Унарные математические операторы

Оператор	Значение	Результат при типе атрибута входных данных				
		Boolean	String	Number	Список	Объект
1	2	3	4	5	6	7
$initialEvent	Проверка вхождения атрибута в воздействие	Boolean	Boolean	Boolean	Boolean	Boolean
!	Логическое отрицание	Boolean	-	-	-	-
$delete	Удаление логического объекта и всеъ связанных с ним сущностей	-	-	-	-	Без возвращаемого значения
abs	Взятие по модулю	-	-	Number	-	-

Специальный оператор $initialEvent предназначен для определения, является ли ее аргумент частью воздействия на диаграмму. С его помощью в зависимости от воздействия строятся различные пути вычислений.

Специальный оператор $delete предназначен для удаления из модели атрибута типа “объект” со всеми входящими в него атрибутами, в том числе, возможно, и другими атрибутами того же типа.

Таблица 3.2.

Бинарные и n-арные математические операторы

Операция	Значение	Типизация
1	2	3
=	Присвоение значения	Значение слева получает тип значения справа, операция не возвращает результат
==	Значение слева равно значению справа	См. Таблица 3.3
>	Значение слева больше значения справа	См. Таблица 3.3
<	Значение слева меньше значения справа	См. Таблица 3.3
>=	Значение слева больше или равно значению справа	См. Таблица 3.3
<=	Значение слева меньше или равно значению справа	См. Таблица 3.3
&&	Логическое И	Определен только на аргументах и результате типа Boolean
||	Логическое ИЛИ	Определен только на аргументах и результате типа Boolean
^	Логическое исключающее ИЛИ	Определен только на аргументах и результате типа Boolean
+	Сумма значений слева и справа	См. Таблица 3.4
-	Разность значений слева и справа	Определен только на аргументах и результате типа Number
*	Произведение значений слева и справа	Определен только на аргументах и результате типа Number
/	Частное от деления значений слева и справа (с константной точностью)	Определен только на аргументах и результате типа Number
$setchildof	Установление отношения принадлежности атрибута объекту	См. Таблица 3.5

$list	формирует список из своих аргументов	Определен на произвольном натуральном числе аргументов любых типов, результат типа список
$append	Соединяет два списка в один	Оба аргумента и результат типа список
$element	Получает из списка элемент по номеру	Первый аргумент – Number, второй аргумент – список. Тип результата произвольный, зависит от типа возвращаемого элемента списка

Специальный оператор $setchildof устанавливает отношение принадлежности одного атрибута типа “объект” (дочернего) другому (родительскому). После его применения удаление родительского атрибутаоператором $delete приведет к удалению также и дочернего атрибута.

Таблица 3.3.

Динамическая типизация для логических операторов сравнения

Тип левого аргумента	Тип результата при типе правого аргумента				
	Boolean	String	Number	Список	Объект
1	2	3	4	5	6
Boolean	Boolean	Не определен	Не определен	Не определен	Не определен
String	Не определен	Boolean	Не определен	Не определен	Не определен
Number	Не определен	Не определен	Boolean	Не определен	Не определен
Список	Не определен	Не определен	Не определен	Не определен	Не определен
Объект	Не определен	Не определен	Не определен	Не определен	Не определен

Таблица 3.4.

Динамическая типизация для оператора +

Тип левого аргумента	Тип результата при типе правого аргумента				
	Boolean	String	Number	Список	Объект
1	2	3	4	5	6
Boolean	Не определен	Не определен	Не определен	Не определен	Не определен
String	String (конкатенация)	String (конкатенация)	String (конкатенация)	Не определен	Не определен
Number	Не определен	Не определен	Number (сложение)	Не определен	Не определен
Список	Не определен	Не определен	Не определен	Не определен	Не определен
Объект	Не определен	Не определен	Не определен	Не определен	Не определен

Таблица 3.5.

Динамическая типизация для оператора setChildOf

Тип левого аргумента	Тип результата при типе правого аргумента				
	Boolean	String	Number	Список	Объект
1	2	3	4	5	6
Boolean	Не определен	Не определен	Не определен	Не определен	Не определен
String	Не определен	Не определен	Не определен	Не определен	Не определен
Number	Не определен	Не определен	Не определен	Не определен	Не определен
Список	Не определен	Не определен	Не определен	Не определен	Не определен
Объект	Без возвращаемого значения	Без возвращаемого значения	Без возвращаемого значения	Без возвращаемого значения	Без возвращаемого значения

В разделе 4 приводятся результаты применения вычислительной модели визуального языка к некоторым важным диаграммам, используемым в задачах РКТ, и показывается, что приведенный в таблицах 3.1 – 3.5 перечень типов данных и операторов достаточен для реализации редакторов этих диаграмм.

3.2. Визуальные примитивы

Визуальный алфавит прикладной вычислительной модели состоит из следующих визуальных символов: точка приема (Accept Point), активная область (Active Area), круг (Circle), эллипс (Ellipsis), линия (Line), текст (Text). Ниже эти символы будут рассмотрены подробнее

Таблица 3.6.

Визуальные примитивы прикладной вычислительной модели

Изображение и поведение	Атрибут	Значение атрибута
1	2	3
Точка приема (Accept Point)		
Представляет собой область, нажатие в которую мышью в режиме установки связи между объектами приводит к установлению новых соотношений между атрибутами диаграммы.	X	Координата по оси X
	Y	Координата по оси Y
	connectionTokens	Токен, описывающий роли в отношении, в которые может вступать эта точка приема
Активная область		
Прямоугольная область (невидимая), которую можно перемещать мышью.	X	Координата левого верхнего угла по оси X
	Y	Координата левого верхнего угла по оси Y
	width	Ширина
	height	Высота
Круг		

Изображение окружности	X	Координата центра по оси X
	Y	Координата центра по оси Y
	radius	радиус
Эллипс		
Изображение эллипса с осями, направленными горизонтально и вертикально	X	Координата левого верхнего угла по оси X
	Y	Координата левого верхнего угла по оси Y
	radius	Ширина (горизонтальная полуось) эллипса
	radius2	Высота (вертикальная полуось) эллипса
Линия		
Изображение отрезка прямой	X1	Координата начала по оси X
	Y1	Координата начала по оси Y
	X2	Координата конца по оси X
	Y2	Координата конца по оси Y
Текст		
Текст, в зависимости от типа атрибута text либо строка, либо таблица с заголовком, если атрибут text – двумерный список.	X	Координата левого верхнего угла по оси X
	Y	Координата левого верхнего угла по оси Y
	Width	Ширина прямоугольника, в который должен быть вписан текст
	Height	Высота прямоугольника, в который должен быть вписан текст

	xMargin	Расстояние от границы прямоугольника до собственно текста по оси X
	yMargin	Расстояние от границы прямоугольника до собственно текста по оси Y
	measuredHeight	Измеренная высота текста
	measuredWidth	Измеренная ширина текста
	Text	Собственно текст
	Schema	Схема, описывающая заголовки столбцов в случае, если текст это двумерный список

Допустимо расширять набор примитивов путем введения новых, пользовательских примитивов, реализующих определенные в программном средстве интерфейсы. В разделе 4 в рамках моделирования применения вычислительной модели визуального языка к некоторым важным диаграммам, используемым в задачах РКТ, продемонстрировано, что приведенный в таблице 3.6 перечень визуальных примитивов достаточен для реализации редакторов этих диаграмм.

3.3. Математическое ядро редактора диаграмм

Математическое ядро редактора диаграмм представляет собой программную реализацию прикладной вычислительной модели. Ядро состоит из множества атрибутов, транслятора выражений, графа зависимостей, описывающего структуру сети, и встроенного набора визуальных примитивов.

Множество атрибутов фактически представляет собой множество пар вида “внутреннее имя – значение”. Каждый атрибут в модели имеет уникальное внутреннее имя. При помещении в модель нового пользовательского объекта (представляющего собой фрагмент

вычислительной модели), содержащего атрибуты, происходит специальное преобразование, в результате которого имена атрибутов переименовываются в уникальные идентификаторы. Таким образом, даже если пользовательский объект будет помещен в модель дважды, это вызовет создание в модели двух разных объектов с одинаковой внутренней структурой, но с уникальными именами атрибутов. В виду этого при помещении в модель нового соотношения между атрибутами необходимо также выполнить специальное преобразование имен, чтобы отношение было установлено именно на тех атрибутах, внутренним именам которых соответствуют атрибуты, использованные в отношении.

Пример 3.1. Поясним сказанное на примере. Предположим, у нас есть описание пользовательского объекта с именем $Entity$, которому принадлежат два атрибута и одно соотношение $Entity: \{A, B, B = 2A\}$. При вставке двух экземпляров такого пользовательского объекта в диаграмму будут сгенерированы автоматические имена; атрибуты, представляющие пользовательские объекты, получат имена вида $\$0.Entity$ и $\$1.Entity$, исходные атрибуты пользовательских объектов будут называться $\$0.Entity.A$, $\$0.Entity.B$, $\$1.Entity.A$, $\$1.Entity.B$ и два отношения, соответственно, будут представлены как $\$0.Entity.B = 2 * \$0.Entity.A$ и $\$1.Entity.B = 2 * \$1.Entity.A$.

Аналогичный подход используется при установлении связей между атрибутами разных пользовательских объектов. В этом случае в отношения атрибуты поступают в виде псевдонимов, для которых известно, какие внутренние имена они на самом деле представляют; при установлении связи псевдонимы преобразуются во внутренние имена и соответствующие отношения добавляются в вычислительную модель.

Еще одной особенностью множества атрибутов является транзакционность. Транзакция поддерживает три этапа – подготовка, выполнение и фиксация либо откат. На этапе подготовки все измененные атрибуты считаются частью воздействия. Во время выполнения используется

алгоритм 2.1. Если он завершен успешно, то выполняется фиксация значений атрибутов, в противном случае может быть выполнен откат (с помощью журнала отката), в результате чего модель будет приведена в состояние, в котором она находилась до начала транзакции.

Транслятор выражений является простым интерпретатором на основе LR(1) грамматики. Транслятор решает три основные задачи. Во-первых, он вычисляет формулы, заданные в отношениях, и присваивает новые значения переменным. Во-вторых, транслятор разбирает выражения на составляющие и строит обратные зависимости между атрибутами. Эти обратные зависимости в форме “при изменении переменной X необходимо пересчитать Y по формуле F” помещаются в граф зависимостей. Наконец, транслятор осуществляет преобразование имен атрибутов в уникальные идентификаторы.

Граф зависимостей является ключевым средством поддержки алгоритма 2.1. По графу можно установить, какие именно атрибуты какими именно функциями должны быть вычислены в рамках алгоритма. Также граф используется для удаления объектов из диаграммы, так как в нем хранится информация о принадлежности атрибутов и объектов другим объектам.

3.4. Преобразования вычислительной модели визуального языка и интерфейс редактора диаграмм

Вычислительная модель визуального языка является системой с динамической структурой. При взаимодействии с пользователем в нее могут вноситься новые элементы и соотношения либо удаляться старые. Поэтому программное средство, поддерживающее прикладную вычислительную модель визуального языка, должно поддерживать основные операции работы с этой моделью, а именно, добавление и удаление объектов, редактирование их свойств, обработка визуальных примитивов, работа с контекстными меню. Ниже будет дано описание реализации поддержки этих действий.

Описание визуального языка поступает в диаграммер в виде XML-дескриптора, схема которого приведена в приложении П3. Дескриптор

содержит два вида пользовательских объектов – объекты-сущности и объекты–отношения. Для каждого объекта описываются несколько разделов – перечень атрибутов, перечень визуальных примитивов, перечень отношений между атрибутами, перечень ограничительных правил, которые всегда должны быть истинными и перечень команд инициализации объекта. При чтении дескриптора редактор диаграмм создает пункты меню, соответствующие добавлению каждого описанного объекта. Кроме того, для некоторых специальных визуальных примитивов указываются описатели контекстных меню, роли для установления связей и пр.

В зависимости от типа пользовательского объекта его добавление в диаграмму осуществляется различным способом. Объект-сущность добавляется в диаграмму путем копирования структуры в семантическую сеть вычислительной модели и присвоения всем его атрибутам и визуальным примитивам внутренних имен. После добавления выполняется транзакция, на этапе подготовки которой выполняется раздел дескриптора, описывающий инициализацию объекта. Объект-сущность при добавлении в диаграмму не устанавливает связи с другими объектами, но может устанавливать связи с глобальными атрибутами диаграммы, если такие имеются.

Объект-отношение при добавлении в диаграмму может устанавливать связи с другими объектами. Для этого используется ролевая модель. Каждый объект, с которым можно установить отношение, имеет в своем составе специальный визуальный примитив – область установки связи (AcceptArea). Примитив содержит в себе перечень экспортируемых атрибутов основного объекта с псевдонимами и идентификатор, называемый ролью, с помощью которого можно выбирать те области установки связи, которые допустимы для данного объекта-отношения. В описании объекта-отношения указано, какие роли ему необходимы для добавления в диаграмму, поэтому при добавлении такого объекта диаграммер просит указать те области установки связи, которые пользователь хочет для этого объекта использовать. Далее все отношения, заданные на атрибутах объекта-отношения, проверяются

транслятором и имена псевдонимов заменяются на внутренние имена атрибутов, связанных с участвующими в установлении связи объектов. Подробный пример рассмотрен в разделе 4.

Абсолютное большинство действий с объектами может быть выполнено путем вычисления некоторой функции математическим ядром. В частности, такой подход используется для удаления и редактирования свойств объекта. Удаление является очевидной операцией вызова функции $delete. Редактирование – более сложный процесс. Для него используется унифицированный редактор, программируемый через функции математического ядра. Поддерживаются следующие функции (Таблица 3.7).

Таблица 3.7

Функции для программирования универсального редактора

Функция	Значение
1	2
$initDialog	Инициализация диалога
$stringValueToDialog	Добавить в диалог поле редактирования атрибута типа String
$numberValueToDialog	Добавить в диалог поле редактирования атрибута типа Number
$booleanValueToDialog	Добавить в диалог поле редактирования атрибута типа Boolean
$stringTableToDialog	Добавить в диалог таблицу для редактирования двумерного списка
$stringTableToDialog	Разрешить или запретить редактирование в столбце с соответствующим номером
$setListColumn	Для столбца установить список выбора значений
$setBooleanColumn	Для столбца установить выбор значения с помощью чекбокса
$showDialog();	Вывести диалог на экран

Работа с диалогом осуществляется в соответствии с обычной схемой транзакции математического ядра. Атрибуты, измененные в диалоговом окне, считаются начальным воздействием. Далее на их основе пересчитываются значения зависящих от них атрибутов и выполняется фиксация транзакции. Вопрос поддержки зависимостей между атрибутами непосредственно в диалоговом окне является дальнейшим направлением работы.

Наконец, работа с графическими примитивами построена следующим образом. Редактор хранит в себе полный список всех в данный момент зарегистрированных в модели примитивов, при этом каждый примитив может на основании координат и типа действия проинформировать диаграмммер о том, что он может это действие обработать. В случае если пользователь выполняет действия (например, кликнул мышкой), диаграммер опрашивает все визуальные примитивы в поисках того, который может отреагировать на это действие.

Редактор поддерживает действия, перечень которых приведен в таблице 3.8:

Таблица 3.8

Действия пользователя, поддерживаемые графическим редактором

Действие	Визуальный примитив	Действие
1	2	3
Щелчок левой кнопкой мыши в обычном режиме	Активная точка, активная область	Перемещение
Щелчок правой кнопкой мыши в обычном режиме	Активная точка, активная область	Контекстное меню
Щелчок левой кнопкой мыши в режиме установки соединения	Область установки соединения	Выбор роли для соединения

Таким образом, только три типа визуальных примитивов из всего перечня могут обрабатывать действия пользователя. При необходимости обработки других действий возможно расширить визуальный словарь новыми примитивами, обрабатывающими эти действия.

3.5. Транслятор диаграмм

Описание визуального языка средствами, отличными от средств формальных теорий, оперирующих правилами вывода (грамматик, теории категорий, различных исчислений и т.п.), ставит вопрос о том, как можно транслировать диаграмму, для которой отсутствует информация, позволяющая ее синтезировать или разложить в дерево построения. В данной работе предлагается выполнять трансляцию с помощью сценариев обхода семантической сети.

Транслятор состоит из двух частей – модуля экспорта семантической сети в дерево данных и модуля обхода этого дерева. Модуль экспорта имеет очень простую структуру: в соответствии со схемой, приведенной в приложении П4, он экспортирует все объекты, атрибуты и значения в XML-документ. Полученный документ содержит в себе полную информацию, по которой можно восстановить диаграмму (но не язык, поэтому диаграмма не будет поддерживать ряд функций, в частности, добавление и удаление объектов).

Модуль обхода построен на основе простого LL(1) транслятора [1,2]. Транслятор поддерживает простые операторы – получение списка объектов по критерию, взятие значения атрибута, условные операторы и вывод. Полный список операторов приведен в таблице 3.9.

Таблица 3.9

Операторы встроенного транслятора диаграмм

Оператор	Значение
1	2
var	Определение переменной
{}	Блок вычислений
foreach ()	Цикл по списку
selector	Псевдоним списка, по которому осуществляется итерация в текущем foreach
write	Вывод текста в результат
writeln	Вывод текста в результат
If then	Оператор условия
If then else	Оператор условия
getObjects()	Получение списка объектов, соответствующих условию
getAttribute()	Получение значения атрибута объекта по имени
getListElement()	Получение элемента списка по номеру
hasNext()	Признак того, что в списке есть непросмотренные элементы, используется совместно с foreach

Как будет показано в разделе 4, даже столь простой транслятор позволяет получать работоспособный код для некоторых языков.

Открытыми остаются вопросы, связанные с трансляцией диаграмм в диаграммы и реинжинирингом (восстановлению диаграмм по текстам).

Выводы по разделу 3.

1. Прикладная вычислительная модель визуального языка, используемом в программном средстве “Diagen”, определяется как вычислительная модели визуального языка, для которой заданы конечные наборы типов данных и операций над ними, а также перечень базовых визуальных примитивов. Уровень выразительной мощи программного средства может быть

увеличен за счет расширения типов данных синтетическими типами и функциями на языке Javascript.

2. Создана программная реализация математического ядра, реализующего прикладную вычислительную модель визуального языка, и его основные функции – построение графа зависимостей, поддержка транзакций, описан способ реализации алгоритма 2.1.
3. Предложен и описан новый способ интерфейсной реализации основных задач по работе с диаграммой – добавления и удаления объектов, связей между объектами, работа с контекстными меню, редактирование свойств логического объекта диаграммы. Показано, что действия пользователя с диаграммой могут обрабатываться с помощью специальных визуальных примитивов, предназначенных исключительно для обработки этих действий; добавлять новые обрабатываемые действия можно за счет создания новых специализированных примитивов.
4. Предложена и описана структура и основные алгоритмы работы транслятора, преобразующего диаграммы в текст в соответствии со сценариями трансляции.
5. Доказано (путем моделирования некоторых важных классов визуальных языков в разделе 4), что вышеперечисленный перечень возможностей программного средства полон и программное средство позволяет реализовать основные визуальные языки, используемые в отрасли информационных технологий. В качестве недостатков средства можно отметить отсутствие проработанных алгоритмов реинжиниринга диаграмм (в частности, восстановления диаграммы по сгенерированному коду) и трансляции диаграмм в диаграммы. Эти вопросы являются направлениями дальнейшей работы.

4. Результаты применения модели в задачах оценивания состояний объектов РКТ

В данном разделе будет описан опыт применения вычислительной модели визуального языка к реальным классам задач и сделанные по результатам выводы о применимости модели, необходимых доработках и перспективных направлениях развития. В качестве задач будут рассмотрены две – задача создание редактора ER-диаграмм, относящаяся к обеспечению импортозамещения, и задача создания мнемосхем элементов ракетно-космической техники при контроле их состояния. Несмотря на то, что обе задачи относятся к разработке визуальных языков, между ними имеются существенные различия, а именно:

- элементы ER-диаграммы обладают существенно более сложным визуальным поведением, в том числе взаимодействует с пользователем через диалоговые окна, в то время как визуальное поведение мнемосхемы в основном ограничено изменением значений индикаторов;
- ER-диаграмма может конструироваться пользователем, в то время как мнемосхема всегда имеет одинаковую структуру;
- элементы ER-диаграммы имеют очень простые контуры (линии, прямоугольники), в то время как мнемосхема состоит из элементов разнообразной формы (окружности, треугольники, закрашенные области, змеевики и пр.);
- ER-диаграмму необходимо транслировать;
- мнемосхема должна интегрироваться с внешними процессами, передающими телеметрическую информацию.

Тем не менее, несмотря на вышеописанные различия, разработанная вычислительная модель визуального языка показала свою применимость к обеим задачам. В подразделе 4.1 будут приведены результаты описания ER-

диаграмм, а в подразделе 4.2 – мнемосхемы тракта наддува баков ракеты-носителя “Союз-2”.

4.1. Решение задачи поддержки общеупотребительных диаграмм в условиях импортозамещения в РКТ на примере ER-диаграмм

4.1.1. Обоснование выбора ER-модели и постановка задачи

В сфере РПО, в том числе, связанной с РКТ, используется большое количество разнообразных визуальных языков. В первую очередь следует отметить три основных группы таких языков – 15 языков нотации IDEF [54], 12 языков нотации UML [3], а также схемы алгоритмов [6]. В зависимости от ситуации, могут использоваться и другие визуальные языки, однако перечисленные типы покрывают основную часть нужд проектировщиков.

Рассматривая вышеописанные диаграммы, можно заметить, что они делятся на два основных класса – описание структуры (например, IDEF1 или диаграмма классов) и описание поведения (например, диаграмма последовательностей, диаграмма действий, схема алгоритма). При этом диаграммы, описывающие структуру, а также некоторые диаграммы, описывающие поведение (например, диаграмма вариантов использования) обычно строятся по единому шаблону – условные прямоугольники, соединенные связями. В данном разделе будет приведена реализация редактора такой диаграммы средствами вычислительной модели на примере диаграммы сущность-связь или ER-диаграммы (IDEF1). Данная диаграмма является крайне важной для РПО, так как является основным средством проектирования схем баз данных.

Полноценный графический редактор ER-диаграмм, такой как Power Designer [8] или ER Win [21], является сложной программной системой, помимо собственно отображения диаграмм, выполняющий множество других функций. Для демонстрации применимости вычислительной модели

визуального языка достаточно реализовать основную функциональность, связанную с рисованием диаграмм и трансляцией в SQL-код. Редактор будет реализован со следующими функциями:

1 отображение диаграммы, добавление, удаление, редактирование свойств сущностей;
2 связывание сущностей и разрыв связей, установка внешних ключей;
3 трансляция полученной диаграммы в SQL код в стандарте ANSI.

Полученный редактор будет обладать базовым функционалом, позволяющим создавать диаграммы и транслировать их в код. Вопросы, связанные с реинжинирингом баз данных, исполнением полученного кода, анализом на корректность, генерацией тестовых данных и пр. выходят за рамки задачи этого раздела и здесь рассматриваться не будут. Кроме того, язык будет реализован в виде, приближенном к графическому редактору Power Designer (рис.4.1), несколько отличающимся от IDEF1.

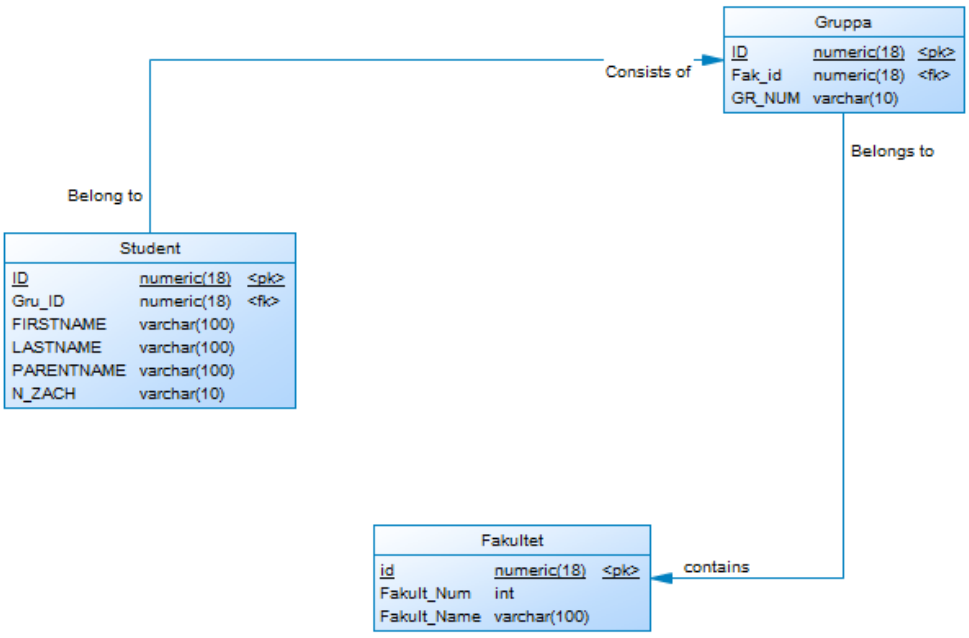


Рис. 4.1. Пример ER-диаграммы, изображенной средствами Power Designer.

4.1.2. Реализация визуальной части объекта «сущность»

Сущность – объект диаграммы, представляющий реляционное отношение, или, в приближенном представлении, таблицу базы данных. Содержательно он состоит из описания имен, типов данных и других свойств атрибутов реляционного отношения, а также признаков внешнего ключа. Рассмотрим отдельно визуальную составляющую объекта “сущность”.

Визуальная часть объекта “сущность” строится из следующих частей (рис. 4.2):

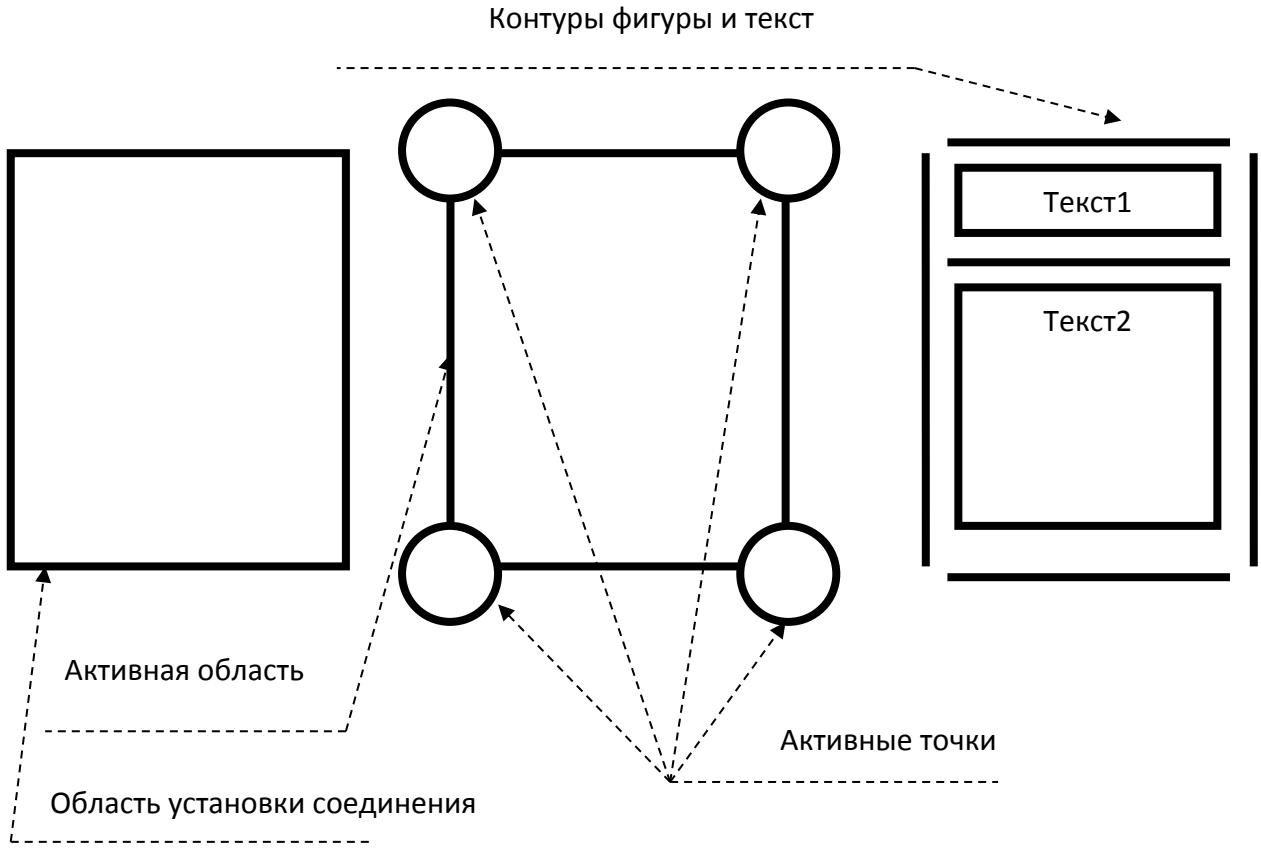


Рис. 4.2. Визуальная структура объекта “сущность”

Таким образом, сущность состоит из шести управляющих элементов, которые отображаются или нет в зависимости от ситуации, и семи элементов, представляющих собственно информацию. В управляющие элементы входят область установки соединения, предназначенная для установки связи между сущностями; четыре активные точки, с помощью которых осуществляется изменение размера фигуры; активная область, за которую фигура перемещается. Элементы, отображающие информацию, состоят из пяти

отрезков прямых, образующих контур фигуры и двух текстовых полей – верхнее представляет название реляционного отношения, а нижнее – таблицу атрибутов.

Рассмотрим систему уравнений, описывающую данный объект. Остовом всего изображения являются пять управляющих объектов – четыре активные точки и активная область (изображены в центральной части (рис. 4.2)). Рассмотрим подробно эти объекты и их атрибуты (рис.4.3).

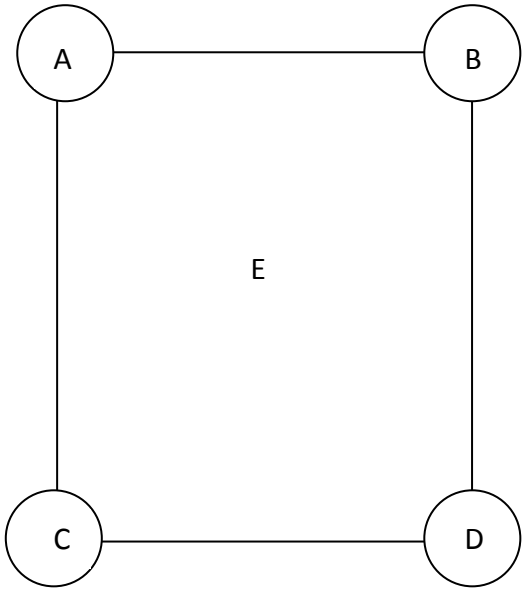


Рис. 4.3. Остов объекта “сущность”

Четыре активные точки обозначаются именами A, B, C и D, а активная область – E. У них имеются следующие атрибуты:

$$\begin{aligned} A &= \{x, y\}; \\ B &= \{x, y\}; \\ C &= \{x, y\}; \\ D &= \{x, y\}; \\ E &= \{x, y, w, h\}. \end{aligned} \qquad (4.1)$$

Для активных точек атрибуты обозначают позицию их центра, для активной области атрибуты X и Y задают координаты левого верхнего угла, а W и H – соответственно, ширину и высоту. Следующая система уравнений описывает остов:

$$C.x = A.x; \quad (4.2)$$

$$A.x = C.x; \quad (4.3)$$

$$A.x = E.x; \quad (4.4)$$

$$E.x = A.x; \quad (4.5)$$

$$B.x = D.x; \quad (4.6)$$

$$D.x = B.x; \quad (4.7)$$

$$\mathrm{cond}(!\,\$\mathrm{initialEvent}(E.x))E.w = B.x - E.x; \quad (4.8)$$

$$\mathrm{cond}(\$\mathrm{initialEvent}(E.x))B.x = E.x + E.w; \quad (4.9)$$

$$B.y = A.y; \quad (4.10)$$

$$A.y = B.y; \quad (4.11)$$

$$A.y = E.y; \quad (4.12)$$

$$E.y = A.y; \quad (4.13)$$

$$C.y = D.y; \quad (4.14)$$

$$D.y = C.y; \quad (4.15)$$

$$\mathrm{cond}(!\,\$\mathrm{initialEvent}(E.y))E.h = C.y - E.y; \quad (4.16)$$

$$\mathrm{cond}(\$\mathrm{initialEvent}(E.y))C.y = E.y + E.h. \quad (4.17)$$

Семантическая сеть состоит из двух несвязанных фрагментов (4.2-4.9) и (4.10-4.17). Эта система в виде семантической сети изображена на рис.4.4.

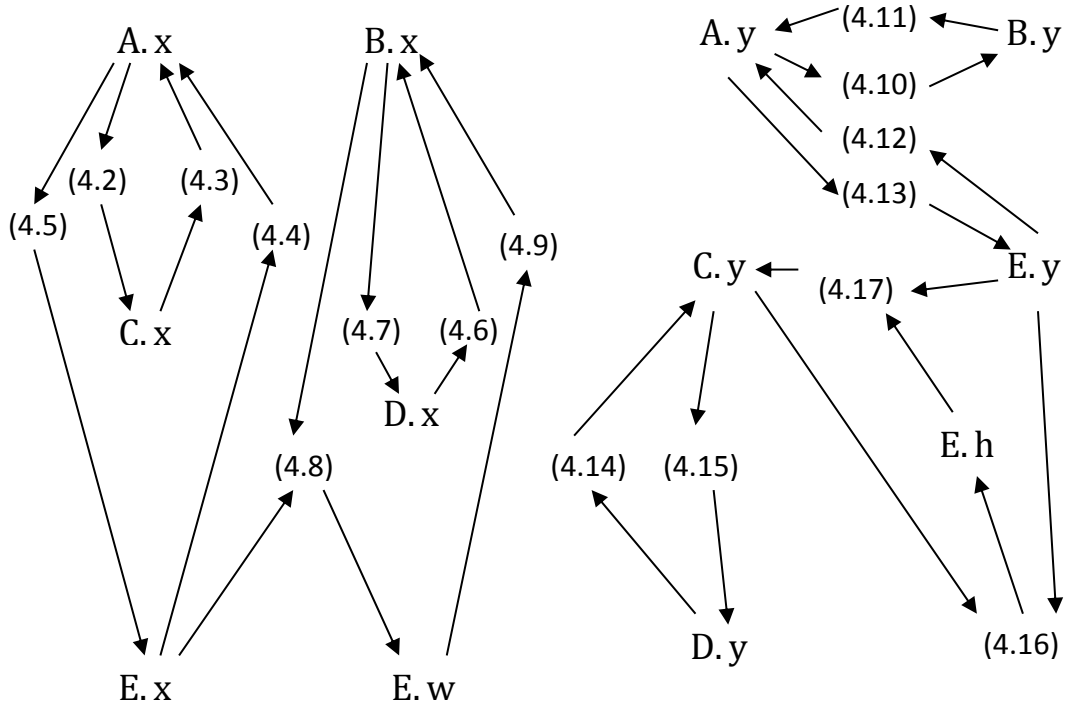


Рис. 4.4. Семантическая сеть, описывающая остов объекта “сущность”

Сеть, изображенная на рис.4.4, представляет собой двудольный граф, узлами которого являются либо атрибуты, либо функции, их вычисляющие. Дуга, выходящая из атрибута и входящая в функцию, представляет входной параметр функции, а дуга, идущая в обратном направлении – присвоение результата. Данная структура практически идентична описанному в подразделе 3.3 графу зависимостей.

С данной сетью можно взаимодействовать пятью различными способами – через активные точки A.B.C.D и через активную область E. Рассмотрим результат выполнения алгоритма 2.1 для каждого из способов.

Первое из изменений – это перемещение угла A, то есть задание новых значений для A.x и A.y. Возникающие возможные последовательности (пути в графе на рис. 4.4) изображены на рис.4.5. Здесь стрелки указывают, какой атрибут изменяется после изменения предыдущего атрибута, а надпись над стрелкой указывает формулу, по которой он вычисляется.

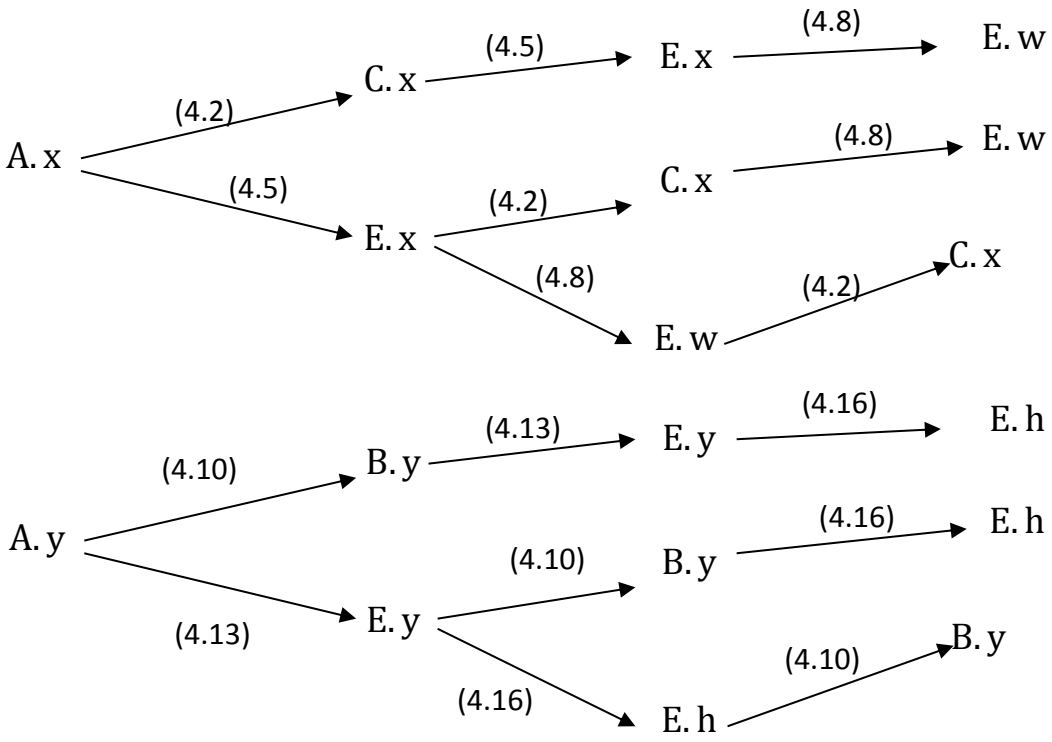


Рис.4.5. Возможные последовательности вычислений при перемещении вершины A.

Как можно увидеть из рис. 4.5, вычисления представляют собой две несвязанные друг с другом последовательности – абсциссы вычисляются отдельно от ординат. Все возможные последовательности приводят к тому, что перемещаются абсцисса угла C и ордината угла B, то есть объект меняет свой размер в соответствии с перемещением угла A; при этом левый верхний угол активной области E устанавливается в те же координаты, что и угол A, а ширина и высота пересчитываются как разница между координатами изменившегося угла A и постоянного угла D. Таким образом, при перемещении угла A объект меняет свой размер, а активная область по-прежнему занимает все пространство объекта.

Аналогичные изменения имеют место и при перемещении вершин B, C и D (рис. 4.6 – 4.9).

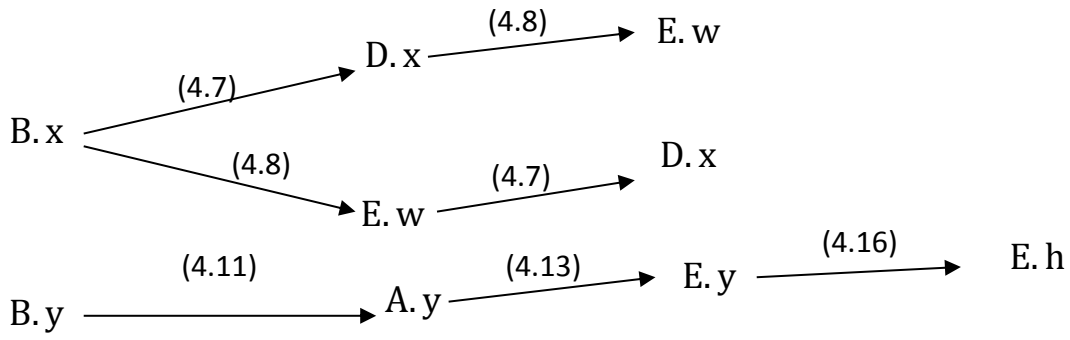


Рис. 4.6. Возможные последовательности вычислений при перемещении вершины B.

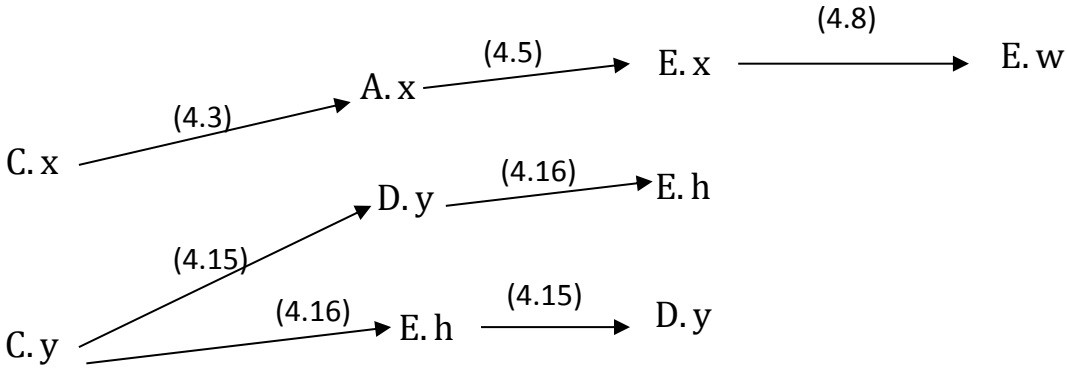


Рис. 4.7. Возможные последовательности вычислений при перемещении вершины C.

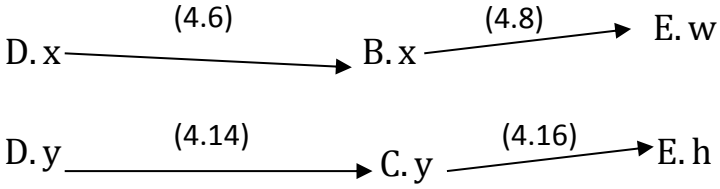


Рис.4.8. Возможные последовательности вычислений при перемещении вершины D.

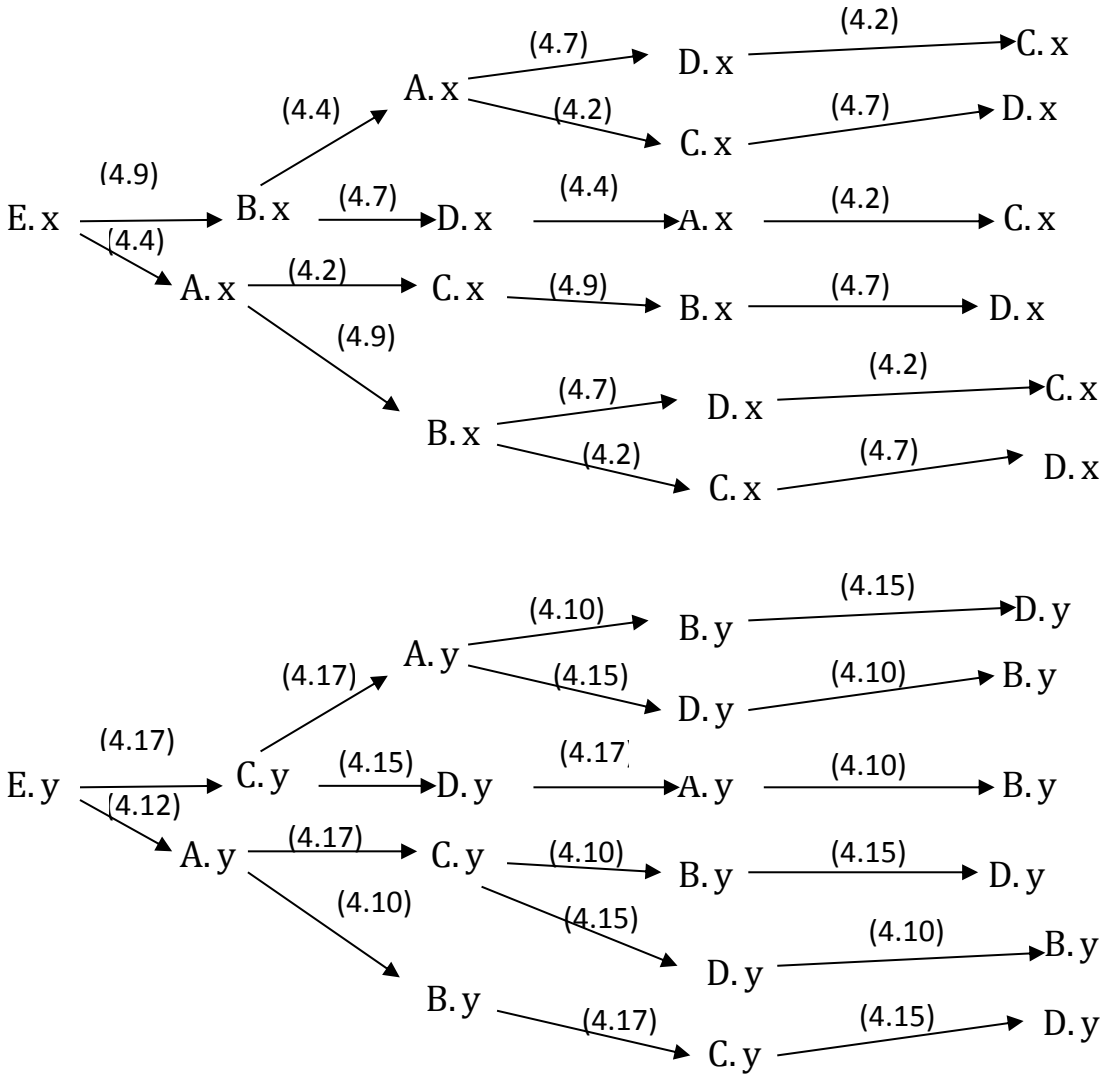


Рис. 4.9. Возможные последовательности вычислений при перемещении вершины E.

Можно легко убедиться, что вне зависимости от выбранного пути вычислений при каждом изменении результат получается одинаковым. Это достигается тем, что всегда сохраняется определенный порядок применения формул, например, на рис. 4.4 ни в какой последовательности невозможно выполнить (4.8) раньше, чем (4.5).

Объект “сущность”, как показано на рис. 4.2, кроме остова, содержит еще большое количество других визуальных элементов. Атрибуты всех этих элементов могут быть вычислены непосредственно как функция от одного из атрибутов остова, потому правильный порядок вычислений никаким образом не может быть нарушен.

Рассмотрим графическую компоненту объекта “сущность” (рис. 4.10.)

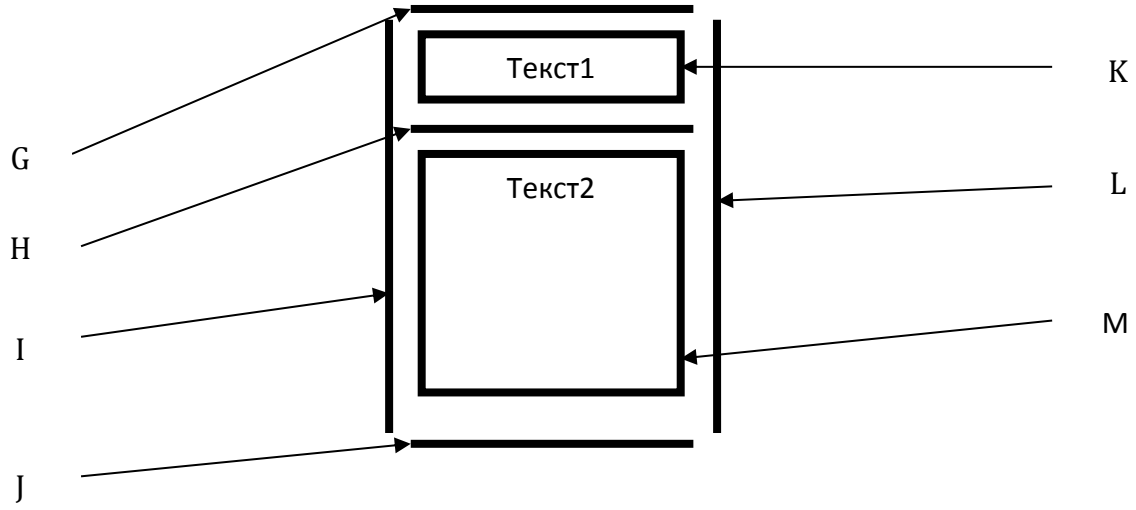


Рис. 4.10. Графическая компонента объекта “сущность”

Графическая компонента состоит из трех горизонтальных линий G,H,J, двух вертикальных I и L и двух текстовых полей K и M.

$$\begin{gathered} G = \{x_1, y_1, x_2, y_2\}; \\ H = \{x_1, y_1, x_2, y_2\}; \\ J = \{x_1, y_1, x_2, y_2\}; \\ I = \{x_1, y_1, x_2, y_2\}; \\ L = \{x_1, y_1, x_2, y_2\}; \\ K = \{x, y, w, h, t\}; \\ M = \{x, y, w, h, t\}. \end{gathered} \tag{4.18}$$

Линии содержат четыре атрибута, попарно представляющие координаты каждого из концов. Текстовые поля содержат семь атрибутов, из которых (x, y) представляют собой координаты левого верхнего угла, (w, h) – ширина и высота и t - собственно текст.

Координаты контура, составленного из графических элементов, подчиняются следующим простым соотношениям:

$$G.x_1 = A.x; \qquad G.y_1 = A.y; \tag{4.19}$$

$$G.x_2 = B.x; \qquad G.y_2 = B.y; \tag{4.20}$$

$$I.x_1 = A.x; \qquad I.y_1 = A.y; \tag{4.21}$$

$$I.x_2 = C.x; \qquad I.y_2 = C.y; \tag{4.22}$$

$$J.x_1 = C.x; \qquad J.y_1 = C.y; \qquad (4.23)$$

$$J.x_2 = D.x; \qquad J.y_2 = D.y; \qquad (4.24)$$

$$L.x_1 = B.x; \qquad L.y_1 = B.y; \qquad (4.25)$$

$$L.x_2 = D.x; \qquad L.y_2 = D.y. \qquad (4.26)$$

Как можно видеть, в виду того, что все зависимости в правой части имеют один из атрибутов остова, нет никакой возможности получить различные результаты при любом порядке вычислений алгоритмом 2.1.

Пусть c - некоторая константа, указывающая на высоту области заголовка. Высота заголовка сделана константной, а не как функция относительно размеров шрифта, по нескольким причинам. Во-первых, в случае пустого заголовка высота должна быть ненулевой. Во-вторых, вычисление высоты заголовка как функции от размеров текста свяжет вычислительную модель с параметрами конкретного графического устройства (без которого высоту текста вычислить невозможно), что приведет к нарушению паттерна MVC и невозможности изображения диаграммы более чем на одном устройстве одновременно.

Оставшаяся внутренняя визуальная структура описывается следующей системой соотношений:

$$K.x = A.x; \qquad K.y = A.y; \qquad (4.27)$$

$$K.w = E.w; \qquad K.h = c; \qquad (4.28)$$

$$H.x_1 = A.x; \qquad H.y_1 = A.y + \text{c}; \qquad (4.29)$$

$$H.x_2 = B.x; \qquad H.y_2 = H.y_1; \qquad (4.30)$$

$$M.x_1 = H.x_1; \qquad M.y_1 = H.y_1; \qquad (4.31)$$

$$M.x_2 = H.x_2; \qquad M.y_2 = H.y_2; \qquad (4.32)$$

$$M.w = K.w; \qquad M.h = E.w - \text{c}. \qquad (4.33)$$

Результат вычисления системы алгоритмом 2.1 не зависит от выбора последовательности вычислений в виду того, что в правой части соотношений находится не более одного атрибута остова.

Область установки соединения по координатам совпадает с объектом E и потому ее реализация очевидна.

Таким образом, вычислительная модель визуального языка позволяет описать объект “сущность”. Для завершения описания диаграммы необходимо дополнить это описание описанием объекта “связь”.

4.1.3. Реализация визуальной части объекта «связь»

Связь – объект диаграммы, представляющий внешние ключи реляционного отношения. Семантически он состоит из перечня соответствий между первичными ключами одного реляционного отношения и атрибутами другого реляционного отношения. Кроме того, для связи могут выделяться дополнительные свойства, в частности, имена ролей связи для каждой таблицы. Рассмотрим отдельно визуальную составляющую объекта “связь”.

Визуальная часть объекта “связь” представляет собой стрелку с изломами, с каждой из сторон которой находятся имена ролей. Пример связей в синтаксисе Power Designer изображен на рис.4.1. В версии этого редактора связь может иметь произвольное (или очень большое) количество изломов. С точки зрения вычислительной модели визуального языка каждый излом добавляет существенное количество дополнительных соотношений, поэтому было принято решение иметь не более трех изломов, что должно быть достаточно для большинства задач.

Объект “связь” является более сложным, чем объекта “сущность”. Его визуальная структура представлена на рис.4.11.

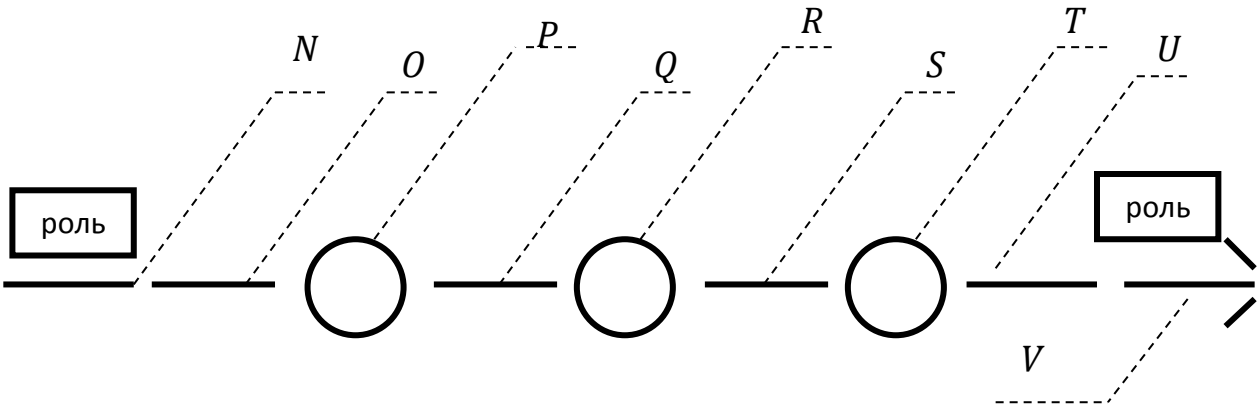


Рис. 4.11. Графическая структура объекта “связь”.

Объект состоит из шести отрезков, которые могут быть только строго горизонтальными или вертикальными, образующих собственно изломанную линию, двух отрезков, изображающих стрелку и трех активных точек в местах излома, а также двух текстовых полей над концами связи. В виду очевидности реализации стрелка и роли рассматриваться далее не будут. Отрезков шесть, а не четыре, ввиду необходимости реализации двенадцати вариантов вхождения связи в сущность с каждой из сторон, которые образуются в зависимости от взаимного расположения сущности и ближайшей активной точки (рис. 4.12). Обработка этих двенадцати вариантов составляет большую часть логики описания объекта “связь”.

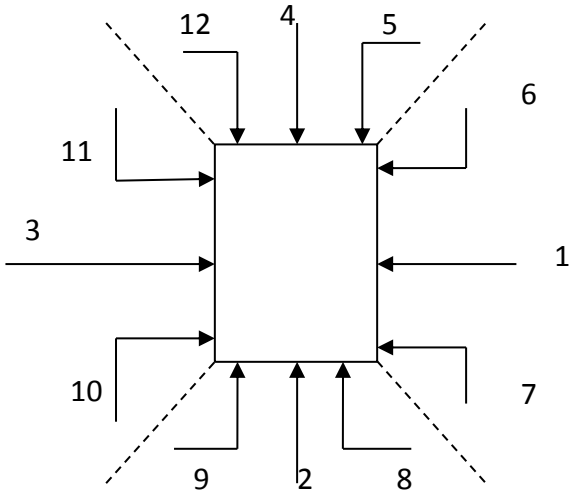


Рис. 4.12. Двенадцать вариантов вхождения связи в сущность.

Рассмотрим реализацию графической логики объекта “связь”. Для установки связи необходимо иметь ссылки на два объекта “сущность”, между

которыми эта связь устанавливается (потенциально это может быть один и тот же объект). Для этой цели каждый объект “сущность” экспортирует свою активную область (E). Будем обозначать экспортируемую активную область сущности, из которой выходит связь, как E_1, а той, в которую связь входит – как E_2.

Рассмотрим входящее соединение сущности и связи. Выбор конкретного способа соединения, как видно на рис. 4.12, зависит от взаимного расположения активной точки T и объекта E_2. Для краткости рассмотрим только один из двенадцати вариантов – связь входит в объект справа налево (рис.4.13).

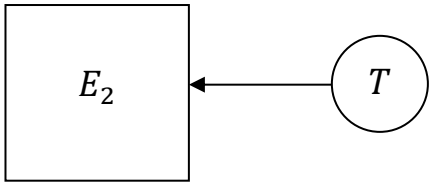


Рис. 4.13. Вариант вхождения связи в сущность номер 1.

$$\mathrm{cond}\big((\mathrm{T.x} > \mathrm{E_2.x} + \mathrm{E_2.w}) \wedge (\mathrm{T.y} > \mathrm{E_2.y}) \wedge (\mathrm{T.y} < \mathrm{E_2.y} + \mathrm{E_2.h})\big) \quad \mathrm{V.x_1} = \mathrm{E_2.x} + \mathrm{E_2.w}; \tag{4.34}$$

$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \quad V.y_1 = T.y; \tag{4.35}$$

$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \quad V.x_2 = T.x; \tag{4.36}$$

$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \quad V.y_2 = T.y; \tag{4.37}$$

$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \quad U.x_1 = E_2.x + E_2.w; \tag{4.38}$$

$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \quad U.y_1 = T.y; \tag{4.39}$$

$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \tag{4.40}$$

$$U.x_2 = T.x;$$
$$cond\big((T.x > E_2.x + E_2.w) \wedge (T.y > E_2.y) \wedge (T.y < E_2.y + E_2.h)\big) \quad (4.41)$$
$$U.y_2 = T.y.$$

Как можно видеть, в варианте 1 (так же, как в вариантах 2, 3 и 4) объекты U и V накладываются друг на друга. Два объекта рисуются один вертикально, другой горизонтально, при выборе вариантов с 5 по 12.

Остальная графическая логика объекта “связь” очевидна, полный текст определения приведен в приложении П5.

Таким образом, объект “связь” по своей структуре является более простым, хотя и содержит значительно большее количество соотношений. Это связано с реализацией большого количества вариантов взаиморасположения сущностей и связей.

4.1.4. Невизуальные атрибуты и сценарий трансляции

Еще одной важной задачей является трансляция диаграмм. В случае модели “сущность-связь” диаграмма может быть транслирована путем последовательного перебора сущностей и связей, что выгодно отличает ее, например, от схем алгоритмов. Для этой цели объекты “сущность” и “связь” должны содержать некоторые атрибуты, не связанные или не напрямую связанные с их визуальным представлением. Прежде всего, эти атрибуты касаются установления соответствия между внешними и первичными ключами.

Основная сложность реализации внешних ключей диаграммы “сущность-связь” заключается в необходимости демонстрации модификатора <fk> для внешних ключей. Так как модель не имеет отношения агрегации, то невозможно установить соотношение между некоторым списком в объекте “сущность” и атрибутами всех объектов “связь”, которые семантически в него входят. Эта проблема решена путем введения предположения, что за одно

изменение диаграммы невозможно поменять атрибуты более чем одного объекта “связь” (в случае удаления нескольких сущностей или связей эти действия должны выполняться строго последовательно, по одному). Функции, реализованные на языке JavaScript, рассматривают один из атрибутов объекта “сущность” спискового типа как отображение и при каждом изменении атрибутов некоторого семантически входящего в него объекта “связь” перезаписывают в нем элементы, относящиеся к этому объекту “связь”. В дальнейшем на основании этого отображения и списка атрибутов сущности генерируется наполнение для текстового поля с пометками <fk>. Таким образом, отношение агрегации моделируется с помощью других соотношений.

Невизуальная логика объекта “связь” заключается в получении списка атрибутов из объектов “сущность”, на которых она установлена, выделении ключевых атрибутов в приемнике и сопоставлении их со списком атрибутов источника. Данная логика реализована в виде фильтрующих списки функций на языке JavaScript. Таким образом, для реализации трансляции в объекте “связь” предусмотрен атрибут, по которому можно установить связь между внешними ключами одной сущности и первичными ключами другой сущности.

Рассмотрим следующий алгоритм.

Алгоритм 4.1. Трансляция диаграммы сущность-связь в код на языке SQL

Шаг 1. Открыть итератор 1 по сущностям;

Шаг 2. Взять из итератора 1 новую сущность;

Шаг 3. Вывести в поток вывода текст “CREATE TABLE”;

Шаг 4. Вывести в поток вывода имя текущей сущности;

Шаг 5. Вывести в поток вывода текст “(”;

Шаг 6. Открыть итератор 2 по списку атрибутов сущности;

Шаг 7. Взять новый элемент из итератора 2 как текущий атрибут сущности;

Шаг 8. Вывести в поток имя поля, тип данных и модификатор из текущего атрибута сущности в синтаксисе SQL;

Шаг 9. Если итератор 2 не пуст, перейти на шаг 7;

Шаг 10. Вывести в поток вывода текст “)”;

Шаг 11. Вывести в поток вывода текст “go”;

Шаг 12. Если итератор 1 не пуст, перейти на шаг 2;

Шаг 13. Открыть итератор 1 по сущностям;

Шаг 14. Взять из итератора 1 новую сущность;

Шаг 15. Открыть итератор 2 по списку атрибутов сущности;

Шаг 16. Взять новый элемент из итератора 2 как текущий атрибут сущности;

Шаг 17. Если элемент содержит признак первичного ключа, закрыть итератор 2 и перейти на шаг 20;

Шаг 18. Если итератор 2 не пуст, перейти на шаг 16;

Шаг 19. Перейти на шаг 31;

Шаг 20. Вывести в поток вывода текст “ALTER TABLE ”;

Шаг 21. Вывести в поток вывода имя сущности;

Шаг 22. Вывести в поток вывода текст “ADD CONSTRAINT PK_”;

Шаг 23. Вывести в поток вывода имя сущности;

Шаг 24. Вывести в поток вывода текст “ PRIMARY KEY(”;

Шаг 25. Открыть итератор 2 по списку атрибутов сущности;

Шаг 26. Взять новый элемент из итератора 2 как текущий атрибут сущности;

Шаг 27. Если элемент содержит признак первичного ключа, вывести его имя в поток вывода (для второго и далее ключевого атрибута добавить запятую);

Шаг 28. Если итератор 2 не пуст, перейти на шаг 26;

Шаг 29. Вывести в поток вывода текст “)”;

Шаг 30. Вывести в поток вывода текст “go”;

Шаг 31. Если итератор 1 не пуст, перейти на шаг 14;

Шаг 32. Открыть итератор 1 по связям;

Шаг 33. Взять из итератора 1 новую связь;

Шаг 34. Вывести в поток вывода текст “ALTER TABLE”;

Шаг 35. Вывести в поток вывода имя сущности, из которой выходит связь;

Шаг 36. Вывести в поток вывода текст “ADD CONSTRAINT PK_”;

Шаг 37. Вывести в поток вывода имя связи;

Шаг 38. Вывести в поток вывода текст “ FOREIGN KEY(”;

Шаг 39. Открыть итератор 2 по списку атрибутов связи;

Шаг 40. Взять новый элемент из итератора 2 как текущий атрибут связи;

Шаг 41. Вывести в поток вывода имя текущего атрибута связи, для второго и далее добавить запятую;

Шаг 42. Если итератор 2 не пуст, перейти на шаг 40;

Шаг 43. Вывести в поток вывода текст “)”;

Шаг 44. Вывести в поток вывода текст “ REFERENCES ”;

Шаг 45. Вывести в поток вывода имя сущности, в которую входит связь;

Шаг 46. Вывести в поток вывода текст “ (”;

Шаг 47. Открыть итератор 2 по списку атрибутов связи;

Шаг 48. Взять новый элемент из итератора 2 как текущий атрибут связи;

Шаг 49. Вывести в поток вывода имя внешнего ключа текущего атрибута связи, для второго и далее добавить запятую;

Шаг 50. Если итератор 2 не пуст, перейти на шаг 47;

Шаг 51. Вывести в поток вывода текст “)”;

Шаг 52. Вывести в поток вывода текст “go”;

Шаг 53. Если итератор 1 не пуст, перейти на шаг 33;

Шаг 54. Конец алгоритма.

Технически алгоритм состоит из трех частей. Шаги 1-12 определяют каркас таблиц. Шаги 13-31 описывают первичные ключи. Шаги 30-53 описывают внешние ключи. Таким образом, в результате выполнения алгоритма 4.1 получается описание схемы базы данных на языке SQL.

Полный текст описания сущности с кодами на JavaScript и сценариями диалоговых окон редактирования атрибутов приведен в приложении П5. Внешний вид диаграммы с рис. 4.1 и результат ее трансляции алгоритмом 2.1 приведен в приложении П6.

4.2. Решение задачи графического представления телеметрической информации при контроле технического состояния объектов ракетно-космической техники

4.2.1. Обоснование выбора моделируемой подсистемы и постановка задачи

Как уже было сказано в разделе 1, помимо задач проектирования программного обеспечения, рассмотренных в предыдущем разделе, при разработке, испытаниях и эксплуатации ракетно-космической техники возникают и узкоспециальные задачи, и одной из них является разработка мнемосхем для выполнения контроля технического состояния различных подсистем [23]. Таким образом, возникает задача программирования мнемосхем и отображения на них телеметрической информации. Указанная задача требует программирования и, таким образом, требует от инженера другой квалификации, нежели контроль технического состояния космического аппарата. Естественным следствием вышесказанного является необходимость отделить программирование мнемосхем и отображение телеметрической информации в отдельную задачу и, по возможности, использовать для них универсальный построитель. Возможность использования для решения этой задачи редактора-конструктора диаграмм на основе вычислительной модели визуального языка, описанного в разделе 3,

будет продемонстрирована на основе одной из подсистем ракеты-носителя “Союз-2”, а именно мнемосхемы тракта наддува топливных баков [23].

Тракт представляет собой систему, обеспечивающую избыточное давление топлива на входе в насосы. Он состоит из большого числа разнообразных элементов, таких как баки азота и пероксида водорода, расходные магистрали, насосы, клапаны, испаритель, магистрали с обратными клапанами, гидравлические редукторы давления и др. Для контроля технического состояния системы используются шесть основных параметров, передаваемых в составе телеметрической информации, имеющих следующие мнемоники: ВПР (давление воздуха после редуктора точной настройки), ВВР (угол поворота привода импульсного регулятора скорости), ДБА (давление в баке азота), ППР (давление пероксида водорода после редуктора), ТНА (число оборотов турбины), ДТО (давление азота на выходе из теплообменника). Числовые значения этих параметров должны отображаться на мнемосхеме.

4.2.2. Реализация мнемосхемы тракта наддува топливных баков

У обозначенной выше задачи реализации мнемосхемы тракта наддува топливных баков, по сравнению с предыдущей рассматриваемой задачей – созданием редактора ER-моделей, можно отметить следующие основные отличия:

- тракт наддува баков содержит существенно большее разнообразие элементов, нежели ER-диаграмма; это приводит к необходимости описания большого количества вариантов поведения при взаимодействии с пользователем для каждого элемента;
- тракт наддува баков обладает статической структурой, т.е. ожидается, что мнемосхема каждый раз при работе имеет одинаковую структуру. При этом представляется нецелесообразным выполнять весь тракт в виде одного

объекта, так как желательно сохранить ряд возможностей, в частности, возможность перемещения элементов.

Ввиду упомянутых особенностей, для упрощения описания системы было принято решение не поддерживать функцию рисования тракта наддува баков пользователем, а генерировать его специальным набором команд. Были определены дополнительные функции, расширяющие основной функционал языка. Перечень функций представлен в таблице 4.1.

Таблица 4.1

Имя	параметры	Возвращаемое значение	Семантика
1	2	3	4
$define	1й параметр – имя определяемой переменной (строка). 2й параметр – тип определяемой переменной (строка)	Без возвращаемого значения	Определение переменной модели
$insertObject	1й параметр – имя класса объекта. 2й параметр – координата X вставки. 3й параметр – координата Y вставки	Значение типа Vertex (логический узел), соответствующее вставленному объекту.	Вставка объекта в модель

$insertConnection	1й параметр – имя класса соединения Далее переменное число параметров, кратное трем. 1й параметр в тройке – объект. 2й параметр в тройке – имя активной области соединения. 3й параметр в тройке – имя роли	Без возвращаемого значения	Вставка соединения в модель
$deref	1й параметр – логический узел 2й параметр – имя атрибута (строка) 31 параметр – новое значение	Без возвращаемого значения	Присваивает атрибуту логического узла новое значение

Последовательное выполнение сценариев на основе этих операторов позволяет создавать диаграммы произвольной сложности с заранее определенной структурой. В рамках приведенной задачи создан сценарий, выполнение которого приводит к созданию мнемосхемы тракта наддува топливных баков. В результате использования этих команд было выявлено, что действия, выполняемые пользовательским интерфейсом редактора диаграмм, также могут быть описаны не с помощью каких-либо дескрипторов, содержащих предопределенное поведение, а исключительно как сценарии на встроенном языке. Построение интерфейса как части вычислительной модели является дальнейшим направлением исследования.

Для реализации мнемосхемы тракта наддува топливных баков было определено 12 объектов различной сложности. Для примера рассмотрим два из них – объект простой структуры (бак с азотом) и объект сложной структуры (вентиль).

Простой элемент схемы –бак с азотом приведен на рис. 4.14.

Рис. 4.14. Схематическое обозначение бака с азотом

Схематично бак может состоятьиз двух окружностей и двух горизонтальных линий (для простоты текстовая метка не рассматривается). Эта фигура принципиально ничем не отличается от фигур, рассмотренных в подразделе 4.1. Рассмотрим разделение фигуры на элементы визуального алфавита (рис.4.15).

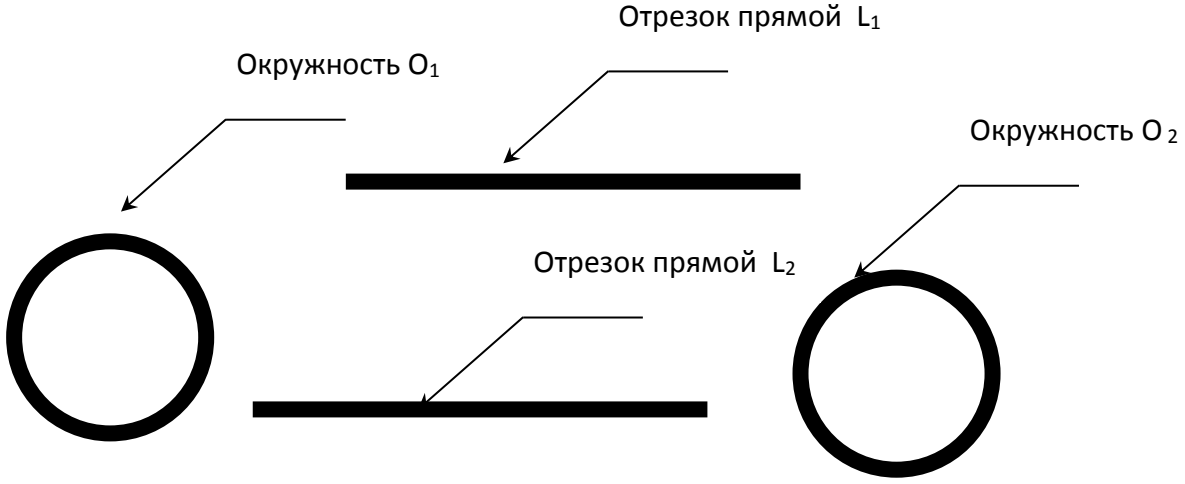


Рис. 4.15. Схема бака с азотом, разбитая на элементы визуального алфавита

Таким образом, фигура состоит из четырех визуальных элементов: двух окружностей O_1 и O_2 , каждая их которых имеет три атрибута – {x,y,r} и двух прямых L_1 и L_2 с атрибутами $\{x_1,x_2,y_1,y_2\}$. Система описывается уравнениями (4.42) и (4.43):

$$\begin{gathered} L_1 = \{x_1, y_1, x_2, y_2\}; \\ L_2 = \{x_1, y_1, x_2, y_2\}; \\ O_1 = \{x, y, r\}; \\ O_2 = \{x, y, r\}. \end{gathered} \tag{4.42}$$

$$\begin{gathered} L_1.y2 = L_1.y_1; \\ O_1.y = L_1.y_1 + O_1.r; \\ O_1.x = L_1.x_1; \\ L_1.x_2 = O_2.x; \\ O_2.x = O_1.x + \text{Длина}; \\ O_2.y = O_1.y; \\ O_2.y = O_1.y; \\ L_2.x_1 = O_1.x; \\ O_2.r = O_1.r; \\ L_2.x_2 = O_2.x; \\ L_2.y_1 = O_1.y + O_1.r; \\ L_2.y_2 = L_2.y_1. \end{gathered} \tag{4.43}$$

Фрагмент семантической сети, соответствующий этой системе уравнений, изображен на рис. 4.16. Изображенные на нем прямоугольники со сплошными границами обозначают атрибуты, а с пунктирными – отношения. Сплошные линии со стрелками обозначают отношения принадлежности, а пунктирные – вхождение атрибутов в отношения в качестве входных или выходных параметров (в зависимости от направления стрелок). Выделенные жирным контуром атрибуты предназначены для воздействия на фигуру, прежде всего, с целью перемещения. Анализ графа показывает, что результат вычислений не зависит от пути вычисления (так как в графе нет узлов-атрибутов, в которые входит более одной пунктирной линии, соответственно, невозможно получить взаимнопротиворечивые пути вычислений), т.е. модель, в терминах Тыугу, является “простой”.

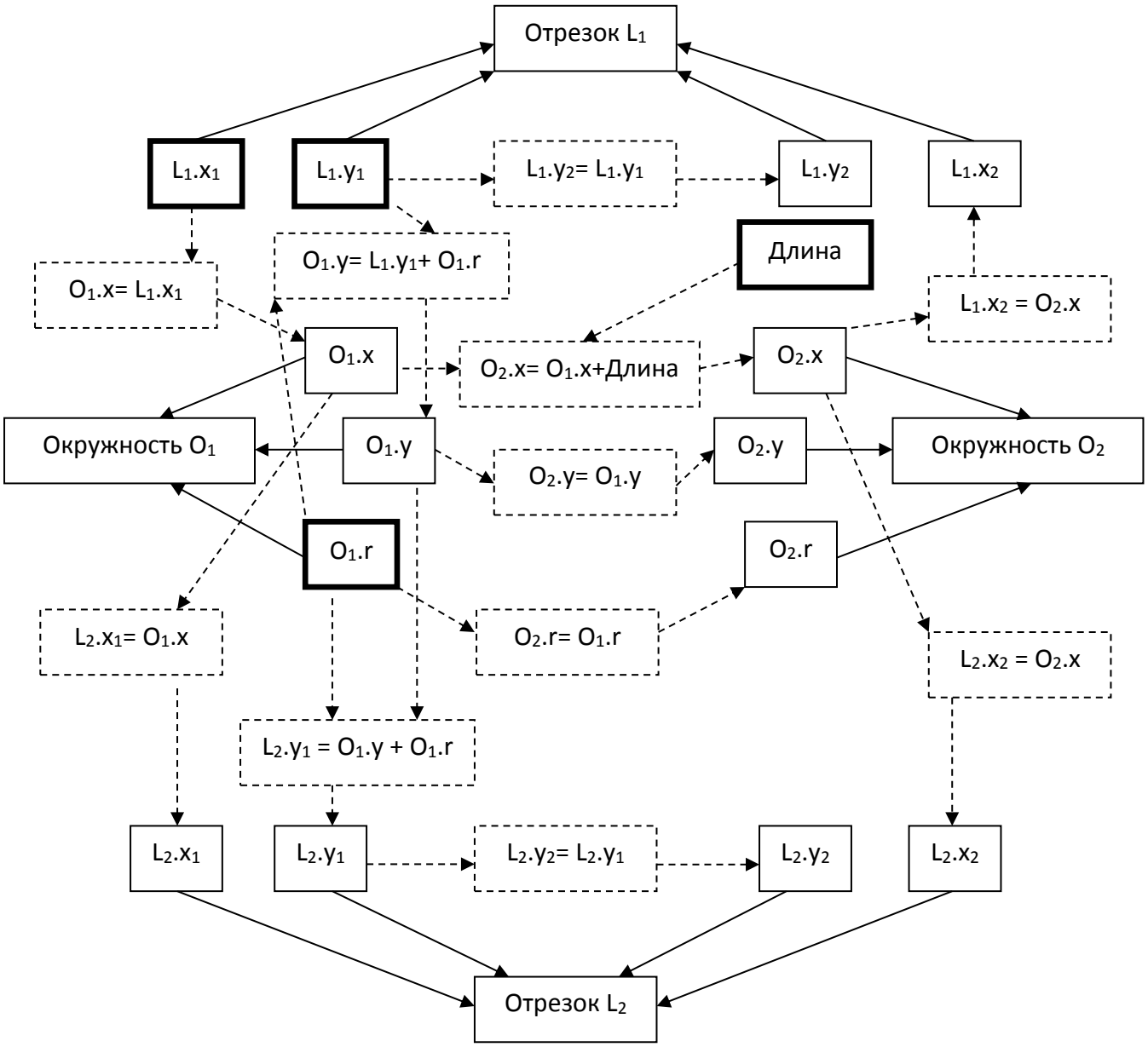


Рис. 4.16. Семантическая сеть, представляющая графическое изображение бака с азотом

Важным вопросом является способ соединения фигуры с графическими элементами магистрали. Бак соединен с тремя магистралями, взаимное положение с которыми необходимо поддерживать. Для этого координаты точек соединения связаны отношениями с атрибутами $L_1.x_1$, $L_1.y_1$ и Длина, т.е. координаты магистралей рассчитываются относительно этих параметров мнемосхемы бака. Перемещение одной мнемосхемы магистрали вызовет

изменение этих атрибутов, в результате чего переместится мнемосхема бака и остальные две магистрали.

В целом мнемосхема бака с азотом проще, нежели сущность в ER-диаграмме. Более сложным элементом является вентиль (рис. 4.17).

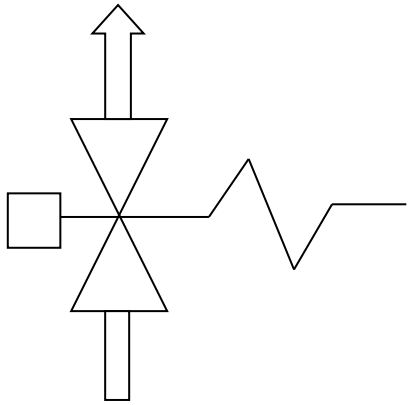


Рис. 4.17. Мнемосхема вентиля

Изображение вентиля состоит из двух треугольников, которые могут иметь различные цвета, прямоугольника слева, стрелки вверх и ломаной линии. Последние три элемента на изображении могут отсутствовать, что вносит в задачу дополнительную сложность.

При изображении вентиля средствами элементарных графических элементов, таких как отрезок прямой, требуется порядка двадцати визуальных элементов. Кроме того, ранее определенным набором визуальных элементов невозможно управлять цветом областей. Таким образом, был сделан вывод о необходимости расширения визуального алфавита более универсальными элементами, а именно – ломаной линией и закрашенной областью, ограниченной ломаной линией (контуром). Таким образом, изображение вентиля делится на визуальные составляющие следующим образом (рис. 4.18):

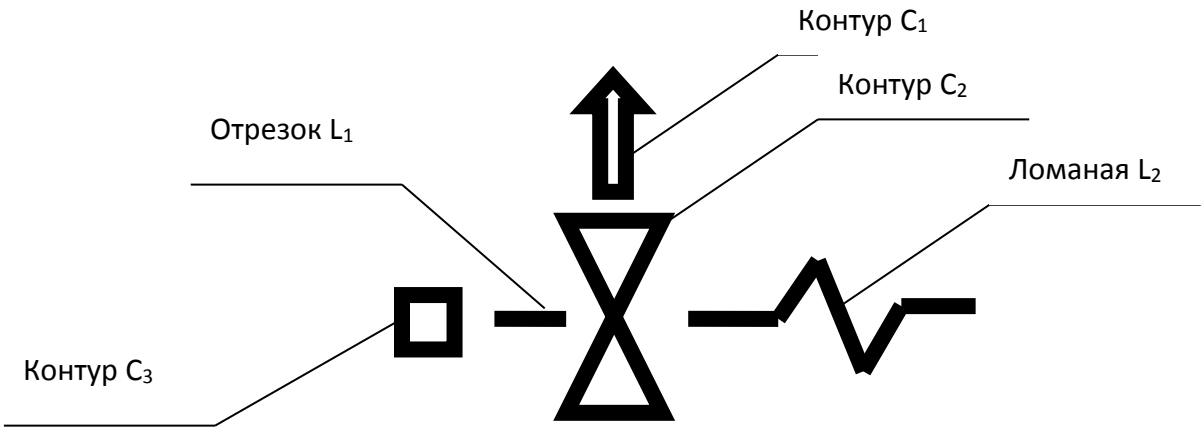


Рис. 4.18. Разделение мнемосхемы вентиля на визуальные элементы

Графический элемент “контур” содержит два атрибута – список координат вершин по абсциссе и ординате, а также атрибут «цвет», указывающий на цвет области, ограниченной контуром. Если атрибут «цвет» не задан, то рисуется только сам контур. Аналогичным образом выполнен графический элемент «ломаная линия», за исключением того, что атрибута «цвет» у нее нет, и она не требует того, чтобы она была замкнутой. Таким образом, за счет введения этих простых элементов, число визуальных составляющих мнемосхемы вентиля упало в четыре раза.

Размер мнемосхемы рассчитывается на основании параметра s, являющегося константой. Параметры X и Y указывают, в какой точке находится левый верхний угол “песочных часов”. Мнемосхема вентиля описывается следующим набором уравнений:

$$\mathrm{C}_1.x = \left[X + \frac{s}{3}, X + \frac{s}{3}, X\frac{s}{6}, X + \frac{s}{2}, X + 5 * \frac{s}{6}, X + 2 * \frac{s}{3}, X + 2 * \frac{s}{3}\right]; \quad (4.44)$$

$$C_1.y = \left[Y, Y - s, Y - s, Y - 3 * \frac{s}{2}, Y - s, Y - s, Y\right]; \quad (4.45)$$

$$\mathrm{C}_2.x = [X, X + s, X, X + s]; \quad (4.46)$$

$$C_2.y = [Y, Y, Y + s * 2, Y + s * 2]; \quad (4.47)$$

$$\mathrm{C}_3.x = \left[X - 3 * \frac{s}{2}, X - \frac{s}{2}, X - \frac{s}{2}, X - 3 * \frac{s}{2}\right]; \quad (4.48)$$

$$\mathrm{C}_3.y = [Y + \tfrac{s}{2}, Y + \tfrac{s}{2}, Y + 3 * \tfrac{s}{2}, Y + 3 * \tfrac{s}{2}]; \quad (4.49)$$

$$L_1.x_1 = X; \quad (4.50)$$

$$L_1.y_1 = Y + s; \quad (4.51)$$

$$L_1.x_2 = X + \frac{s}{2}; \quad (4.52)$$

$$L_1.y_2 = Y + s; \quad (4.53)$$

$$L_2.x = [X + s, X + 2 * s, X + 5 * \frac{s}{2}, X + 3 * s, X + 4 * s]; \quad (4.54)$$

$$L_2.y = \left[Y + s, Y + s, Y + \frac{s}{2}, Y + 3 * \frac{s}{2}, Y + s\right]. \quad (4.55)$$

Остается вопрос, каким образом следует контролировать отсутствие ряда элементов в различных вариантах вентиля. Эта задача решается путем введения в каждый элемент визуального алфавита дополнительного атрибута visible, определяющего, должен он отображаться на экране или нет. Таким образом, в зависимости от ситуации та же самая система уравнений может описывать несколько изображений (рис. 4.19):

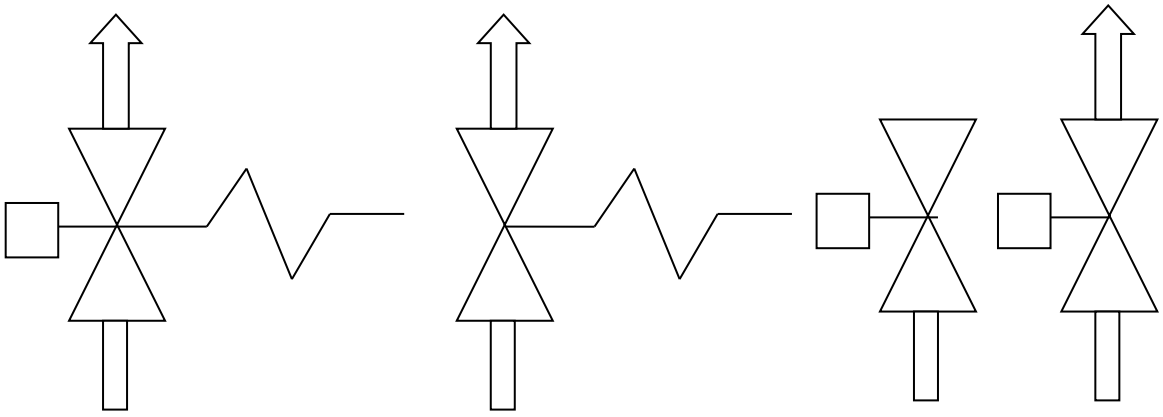


Рис. 4.19. Варианты изображения вентилей

Фрагмент семантической сети, соответствующий вентилю, изображен на рис. 4.20.

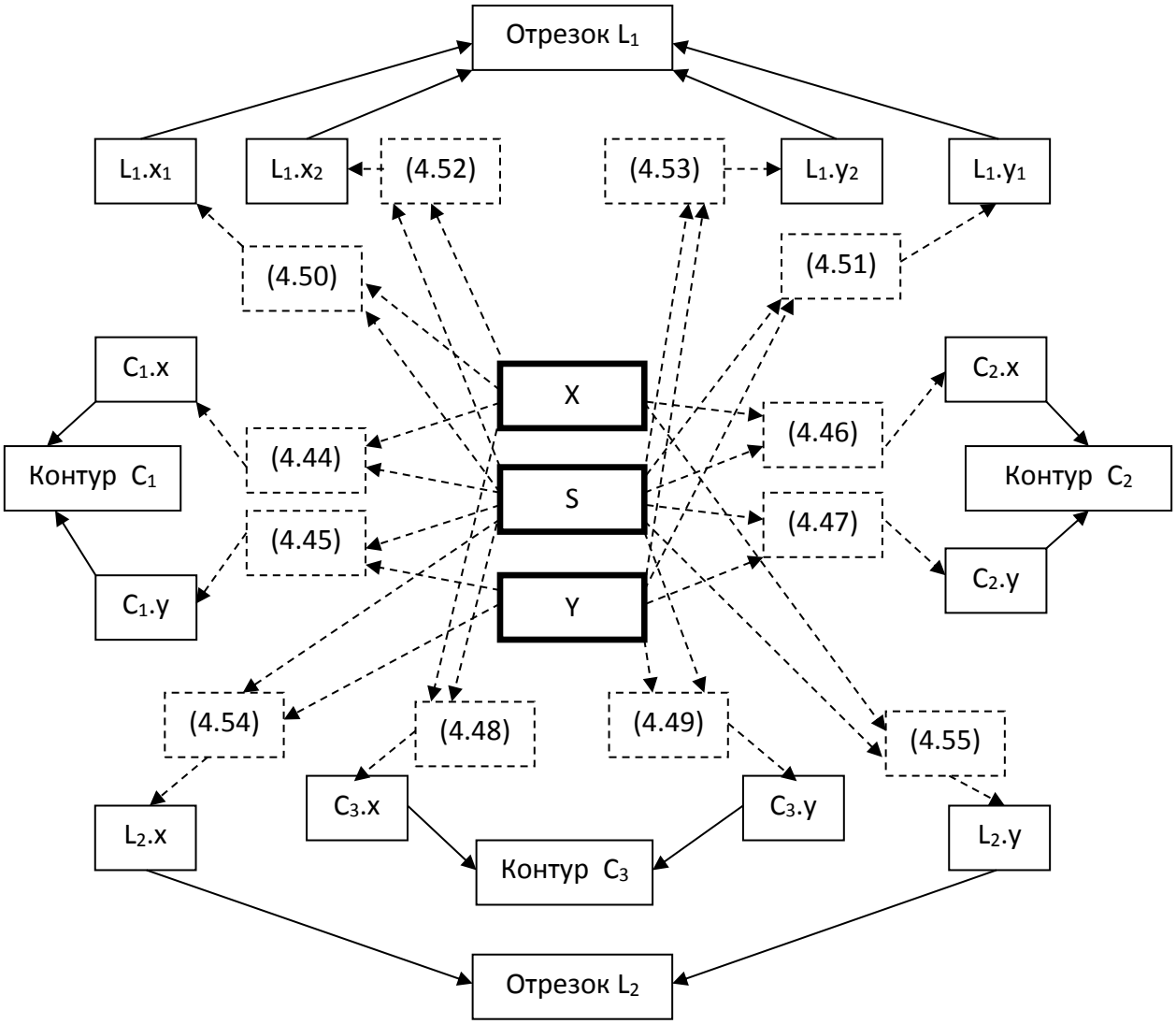


Рис. 4.20. Семантическая сеть, описывающая изображение вентиля

Так же, как и на предыдущем изображении вычислительной модели, прямоугольники со сплошными границами обозначают узлы-значения, а с пунктирными – узлы-отношения. Сплошные стрелки указывают на отношения принадлежности, в пунктирные стрелки – на поток вычисления. В центре изображения на Рис. находятся три интерфейсных атрибута – X, Y (координаты) и S (размер). Все отношения между атрибутами имеют в качестве источника только два из этих трех атрибутов. Поэтому получившаяся вычислительная модель является простой. Таким образом, как можно видеть из приведенной системы уравнений (4.44-4.55) и рис. 4.20, описание вентиля

средствами вычислительной модели визуального языка может быть реализовано.

Еще одним элементом, изображение которого необходимо рассмотреть, является бак с воздухом (рис. 4.21).

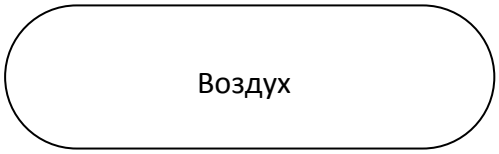


Рис. 4.21. Мнемосхема бака с воздухом

Особенностью этого элемента является наличие скругленных углов. Элемент может быть изображен с помощью восьми примитивных элементов визуального алфавита – четырех дуг и четырех отрезков прямой, однако это не представляется эффективным. Значительно более правильным является введение в набор визуальных примитивов соответствующего элемента – прямоугольника с возможностью скругления углов. Непосредственным выводом из этого, а также из ранее изложенных причин добавления новых элементов является то, что набор визуальных примитивов языка должен включать в себя все основные примитивы рисования графических библиотек. Это связано, в том числе, и с тем, что разработчики визуальных языков ориентируются на эти библиотеки как на средства реализации визуальных редакторов.

4.2.3. Отображение телеметрической информации на мнемосхеме

Изложенные в подразделе 4.2.2 описания графических объектов мнемосхемы тракта наддува топливных баков ценны не сами по себе, а в совокупности с возможностью выполнения мониторинга технического состояния этой системы путем анализа телеметрической информации. Таким образом, возникает задача загрузить телеметрическую информацию из

заданного источника динных и визуализировать ее на мнемосхеме. Прием телеметрической информации должен быть обеспечен в виде, который не зависит от платформы и языка программирования, чтобы графический редактор, отображающий мнемосхему, мог работать с любыми источниками телеметрической информации. Отображение телеметрируемых параметров на мнемосхеме должно производиться в соответствии с принятым де-факто стандартом – с помощью текста (значения параметров) и цвета (оценка корректности). Необходимо отметить, что нет технических препятствий для более сложных способов представления телеметрической информации, например, для динамических мнемосхем.

В качестве средства общения между источником телеметрии и визуальным редактором использована технология сокетов. Редактор открывает серверный сокет, к которому могут присоединиться клиенты, передающие информацию. Каждый клиент передает информацию в виде UNICODE строк определенного формата (пар имя переменной и значение переменной, разделенные двоеточием; собственно пары разделены точкой с запятой). Получив такую строку, редактор рассматривает ее как описание воздействия на вычислительную модель и начинает транзакцию, в рамках которой выполняется пересчет модели. Таким образом, специальное программное обеспечение, функционирующее на любом компьютере в сети, может телеметрировать параметры в графический редактор, исполняющий мнемосхему, вне зависимости от языка программирования и платформы.

Для демонстрации данных телеметрии на экране был добавлен специальный графический объект (рис. 4.22).

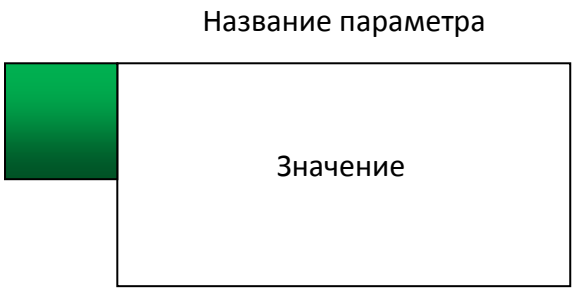


Рис.4.22. Графическое представление параметра телеметрии

Схематически графический объект представляет собой прямоугольник и область, заполненную цветом. В зависимости от обстоятельств цвет может меняться (обычно на красный, желтый или зеленый – так называемый “принцип светофора”).

С точки зрения визуального алфавита вычислительной модели визуального языка, объект состоит из двух прямоугольников с нулевым скруглением углов и двух полей текста – названия параметра и значения. При этом левый прямоугольник имеет атрибут цвета. Название параметра является статическим, значение связано отношением равенства с атрибутом модели, представляющим телеметрируемый параметр. Набор отношений и семантическая сеть здесь не приводятся в виду тривиальности.

Управление автоматической оценкой технического состояния производится путем связывания цвета левого прямоугольника с атрибутом, представляющим телеметрируемый параметр. Например, следующая пара команд определяет, что прямоугольник (внутреннее имя eye) зеленый (65280 = 0x00FF00), если параметр больше либо равен нулю и красный (16711680 = 0xFF0000), если параметр $ABC меньше нуля. При этом

cond($numberValue($ABC))<0) eye.color=16711680;
cond($numberValue($ABC))>=0) eye.color=65280.

Результирующая мнемосхема тракта наддува баков с элементами отображения телеметрической информации представлена на рис. 4.23.

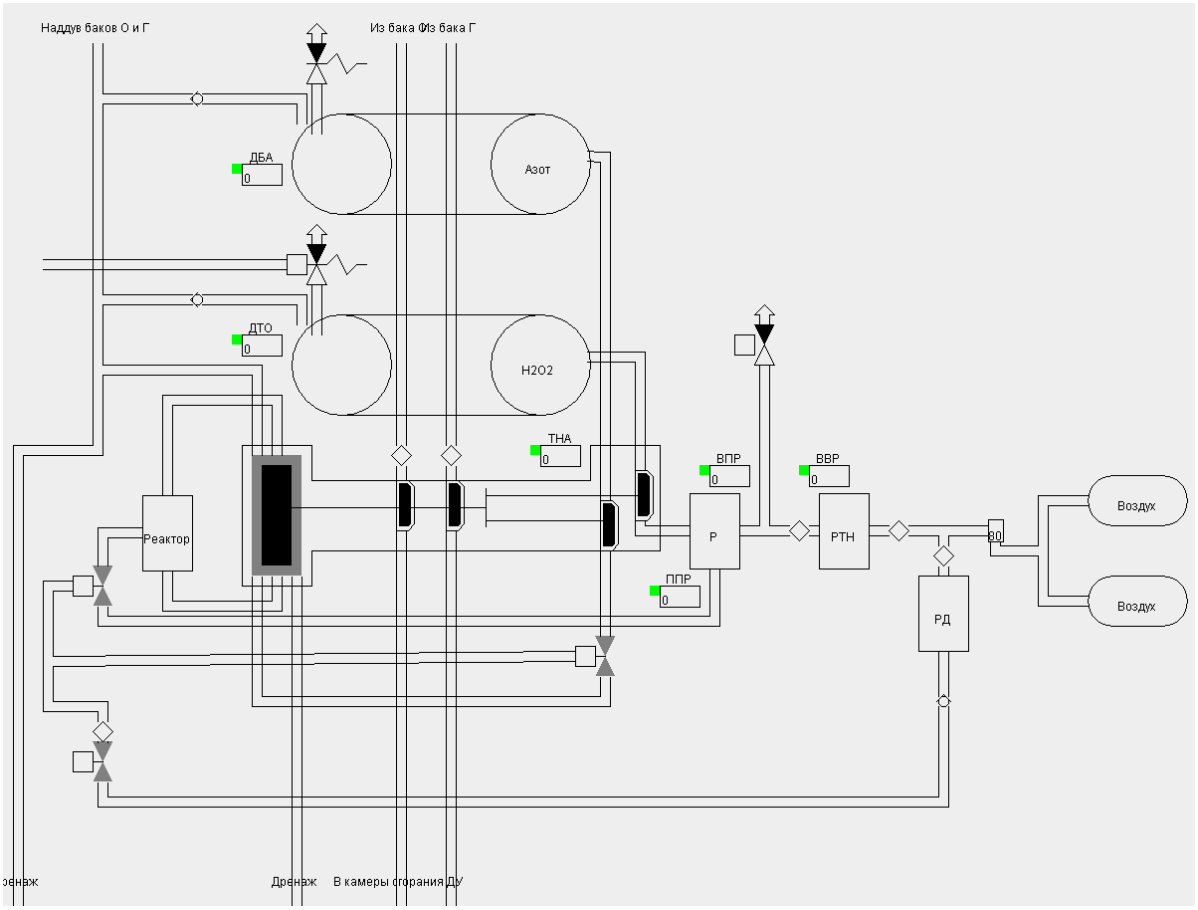


Рис. 4.23. Мнемосхема тракта наддува топливных баков

4.3. Технико - экономическая эффективность

Рассмотрение предмета было бы неполным без экспертной оценки технико-экономической эффективности предложенного специального программного обеспечения (СПО). В этом подразделе будут рассмотрены задачи, решаемые специальным программным обеспечением, описанным в разделе 3, и приведены расчеты экономического эффекта, которые они позволяют достичь.

СПО позволяет решать две основные задачи. Во-первых, это замена имеющихся средств проектирования в условиях импортозамещения. Во-вторых, это разработка средств контроля технического состояния объектов ракетно-космической техники. Рассмотрим две эти задачи.

Импортозамещение редакторов диаграмм в первую очередь должно касаться средств редактирования ER-диаграмм и UML-диаграмм. Ниже представлена стоимость основных продуктов, имеющихся на рынке, в минимальной комплектации (по состоянию на январь 2018г.) [13] (таблица 4.2):

Таблица 4.2.

No	Название продукта	Цена на рабочее место, руб.
1	SAP Power Designer 16.5 (год технической поддержки)	150 000
2	All Fusion ER Win Data Modeller Navigator Edition 9.7 (год технической поддержки)	61 113
3	IBM Rational Rose Modeller Authorized User License (год технической поддержки)	110 000

Вышеперечисленные инструменты являются мощными средствами моделирования с поддержкой большого количества баз данных и широкими возможностями. Предположив, что решаемая задача состоит исключительно в проектировании структуры базы данных, можно считать, что большая часть этих возможностей не нужна.

Оценка времени разработки описания ER-диаграммы из подраздела 4.1 составляет, по предварительным и ориентировочным оценкам, две недели, с учетом визуальной составляющей и трансляции (без реинжиниринга) в одну базу данных. Исходя из расчета зарплаты программиста 150 000 р. в месяц, стоимость разработки одной диаграммы составляет 75 000 р. Таким образом, оценочная стоимость разработки средств, поддерживающих популярные нотации, может составлять (см. таблицу 4.3):

Таблица 4.3.

Нотация	Число диаграмм	Стоимость
ER	1	75 000
UML	12	900 000
IDEF	15	1 125 000

Таким образом, экономическая эффективность импортозамещения средств проектирования баз данных достигается уже при двух лицензиях на рабочее место, средств проектирования UML – при девяти и средств моделирования IDEF – при восьми. Это достигается путем сужения функциональных возможностей до минимально необходимых.

Вторая решаемая специальным программным обеспечением задача – упрощение разработки мнемосхем для оценки технического состояния средств ракетно-космической техники. Использование подходов, описанных в разделе 4.2, позволит решить целый ряд задач, а именно:

- дать возможность любому инженеру разрабатывать мнемосхемы, так как подход не требует специальных знаний из области программной инженерии;
- дать возможность быстро вносить изменения в средства анализа телеметрической информации, так как время внесения изменений – порядка минуты;
- дать возможность интегрировать различные сторонние программные средства с системой визуализации; источник телеметрируемых параметров может быть расположен на другом компьютере в сети и написан на другом языке.

По оценкам экспертов, за счет внедрения вычислительной модели визуального языка стоимость разработки ПО автоматизированных систем оценки качества СТО снижается на 20-30%, а время разработки на 10-15%.

Выводы по разделу 4.

1. Разработан редактор диаграмм” сущность-связь” средствами вычислительной модели визуального языка, включая трансляцию инфологической модели в язык SQL.
2. Вычислительная модель визуального языка пригодна для использования при оценке технического состояния ракетно-космической техники, реализована соответствующая мнемосхема, представляющая одну из подсистем ракеты-носителя “Союз-2”.
3. Телеметрическая информация может быть передана в вычислительную модель через изменение ее интерфейсные атрибутов и является естественной частью этой модели.
4. Наиболее удобно визуализировать телеметрическую информацию используя атрибуты цвета элементов модели. Возможно добавление звука.
5. Библиотека визуальных примитивов требует пересмотра в сторону добавления элементов сложного синтаксиса, так как работа с простыми визуальными примитивами приводит к добавлению в модель большого количества отношений.
6. Использование размеров текста, вычисляемых на основе данных о шрифте и графическом контексте, приводит к нарушению паттерна MVC (модель-представление) и становится невозможным связать одну модель с несколькими графическими устройствами.
7. Отсутствие в модели отношения агрегации накладывает дополнительные ограничения на реализуемые диаграммы. В случае диаграмм” сущность-связь” отсутствие отношения агрегации можнт быть решено за счет допущения, что невозможно одновременно изменить атрибуты более чем одной связи. Для реализации визуальных языков программирования подобное предположение неприемлемо, следовательно, модель должна быть расширена за счет отношения агрегации. Алгоритм 2.1 с введением отношения агрегации также требует доработки.

8. Еще одним существенным ограничением является отсутствие в прикладной вычислительной модели визуального языка более высокоуровневых типов, чем списки – отображений и множеств. Целесообразно расширить прикладную вычислительную модель за счет введения этих типов.
9. Полностью отказаться от традиционного программирования в пользу математических соотношений не удалось даже для столь простой диаграммы, как ”сущность-связь”. Реализация потребовала создания ряда функций на языке JavaScript, замена которых встраиваемыми математическими операциями была признана нецелесообразной.
10. Дальнейшее уменьшение времени на разработку мнемосхем может быть достигнуто за счет введения в вычислительную модель визуального языка отношения наследования. Переиспользование кода объектов с применением полиморфизма позволило бы упростить их логику и уменьшить количество строк описания.
11. Еще одним дальнейшим направлением исследования является построение интерфейсов визуального редактора как части общей вычислительной модели исполняемого им визуального языка.
12. Эффективность внедрения метода в рамках импортозамещения средств проектирования достигается уже при наличии 2- 10 рабочих мест, в зависимости от замещаемой технологии. Это достигается путем сужения возможностей средств проектирования до минимально необходимых.
13. Технико-экономическая эффективность при внедрении подхода для анализа технического состояния средств ракетно-космической техники заключается в предоставлении возможности создания мнемосхем инженерам, не имеющим образования в области IT, а также в возможности интеграции средств визуализации мнемосхем со сторонними источниками телеметрической информации, потенциально написанными на других языках и находящихся на других компьютерах, уменьшении количества ошибок и снижении стоимости разработки ПО для контроля качества ТС.

Заключение

В диссертационной работе сформулирована и решена актуальная научно-техническая задача определения универсального графического редактора диаграмм, использующего единую математическую модель для описания предметной области и правил визуализации. Предложенное решение отличается простотой, универсальностью и свободно от использования сложных формализмов, аналогичных формализмам графовых грамматик. Апробирование предложенного подхода произведено путем его применения для построения графического редактора диаграмм, используемых в ходе испытаний и штатной эксплуатации семейства ракет-носителей "Союз-2". В процессе выполнения исследований в рамках диссертационной работы были получены следующие научные результаты:

1) Предложена модель визуального языка (вычислительная модель визуального языка) на основе модифицированных вычислительных моделей Тыугу, позволяющая эффективно строить человеко-машинные интерфейсы на основе диаграмм;
2) разработан алгоритм расчета модели при воздействии на интерфейс извне, как пользователем, так и внешними приложениями, а также приведения модели к стационарному (детерминированному) виду;
3) разработан алгоритм трансляции диаграмм, заданных вычислительными моделями визуального языка в тексты программ;
4) Разработан метод построения интерфейсов между моделью и источником данных, позволяющий, в частности, эффективно поддерживать взаимодействие с пользователем и источником телеметрической информации.

При этом можно сделать следующие **выводы и рекомендации**:

А. В области теоретических исследований:

1. Вычислительная модель визуального языка на основе модифицированных вычислительных моделей Тыугу позволяет описать визуальный язык, при этом достигается ряд преимуществ по сравнению с графовыми грамматиками и другими средствами формального описания визуальных языков, а именно: относительная простота решения и единообразность математического аппарата с используемым для описания моделей эталонного функционирования оцениваемых объектов; кроме того, более не требуется выделение абстрактного графа синтаксиса. Модель функционирует в соответствии с предложенным алгоритмом пересчета формализованной семантической сети при взаимодействии с ней. Вычислительную модель всегда можно привести к стационарному виду, то есть к виду когда вычисления будут приводить к одному и тому же результату независимо от пути.

2. Добавление в вычислительную модель узлов особого вида – графических примитивов – позволяет реализовывать графическое изображение диаграммы. Задача деления диаграммы на объекты также решается добавлением узлов особого вида – логических объектов, связанных иерархическими отношениями. Таким образом, вычислительная модель трактует все свои составляющие единообразно и одновременно может представлять себя тремя способами – смысловым наполнением (данными), графически и в виде взаимодействующих объектов. Показано, что подобная сеть поддерживает интерфейс как с пользователем, так и со сторонними приложениями. При этом диаграмма имеет механизмы для динамического изменения без участия пользователя, причем на всех уровнях представления, а также простое решение для декомпозиции логического объекта в другую диаграмму.

3. Трансляция диаграмм, заданных в вычислительной модели визуального языка, возможна как через преобразование в графовые грамматики, так и через сценарии обхода графа. Таким образом, предложенная модель визуального языка может описывать собственно язык, диаграмму на

этом языке и использоваться для трансляции диаграммы в текст, то есть является законченной системой, пригодной для описания редакторов диаграмм.

4. Существенные ограничения модель связаны с отсутствием в ней отношений агрегации и наследования. При этом Алгоритм 2.1 с введением отношения агрегации также требует существенной доработки.

Б. В области прикладных исследований:

1. Доказано (путем моделирования некоторых важных классов визуальных языков в разделе 4), что разработанное программное средство и прикладная вычислительная модель визуального языка позволяют реализовать основные визуальные языки, используемые в отрасли информационных технологий.

2. Вычислительная модель визуального языка пригодна для использования при оценке технического состояния ракетно-космической техники, реализована соответствующая мнемосхема, представляющая одну из подсистем ракеты-носителя “Союз-2”. При этом она позволяет получать на вход телеметрическую информацию.

3. Библиотека визуальных примитивов требует пересмотра в сторону добавления элементов сложного синтаксиса, так как работа с простыми визуальными примитивами приводит к добавлению в модель большого количества отношений.

4. Целесообразно расширить прикладную вычислительную модель за счет введения высокоуровневых типов данных (множеств, отображений).

5. Еще одним дальнейшим **перспективным направлением дальнейшей разработки темы** является построение интерфейсов визуального редактора как части общей вычислительной модели исполняемого им визуального языка.

6. Технико-экономическая эффективность при внедрении подхода для анализа технического состояния средств ракетно-космической техники заключается в предоставлении возможности создания мнемосхем инженерам, не имеющим образования в области IT, а также в возможности интеграции средств визуализации мнемосхем со сторонними источниками телеметрической информации, потенциально написанными на других языках и находящихся на других компьютерах.

Основной эффект от использования предлагаемого способа заключается в уменьшении сроков и трудоемкости разработки новых средств графического редактирования диаграмм, необходимость в которых возникает в рамках политики импортозамещения, а также при решении узкоспециализированных задач в ракетостроении.

Полученные результаты соответствуют п. 3 «модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем» п. 7 «человеко-машинные интерфейсы; модели, методы, алгоритмы и программные средства машинной графики, визуализации, обработки изображений, систем виртуальной реальности, мультимедийного общения» паспорта специальности 05.13.11 «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей».

Список литературы

1. Ахо, А. Теория синтаксического анализа, перевода и компиляции: в 2-х т. Т.1 / А. Ахо, Дж. Ульман; ред. В. М. Курочкин, перев. В. И. Агафонов - Москва: Мир, 1978.

2. Ахо, А. Теория синтаксического анализа, перевода и компиляции: в 2-х т. Т.2 / А. Ахо, Дж. Ульман; ред. В. М. Курочкин, перев. В. И. Агафонов - Москва: Мир, 1978.

3. Буч, Г. UML руководство пользователя / Г. Буч, Дж. Рамбо, А. Джекобсон; перев. А.А. Слинкин - Москва: ДМК, 1999.

4. Вирт, Н. Алгоритмы + структуры данных = программы / пер. с англ. Л.Ю. Иоффе; под ред. Д.Б. Подшивалова. – Москва: Мир, 1985. – 406 с. – (серия: «Математическое обеспечение ЭВМ»).

5. Гладкий, А.В. Формальные грамматики и языки / Москва: Мир, 1973. – 368 с.

6. ГОСТ 19.701-90 (ИСО 5807-85). Схема алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения – Введ. 01.01.1992. – Москва: Изд-во стандартов, 1991.

7. Городецкий, В.И. Технология разработки прикладных многоагентных систем в инструментальной среде MASDK / В.И. Городецкий, О.В. Карсаев, В.Г.Конюший, В.В. Самойлов, Е.В. Маньков, А.В. Малышев // Труды СПИИРАН. Т. 1, №3. - СПб. - 2006 - с.11–32

8. Дубейковский, В. Эффективное моделирование с CA ERwin Process Modeler (BPwin; AllFusion Process Modeler) / Москва: Диалог-МИФИ. 2009. – 284 с.

9. Ершов, А.П. Операторные алгоритмы // Проблемы кибернетики: Сб. статей. Вып. 3. – Москва: Физматгиз, 1960. – С. 5-48.

10. Ершов, А.П. Современное состояние теории схем программ //Проблемы кибернетики: Сб. статей. Вып. 27. – Москва: Наука, 1973. – С. 87-110.

11. Иванищев, В.В. Язык алгоритмических сетей: препринт. №63 / В.В. Иванищев, В.Е. Марлей, В.П. Морозов. – ЛНИВЦ АН СССР. – Л., 1984. – 37 с.

12. Игнатьев, М.Б. Моделирование слабо формализованных систем на основе явных и неявных экспертных знаний / М.Б. Игнатьев, В.Е. Марлей, В.В. Михайлов, А.В. Спесивцев / СПб.: Политех. 2018. — 500 с.

13. Интернет-магазин компании “Интерфейс” [Электронный ресурс]: URL: http://www.itshop.ru (дата обращения: 1.08.2019).

14. Каргин, В.А. Автоматизированная система информационной поддержки принятия решений по контролю в реальном времени состояния ракетно-космической техники / В.А. Каргин, О.В. Майданович, М.Ю. Охтилев // Приборостроение, 2010. - Т. 53, № 11. - С. 20-23.

15. Клумова, И. Н. Игра «Жизнь» // Квант. — 1974. — № 9. — С. 26—30.

16. Котов, В.Е. Сети Петри / Москва: Наука, 1984. –160 с.

17. Котов, В.Е. Введение в теорию схем программ / Москва: Наука, 1978. – 260 с.

18. Котов, В.Е. Теория схем программ / В.Е. Котов, В.К. Сабельфельд - Москва: Наука, 1991. –248 с.

19. Ляпунов, А. А. О логических схемах программ// Проблемы кибернетики. — 1958. — Вып. 1. — С. 46—74.

20. Майданович, О.В. Комплексная автоматизация мониторинга состояния космических средств на основе интеллектуальных информационных технологий / О.В. Майданович, М.Ю. Охтилев, Б.В. Соколов, Р.М. Юсупов // Информационные технологии. – Москва: Наука, 2011. - № 10. – С. 2-29.

21. Нартова, А. PowerDesigner 15. Моделирование данных / А. Нартова. – Москва: Лори, 1992. – 480 с.

22. Непейвода, Н. Н. Выводы в форме графов //Семиотика и информатика / Н.Н. Непейвода, - Москва: ВИНИТИ, 1986. - №26. - С. 52-82.

23. Охтилев, М.Ю. Теория и практика построения автоматизированных систем мониторинга технического состояния космических средств / М.Ю. Охтилев,

Б.В. Соколов и др.; под ред О.В.Майдановича; – СПб.: ВКА им. А.Ф.Можайского, 2011. - 219 с.

24. Охтилев, М.Ю. Интеллектуальные технологии мониторинга и управления структурной динамикой сложных технических объектов / М.Ю. Охтилев, Б.В. Соколов, Р.М. Юсупов - М.: Наука, 2006. - 410 с.

25. Солонина, А. И. Цифровая обработка сигналов. Моделирование в Simulink / А.И. Солонина – СПб.: БХВ, 2012. – 432 с.

26. Солоницын, Ю. Microsoft Visio 2007. Создание деловой графики / Ю. Солоницын – СПб.:Питер, 2009. – 160 с.

27. Степанов, П.А. Создание универсального репозитория для хранения диаграмм, используемых CASE – средствами / П.А. Степанов, В.В. Фильчаков // тезисы конференции. Шестая международная научно-техническая конференция студентов и аспирантов “Радиотехника, электроника и энергетика”: Москва, 16-17 марта 2000г. – Москва: МЭИ, 2000. - том 1. - С. 302.

28. Степанов, П.А. Методы построения универсального репозитория для хранения диаграмм, используемых CASE – средствами / П.А. Степанов // Тезисы докладов 8й международной студенческой школы-семинара “Новые информационные технологии” Украина, г. Судак 1-7 мая 2000г. – Москва: МГИЭМ, 2000 - С. 303.

29. Степанов, П.А. Подходы к созданию универсального репозитория, используемого CASE – средствами / П.А. Степанов // тезисы докладов. 3я научная сессия аспирантов ГУАП. Санкт-Петербург, 10-14 апреля 2000г. – СПб.: ГУАП, 2000. - С.161.

30. Степанов, П.А. Описание диаграммного языка и разработка генератора диаграммеров на основе универсального репозитория. / П.А. Степанов, А.В. Бржезовский // Тезисы докладов. Пятая Санкт-Петербургская Ассамблея молодых учёных и специалистов. СПб, 19 декабря 2000г. - Спб: СПбГУ, 2000. - С.45.

31. Степанов, П.А. Описание семантики диаграмм на основе отношений между данными графических объектов / П.А. Степанов, А.В. Бржезовский // Седьмая международная научно-техническая конференция студентов и аспирантов “Радиотехника, электроника и энергетика” Москва, 15-16 марта 2001г. – Москва: МЭИ, 2001. - Том 1. - С. 271.

32. Степанов П. А., Исследование применимости вычислительных моделей для создания редактора диаграмм на примере ER-модели / П.А. Степанов // 19я научная сессия ГУАП СПб. 11-15 апреля 2016г. - СПб.: ГУАП, 2016. - С.263-269.

33. Степанов, П. А. Вычислительная модель визуального языка / П. А. Степанов, М. Ю. Охтилев // Изв. вузов. Приборостроение. 2006. - Т. 49, № 11. - С. 28—32.

34. Степанов, П. А., Применение вычислительных моделей для создания редактора диаграмм / П. А. Степанов, М. Ю. Охтилев // Изв. вузов. Приборостроение. 2016. - Т. 59, № 11. - С. 939—942. DOI: 10.17586/0021-3454-2016-59-11-939-943

35. Степанов, П. А. Применение вычислительной модели визуального языка к задачам визуального контроля технического состояния ракетно-космической техники / П.А. Степанов //Авиакосмическое приборостроение. - 2017. - №5. - С. 28-32.

36. Степанов, П.А. Эффективная разработка мнемосхем при контроле технического состояния ракетно-космической техники / П.А.Степанов, М.Ю. Охтилев // Труды конференции. 19я международная конференция “Проблемы управления и моделирования в сложных системах”. Самара, 12-15 сентября 2017 г. – Самара: ИПУСС РАН, 2017. – С. 224-229.

37. Степанов, П. А. Визуализация телеметрической информации средствами вцычислительных моделей // Сборник докладов конференции. 20я научная сессия ГУАП, СПб. 10-14 апреля 2017 г. - Санкт-Петербург: ГУАП, 2017. - С.293-296.

38. Степанов, П. А. Реализация пользовательского интерфейса редактора диаграмм средствами вычислительных моделей // Сборник докладов конференции. 20я научная сессия ГУАП. Санкт-Петербург, 10-14 апреля 2017 г. Санкт-Петербург: ГУАП, 2017. - С.297-300.

39. Степанов, П.А. Построение визуальных средств анализа телеметрической информации при оценивании технического состояния космических средств с использованием вычислительных моделей [Электронный ресурс] // Труды конференции. Восьмая всероссийская научно-практическая конференция «Имитационное моделирование. Теория и практика» (ИММОД-2017) г. Санкт-Петербург, 18-20 октября 2017 г. - СПб.: Изд-во ВВМ, 2017. - С. 294-298.

40. Степанов, П.А. Визуализация состояния сложных технических объектов с помощью вычислительных моделей / П.А. Степанов, М.Ю. Охтилев, Б.В. Соколов // Информационно-управляющие системы, 2017. - № 6. - С. 132–135. doi:10.15217/issn1684-8853.2017.6.132

41. Степанов П.А. Использование вычислительных моделей для оценки и отображения технического состояния дорожной архитектуры / П. А. Степанов // Автоматизация в промышленности. 2018. - № 4. - С. 29–32.

42. Степанов П.А. Применение вычислительных моделей при обнаружении источников отказов в облачных инфраструктурах / П. А. Степанов // Информатизация и связь. 2019. - № 3. - С. 108–111.

43. Свидетельство о государственной регистрации программы для ЭВМ № 2017615060 “Анализатор и интерпретатор математических соотношений, заданных в вычислительной модели визуального языка”.

44. Свидетельство о государственной регистрации программы для ЭВМ № 2017617893 “Программа построения графа зависимостей переменных для вычислительной модели визуального языка”.

45. Свидетельство о государственной регистрации программы для ЭВМ № 2017617879 “Программа управления внутренней архитектурой вычислительной модели визуального языка”.

46. Терехов, А. Н. REAL: методология и CASE-средство для разработки систем реального времени и информационных систем / А. Н. Терехов, К.Ю. Романовский, Д. В. Кознов, П. С. Долгов, А. Н. Иванов // Программирование. 1- 999. - № 5. - С. 44–52.

47. Терехов, А. Н. Архитектура среды визуального моделирования QReal / А. Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов // Системное программирование. – 2009. – Вып. 4. – С. 171-196.

48. Тревис, Дж. Lab View для всех / Дж. Тревис, Дж. Кринг; перев. М. Михеев - Москва: ДМК пресс, 2015.

49. Тыугу, Э. Х. Вычислительные фреймы и структурный синтез программ // Известия АН СССР. Техническая кибернетика. - 1982 г. -№ 6. - С. 176-182.

50. Тыугу, Э. Х. Исследование по математическому обеспечению ЭВМ. - Таллинн: Академия наук Эст. ССР, 1973-1979.

51. Тыугу, Э. Х. Концептуальное программирование. - Москва: Наука, 1984. – 256 с.

52. Тыугу, Э. Х. Решение задач на вычислительных моделях // Журнал вычислительной математики и математической физики. - 1970 г. - №3, Т. 10. - С. 716-733.

53. Хомский, Н. Введение в формальный анализ естественных языков / Н. Хомский, Дж. Миллер // Кибернетический сборник, под ред. А.А.Ляпунова и О.Б.Лупанова. — М.: Мир, 1965.

54. Черемных, С.В. Структурный анализ систем: IDEF-технологии / С.В. Черемных, И.О. Семенов, В.С. Ручкин – Москва.: "Финансы и статистика", 2001. - 208 с.

55. Чен, П. Модель "сущность-связь" – шаг к единому представлению о данных / П. Чен // Системы Управления Базами Данных. - Москва: Открытые Системы, 1995 г. - 3. - С. 137-158.

56. Шмелев, В.В. Модели технологических процессов функционирования космических средств / В.В. Шмелев //Авиакосмическое приборостроение. – 2015. - № 4. – С. 78-93.

57. Янов, Ю. И. О логических схемах алгоритмов / Ю. И. Янов // Проблемы кибернетики. — 1958. — Вып. 1. — С. 75—127.

58. Янов, Ю. И. О локальных преобразованиях схем алгоритмов / Ю. И. Янов // Проблемы кибернетики. — 1968. — Вып. 20. — С. 201—216.

59. Adachi, Y. An NCE Context-sensitive Graph Grammar for Visual Design Languages. / Y. Adachi, S. Kobayashi, K. Tsuchida, T. Yaku // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain 1999. – P. 228-235.

60. Adachi, Y. A Context-Sensitive NCE Graph Grammar and its Parsability. / Y. Adachi, Y. Nakajima // Proceedings of 2000 IEEE International Symposium on Visual Languages, Sept. 10-13, 2000, USA, WA, Seattle. - 2000. - P. 111-118.

61. Atkins, M. S. Role of Visual Languages in Developing Image Analysis Algorithms. / M. S. Atkins, T. Zuk, B. Johnston, S. Fraser // 1994 IEEE Symposium on Visual Languages. USA. MO, St. Louis. - 1994. – P. 262-269.

62. Barendregt, H.P. The Lambda Calculus – Its Syntax and Semantics. / H.P. Barendregt - 1981 – 621 pp.

63. Bawden, A. Connection Graphs. / A. Bawden // Proceedings of the ACM Conference on Lisp and Functional Programming (LFP). USA, Massachusets, Cambridge, NY.:ACM, 1986. - 1986. - P. 258-265.

64. Beaumont, M. Low Level Visual Programming. / M. Beaumont, D. Jackson // Proceedings of 1997 IEEE International Symposium on Visual Languages. Sep 23-27, Italy, Capri. – 1997. – P. 410-417.

65. Bell, M.A. Visual Author Languages for Computer-Aided Learning / M.A. Bell, D. Jackson // 1992 IEEE Workshop on Visual Languages. Sept. 15-18, 1992, USA, WA, Seattle. - 1992. – P. 258-260.

66. Bianchi, N. Plastic Visual Tools / N. Bianchi, P. Bottoni, P. Mussio, M. Protti // 1992 IEEE Workshop on Visual Languages, 1992. Sept. 15-18, 1992, USA, WA, Seattle. - P. 258-260.

67. Bianchi, A. A Visual System for the Generation of Banking Legacy System Gateways. / A. Bianchi, G. Costagliola, P. D'Ambrosio, R. Franchese, G. Scanniello

// Proceedings of 2001 IEEE International Symposia on Human-Centric Languages and Environments, Sep. 18-22, 2001, USA, PA, Pittsburg. – 2001. – P. 350-357.

68. Bianchi, A. Designing Usable Visual Languages: The Case of Immune System Studies / A. Bianchi, M. D'Enza, M. Matera, A. Betta // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 254-261.

69. Bimonte, S. Prototype of a Visual Language for Spatial Data Mining Based on the 'Miner Trip' Metaphor: VisMiner. / S. Bimonte, F. Fenucci, R. Laurini, G. Polese // Proceedings of IEEE 2003 International Symposia on Human-Centric Computing Languages and Environments, Oct. 28-31 2003 New Zeland, Aukland. – 2003. – P. 76-83.

70. Birchman, J. J. An Implementation of the VIVA Visual Language on the NeXT Computer. / J. J. Birchman, S. L. Tanimoto // Proceedings of 1992 IEEE Workshop on Visual Languages. Sept. 15-18, 1992, USA, WA, Seattle. - 1992 - P.177-183.

71. Blackwell, A. F. Metacognitive Theories of Visual Programming: What do we think we are doing? / A. F. Blackwell // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 240-246.

72. Blackwell, A.F. Does Metaphor Increase Visual Language Usability? / A.F. Blackwell, T.R.G. Green // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 204-205.

73. Borges, J.A. Multiparadigm Visual Programming Language / J.A. Borges, R.E. Johnson // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 233-240.

74. Bottoni, P. Dimensions of Visual Interaction Design / P. Bottoni, S.-K. Chang, M.F. Costabile, S. Levialdi, P. Mussio // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 288-295.

75. Bottoni, P. Towards Formal Measures of Usability for Visual Interactive Systems. / P. Bottoni, M.F. Costabile, S. Levialdi, A. Piccinno // Proceedings of

IEEE 2002 International Symposia on Human-Centric Computing Languages and Environments, Sep. 3-6 2002 USA, Virginia, Arlington. – 2002. – P. 188-197.

76. Bottoni, P. Formalising Visual Languages. / P. Bottoni, M.F. Costabile, S. Levialdi, P. Mussio // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 45-52.

77. Bottoni, P. Visual Conditional Attributed Rewriting Systems in Visual Language Specification / P. Bottoni, M.F. Costabile, S. Levialdi, P. Mussio // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 156-163.

78. Bricken, W. Spatial Representation of Elementary Algebra. / W. Bricken // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. – P. 56-62.

79. Brooks, F. No Silver Bullet / Brooks, F. // IEEE Computer. – 1987. - № 20(4). – P. 10-19.

80. Brown, D.R. The Whiteboard Environment: An Electronic Sketchpad for Data Structure Design and Algorithm Description. / D.R. Brown, B. Van der Zanden // Proceedings of 1998 IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P. 288-295.

81. Brown, M.H. Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom / M.H. Brown, M.A. Najork // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 266-275.

82. Brown, M.H. Color and sound in algorithm animation / M.H. Brown, J. Hershberg // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe. - 1991. – P. 10-17.

83. Burnett M. Abstraction in the Demand-Driven, Temporal-Assignment, Visual Language Model / Ph.D. Thesis, Department of Computer Science, University of Kansas. – 1991.

84. Campbell, J.D. A Visual Language System for Developing and Presenting Internet-based Education. / J.D. Campbell, D.E. Mahling //Proceedings of 1998

IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P. 66-67.

85. Catarci, T. Iconic and Diagrammatic Interfaces: Integrated Approach / T. Catarci, A. Massari, G. Santucci // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.–P. 199-204.

86. Chang, S.-K. Visual Languages: A Tutorial and Survey / S.-K. Chang //IEEE Software Magazine. - Jan, 1987. - Vol 4 issue 1.

87. Chang, S.-K. Picture Processing Grammar and Its Applications. / S.-K. Chang // Information Sciences. -1971. - №3 - P. 121 – 148.

88. Chang, S.-K. A Visual Language for Authorization Modeling. / S.K. Chang, G. Lese, R. Thomas, S. Das // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. – P. 110-118.

89. Chang, S.-K. Iconic Indexing by 2-D strings / S.-K. Chang, Q. Y. Shi // IEEE Transitions on Pattern Analysis and Machine Intelligence. - Vol. PAMI-9, No 3. – P. 413-428.

90. Chang, S.-K. Deriving the Meaning of Iconic Sentences for Augmentative Communication / S.-K. Chang, S. Orefice, G. Polese, B.R. Baker // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 267-274.

91. Chow, A.L. Topological Compositions Systems: Specification for Lexical Elements of Visual Languages / A.L. Chow, R.V. Rubin // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991. - P. 118-124.

92. Chattratichat, J. A Visual Language for Internet-based Data Mining and Data Visualization. / J. Chattratichat, Y. Guo, J. Syed// Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 64-71.

93. Chok, S.S. Automatic Construction of User Interfaces from Constraint Multiset Grammars / S.S. Chok, K. Marriott // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 242-249.

94. Chomsky, N. The Logical Structure of Linguistic Theory. / N. Chomsky // Chicago. - University of Chicago Press. – 1975. –592 pp.

95. Citrin, W.V. Requirements for Graphical Front Ends for Visual Languages / W.V. Citrin // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 142-150.

96. Citrin, W.V. An Execution Model for Demonstration-Based Visual Languages / W.V. Citrin // 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. – P. 249-251.

97. Citrin, W. Programming with Visual Expressions. / W. Citrin, R. Hall, B. Zorn // IEEE Symp. On Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 294-301.

98. Collopy, F. Visual Music in a Visual Programming Language / F. Collopy, R. M. Fuhrer, D. Jameson // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 111 – 120.

99. Costagliola, G. DR PARSERS: a generalization of LR parsers / G. Costagliola, S.-K. Chang // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 174-180.

100. Costagliola, G. A Framework of Syntactic Models for the Implementation of Visual Languages / G. Costagliola, A. De Lucia, S. Orefice, G. Tortora // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. – 1997. –P. 58-65.

101. Costagliola, G. Using Extended Positional Grammars to Develop Visual Modeling Languages. / G. Costagliola, V. Deufemia, F. Ferucci, C. Gravino//Proceedings of the 14th international conference on Software engineering and knowledge engineering, Jul 15-19 2002 Italy, Ischia. - Vol. 27. – 2002. – P.201-208.

102. Costagliola, G. Automatic Parser Generation for Pictorial Languages / G. Costagliola, S. Orefice, G. Polese, G. Tortora, M. Tucci // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - P. 306-313.

103. Costagliola, G. Extended Positional Grammars. / G. Costagliola, G. Polese // Proceedings of 2000 IEEE Symposium on Visual Languages, Sep 10-13 2000, USA, WA, Seattle. - 2000. - – P. 103-110.

104. Costagliola, G. A Visual Language Based System for the Efficient Management of the Software Development Process. / G. Costagliola, G. Polese, G. Tortora, P. D'Ambrosio // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. - P. 426-429.

105. Costagliola, G. A Generalized Parser for 2-D Languages. / G. Costagliola, M. Tomita, S.-K. Chang // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.- P. 98-104.

106. Costagliola, G. Extended Positional Grammars: A Formalism for Describing and Parsing Visual Languages / G. Costagliola // Visual Languages for Interactive Computing: Definitions and Formalization - 2007 – P.102-116,

107. Coulmann, L. General Requirements for a Program Visualization Tool / L. Coulmann // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 37-41.

108. Cox, K.C. Abstraction in Algorithm Animation / K.C. Cox, G.-C. Roman //Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. – 1992. – P.18-25.

109. Cox, P.T. Prograph: a step towards liberating programming from textual from textual conditioning. / P.T. Cox, F.R. Giles, T. Pietrzykowsky // Proceedings of 1989 IEEE Workshop on Visual Languages, Oct 4-6 Italy, Rome,1989. - 1989. - P. 150-156.

110. Cox, P.T. Visual Programming for Robot Control. / P.T. Cox, T.J. Smedley // Proceedings of 1998 IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P.217-224.

111. Crimi, C. Relation Grammars for Modelling Multi-dimensional Structures. / C. Crimi, A. Guercio, G. Nota, G. Pacini, G. Tortom, M. Tucci // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 168-173.

112. Davis, M. Media Streams: An Iconic Visual Language for Video Annotation. / M. Davis // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 196-202.

113. Del Bimbo, A. Spatio-temporal Logic for Image Sequence Coding and Retrieval. / A. Del Bimbo, E. Vicario, D. Zingoni // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. – 1992. - P. 228-230.

114. Del Bimbo, A. Sequence Retrieval by Contents through Spatio Temporal Indexing. / A. Del Bimbo, E. Vicario, D. Zingoni // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 88-92.

115. Del Bimbo, A. Visual Specification of Virtual Worlds / A. Del Bimbo, E. Vicario, D. Zingoni // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. –P. 376-378.

116. Del Bimbo, A. Visual Image Retrieval by Elastic Deformation of Object Sketches. / A. Del Bimbo, P. Pala, S. Santini // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. - P. 216-223.

117. Delest, M. CALICO, A Visual Tool for Combinatorial Mathematics. / M. Delest, J.M. Fedou, G. Melancon, N. Rouillon // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 379-380.

118. Earley J. An Efficient Context-Free Parsing Algorithm. / J. Earley // Communications of ACM. - 1970. - №13, Vol 2. - P. 94-102.

119. Eden, M. On the formalization of handwriting. / M. Eden //Proc. Symposia in Appl. Math. (1960) - Vol. 12, Amer Math. Soc., Providence, R. I. - 1961, - P. 83-88.

120. Eden, M. Handwriting and Pattern Recognition. / M. Eden // IRE Trans. - IT-8. – 1962. - P. 160-166.

121. Ehrig, H. Fundamentals of Algebraic Graph Transformation. / H. Ehrig, K. Ehrig, U. Prange, G. Taentzer - Springer-Verlag, Berlin – Heidelberg. – 2006. – 390 pp.

122. Erwig, M. DEAL - A Language for Depicting Algorithms. / M. Erwig // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. – P. 184-186.

123. Erwig, M. Functional Programming with Graphs. / M. Erwig // 2nd ACM SIGPLAN Int. Conf. On Functional Programming, Amsterdam, The Netherlands. – 1997. - P. 52-65.

124. Erwig, M. Inductive Graphs and Functional Graph Algorithms / M. Erwig // Journal of Functional Programming. – 2001. - Vol. 11, No. 5. - P. 467-492.

125. Erwig, M. Semantics of Visual Languages. / M. Erwig // 13th IEEE Symp. on Visual Languages, Sep. 23-26 Italy, Capri. -1997.- P. 300-307.

126. Erwig, M. Visual Graphs. / M. Erwig // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P.122-129.

127. Esser, R. Moses – a tool suite for visual modeling of discrete-event systems. / R. Esser, J.W. Janneck // Proceedings of 2001 IEEE International Symposia on Human-Centric Languages and Environments, Sep. 18-22, 2001, USA, PA, Pittsburg. – 2001. – P. 272-279.

128. Ferucci, F. Efficient Parsing of Multidimensional Structures / F. Ferucci, G. Panini, G. Tortora, M. Tucci, G. Vitiello // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.- P. 105-110.

129. Ferrucci, F. A Predictive Parser for Visual Languages Specified by Relation Grammars. / F. Ferrucci, G. Tortora, M. Tucci, G. Vitiello // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. - P. 245-252.

130. Ferrucci, F. An Interpreter for Diagrammatic Languages Based on SR Grammars / F. Ferrucci, F. Napolitano, G. Tortora, M. Tucci, G. Vitiello //

Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. -P. 292-299.

131. Franck, G. Representing Nodes and Arcs in 3D Networks / G. Franck, C. Ware // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. – 1994. – P. 189-190.

132. Futrelle, R. P. Understanding Diagrams in Technical Documents. / R. P. Futrelle, I. A. Kakadiaris, J. Alexander, C. M. Carriero, N. Nikolakis, J. M. Futrelle // IEEE Computer. - 25(7). - 1992. -P. 75-78.

133. Futrelle, R. P. Efficient Analysis of Complex Diagrams using Constraint-Based Parsing. / R. P. Futrelle, N. Nikolakis // Intl. Conf. on Document Analysis & Recognition, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 782-790.

134. Glaser, H. Psh - The Next Generation of Command Line Interfaces. / H. Glaser, T.J. Smedley // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 29-39.

135. Geltz, M. A Framework for Visualizing Data Structures / M. Geltz // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 396-400.

136. Glinert, E.P., Introduction to Visual Programming Environments / E.P. Glinert // SICGRAPH ’89 Course Notes, ACM. - 1989.

137. Glinert, E.P. Visual Tools and Languages: Directions for the 90’s / E.P. Glinert // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.– P. 89-95.

138. Gloor, P.A. AACE - Algorithm Animation for Computer Science Education / P.A. Gloor // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P.25-31.

139. Green, T.R.G. Pictures of Programs and Other Processes, or How to Do Things With Lines. / T.R.G. Green// Behaviour and Information Technology. – 1989. - P 3-36.

140. Green, T.R.G. Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against Match-Mismatch Conjecture. / T.R.G. Green, M. Petre, R.K.E. Bellamy // Empirical Studies of Programmers (4th workshop). – 1992.

141. Green, T. R. G. Using the Cognitive Walkthrough to Improve the Design of a Visual Programming Experiment / T.R.G. Green, M.M. Burnett, A.J. Ko, K.J. Rothermel, C.R. Cook, J. Schonfeld // Proceedings of 2000 IEEE Symposium on Visual Languages, Sep 10-13 2000, USA, WA, Seattle. – 2000. – P. 172-179.

142. Grundyt, J. C. Visual Language Support for Planning and Coordination in Cooperative Work Systems. / J. C. Grundyt, J. G. Hosking // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 324-325.

143. Golin, E.J. The specification of Visual Syntax. / E.J. Golin, S.P. Reiss // Proceedings of 1989 IEEE Workshop on Visual Languages, Oct 4-6 Italy, Rome,1989. – 1989. P. 105-110.

144. Golin, E.J. A Method for the Specification and Parsing of Visual Languages. PHd Thesis / E.J. Golin. - Brown University. - 1989.

145. Golin, E.J. The Visual Programmers Workbench / E.J. Golin, R.V. Rubin, J.W. Il // 11th World Computer Congress. – IFIO. – August. – 1989.

146. Golin, E.J. A Compiler Generator for Visual Languages. / E.J. Golin, T. Magliery // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 315-321.

147. Goodman, N. Languages of Art. / N. Goodman, - Hacket Publishing Co. – 1976.

148. Gurka, J.S. Testing Effectiveness of Algorithm Animation. / J.S. Gurka, W. Citrin // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 182-189.

149. Haamlev, V. A Declarative Formalism for Specifying Graphical Layout / V. Haamlev, R. Moller // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 54-59.

150. Harel, D, StateCharts: A Visual Approach to complex systems. / D. Harel // Tech Rep. CS 84-05, Weismann Institute of Science. - July 1984. -1984.

151. Hailpern, B., Multiparadigm Languages and Environments / B. Hailpern // IEEE Software. - № 3(1). - 1986. – P. 6-9.

152. Helm, R. Building visual language parsers. / R. Helm, K. Marriott, M. Odersky //Proc. ACM Conf. Human Factors in Computing, Apr.27-May 2 1991 USA, Louisiana, New Orleans. – 1991. – P. 105-112.

153. Hirakawa, M. Interpretation of Icon Overlapping in Iconic Programming / M. Hirakawa, Y. Nashimura, M. Kado, T. Ichikawa // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.- P.254-259.

154. Hirakawa, M. A Universal Language System for Visual Programming / M. Hirakawa, M. Yoshimi, T. Ichikawa // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 156-161.

155. Holt, C.M. A Visual Language Based on Functions / C.M. Holt // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 221-226.

156. Holt, C. M. An Algebra of Lines and Boxes. / C. M. Holt// Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. -1994. - P. 55-62.

157. Idini, R. Programming Web-Based Applications within a Data-Flow VL. / R. Idini, M. Mosconi, M. Porta // Proceedings of 1998 IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P. 80-81.

158. James, J. A Boundary Notation for Visual Mathematics. / J. James, W. Bricken // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. – P. 267-269.

159. Janneck, J.W. A Predicate-Based Approach for Defining Visual Syntax. / J.W. Janneck, R. Esser // Proceedings of 2001 IEEE International Symposia on Human-Centric Languages and Environments, Sep. 18-22, 2001, USA, PA, Pittsburg. – 2001. – P. 40-47.

160. Janneck, J.W. Graph-type definition language (GTDL) - specification. / J.W. Janneck // Technical Report. - Computer Engineering and Network Laboratory. - 2000.

161. Jorge, J.A.P. Online Parsing of Visual Languages Using Adjacency Grammars. / J.A.P. Jorge, E.P. Glinert // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 150-257.

162. Jungert, E. Symbolic Expressions within a Spatial Algebra: Unification and Impact upon Spatial Reasoning / E. Jungert // Proceedings of 1989 IEEE Workshop on Visual Languages, Oct 4-6 Italy, Rome,1989.– 1989. - P. 157-162.

163. Kam, N. The Immune System as a Reactive System: Modelling T Cell Activation With Statecharts / N. Kam, I.R. Cohen, D. Harel // Proceedings of 2001 IEEE International Symposia on Human-Centric Languages and Environments, Sep. 18-22, 2001, USA, PA, Pittsburg. – 2001. – P. 15-22.

164. Kelly, S. "MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment", / S. Kelly, K. Lyytinen, M. Rossi //Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080, Springer-Verlag. - 1996. - pp. 1–21.

165. Kirch, R. Computer Interpretation of English Text and Picture Patterns / R. Kirch // IEEE Trans.- EC-13,4 (Aug 1964). – P. 363-376.

166. Kim, H. BIRD: Browsing Interface for the Retrieval of Documents. / H. Kim, R.R. Korfhage // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. - P. 176-177.

167. Knuth, D.E. Semantics of context-free languages. / D.E. Knuth// Mathematical Systems Theory. - vol. 2 no 2. – 1968. – P. 127-145.

168. Knuth, D.E TEX and Metafont - New Directions in Typesetting / D.E. Knuth, -Digital Press. – 1979. – 105 pp.

169. Koifman, I. MAVIS: A Multi-Level Algorithm Visualization System within a Colloborative Distance Learning Environment / I. Koifman, I. Shimshoni, A. Tai // Proceedings of IEEE 2002 International Symposia on Human-Centric Computing Languages and Environments, Sep. 3-6 2002 USA, Virginia, Arlington. – 2002. – P.216-225.

170. Koike, Y. Improving Readability of Iconic Programs with Multiple View Object Representation / Y. Koike, Y. Maeda, Y. Koseki // Proceedings of 11th IEEE

International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 37-44.

171. Kojima, K. Parsing Graphical Function Sequence. / K. Kojima, B. Mayers // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.- p.111-117.

172. Kong, J. Graph-based Consistency Checking in Spatial Information Systems. / J. Kong, K. Zhang //Proceedings of IEEE 2003 International Symposia on Human-Centric Computing Languages and Environments, Oct. 28-31 2003 New Zeland, Aukland. – 2003. – P. 153-160.

173. Lam, V.S.W. Analyzing Equivalences of UML Statechart Diagrams by Structural Congruence and Open Bisimulations. / V.S.W. Lam, J. Paget // Proceedings of IEEE 2003 International Symposia on Human-Centric Computing Languages and Environments, Oct. 28-31 2003 New Zeland, Aukland. – 2003. – P. 137-144.

174. Lauer, T.W. Density in Scatterplots and the Estimation of Correlation. / T.W. Lauer, G.W. Post //Behaviour and Information Technology. – 1989. - № 8(3). – P. 235-244.

175. Lawrence, A.W. Empirically Evaluating the Use of Animations to Teach Algorithms. / A.W. Lawrence, A. M. Badre, J. T. Stasko// Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. P. 48-54.

176. Lee, J. Visual Translation: From Native Language to Sign Language / Lee J., T.L. Kunii // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. – 1992. – P. 103-109.

177. Leopold, J.L. Keyboardless Visual Programming Using Voice, Handwriting and Gesture / J.L. Leopold, A.L. Ambler // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. – 1997. – P. 28-35.

178. Lieberman, H. Dominoes and Storyboards: beyond “Icons on Strings” / H. Lieberman //Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. – 1992. – P. 65-71.

179. Lieberman, H. The Visual Language of Experts in Graphic Design / H. Lieberman // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 5-12.

180. Maizel, J.V. Enhanced Graphic Matrix Analysis of Nucleic Acid and Protein Sequences. / J.V. Maizel, R.P. Lenk // Proceedings of the National Academy of Science - USA 1978, №12. – P. 7665-7669.

181. Markus, A. Experiments with Genetic Algorithms for Displaying Graphs / A. Markus // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991. - P. 62-67.

182. Marriott, K. Constraint Multiset Grammars /K. Marriott // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. -P. 118-125.

183. Marriott, K. Towards a Hierarchy of Visual Languages. / K. Marriott, B. Meyer // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 196-203.

184. Marks, J. A Syntax and Semantics for Network Diagrams / J. Marks //1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 104-110.

185. Masui, T. Graphic Object Layout with Interactive Genetic Algorithms / T. Masui // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 74-80.

186. McWhirter, J.D. A Characterization Framework for Visual Languages. / J.D. McWhirter, G.J. Nutt // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 246-248.

187. McWhirter, J.D. Escalante: An Environment for the Rapid Construction of Visual Language Applications. / J.D. McWhirter, G.J. Nutt // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. -P. 15-22.

188. Meyer, B. Pictures Depicting Pictures On the Specification of Visual Languages by Visual Grammars / B. Meyer // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 41-47.

189. Myers, B.A. Visual Programming, Programming by Example, and Program Visualisation: a Taxonomy. / B.A. Myers // Proceeding of Human Factors in Computing Systems. – 1986. - P. 59-66.

190. Myers, B. The State of the Art in Visual Programming and Program Visualization. / B. Myers // Tech. Report CMU-CS-88-114. - Carnegie Mellon University, Pittsburgh, PA. - 1988.

191. Myers, B.A. Invisible Programming / B.A. Myers // 1990 IEEE Workshop on Visual Language, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 203-208.

192. Marriott, K. Constraint Multiset Grammars. / K. Marriott// Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. - P. 118-125.

193. McWhirter, J.D. AlgorithmExplorer: A Student-Centered Algorithm Animation System. / J.D. McWhirter// Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 174-181.

194. Mima, Y. A Visual Programming Environment for Programming by Example Abstraction / Y. Mima // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.–P. 132-137.

195. Minas, M. Diagram Editing with Hypergraph Parser Support / M. Minas // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. - P. 226-233.

196. Minas, M. A High-Level Visual Language for Generating Web Structures. / M. Minas, L. Shklar // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 284-285.

197. Minas, M. Specification of Diagram Editors Providing Layout Adjustment with Minimal Change / M. Minas, G. Viehstaedt // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. - P. 324-329.

198. Minas, M. DiaGen: A Generator for Diagram Editors Providing Direct Manipulation and Execution of Diagrams. / M. Minas, G. Viehstaedt // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 203-210.

199. Mussio, P. Visual Programming in a Visual Environment for Liver Simulation Studies / P. Mussio, M. Retrograde, M. Protti, F. Colombo, M. Finadri, P. Gentini // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 29-35.

200. Mussio, P. Multi-iconic Multi-Dimensional computation: a medical case. / P. Mussio, P. Bottoni, M. Protti, M. Finadri, P. Gentini // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991. - P. 47-53.

201. Najork, M.A. Specifying Visual Languages with Conditional Set Rewrite Systems. / M.A. Najork, S.M. Kaplan // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. - P. 12-18.

202. Neary, D.S. Visual Representation of Algebraic Notations: a User-Oriented Approach. / D.S. Neary, M.R. Woodward // Proceedings of 2001 IEEE International Symposia on Human-Centric Languages and Environments, Sep. 18-22, 2001, USA, PA, Pittsburg. – 2001. – P. 62-63.

203. Narasimhan, R. A Linguistic approach to pattern recognition. / R. Narasimhan // Rep. No 21, Digital Computer Lab, U. of Illinois, Urbana. – 1962.

204. Narasimhan, R. Syntax-Directed interpretation of classes of pictures. / R. Narasimhan // Comm ACM 9, 3 (March 1966). - P. 166-173.

205. Nickerson, J.V. Visual Programming: Limits of Graphic Representation / J.V. Nickerson // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. – 1994. – P. 178-179.

206. Nuchprayoon, A. GUIDO, a Visual Tool for Retrieving Documents. / A. Nuchprayoon, R.R. Korfhage // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. – P.64-71.

207. Ohki, M. A Program Visualization Tool for Program Comprehension. / M. Ohki, Y. Hosaka // Proceedings of IEEE 2003 International Symposia on Human-

Centric Computing Languages and Environments, Oct. 28-31 2003 New Zeland, Aukland. – 2003. – P. 263-265.

208. Orefice, S. A 2D Interactive Parser for Iconic Languages / S. Orefice, G. Polese, M. Tucci, G. Tortora, G. Costagliola, S-K. Chang // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 207-213.

209. Parrow, J. An Introduction to the a-Calculus. / J. Parrow //Handbook of Pmcess Algebra. - 2001. -P. 479-543.

210. Pane, J.F. Using HCI Techniques to Design a More Usable Programming System. / J.F. Pane, B.A. Myers, L.B. Miller //Proceedings of IEEE 2002 International Symposia on Human-Centric Computing Languages and Environments, Sep. 3-6 2002 USA, Virginia, Arlington. – 2002. – P.198-206.

211. Pandey, R.K. Is It Easier to Write Matrix Manipulation Programs Visually or Textually? An Empirical Study / R.K. Pandey, M.M. Bumett// Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. - P. 344-351.

212. Pereira, F.C.N. Definite Clause Grammars for Language Analysis – a Survey for the Formalism and a Comparison with Augmented Transition Networks. / F.C.N. Pereira, H.D. Warren // Artificial Intelligence. - 1980. - vol 13. - P. 231-278.

213. Petraglia, G. Towards Normalized Iconic Indexing Structures. / G. Petraglia, M. Sebillo, M. Tucci, G. Tortora // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. - P. 392-394.

214. Pfaltz, J. L. Web Grammar. / J. L. Pfaltz, A. Rosenfeld // Proceedings of the International Joint Conference on Artificial Intelligence, Washington, D.C., 1969. - P. 609-619.

215. Pfaltz, J. L. Web grammars and picture description. / J. L. Pfaltz // Computer Graphics and Image Processing. - 1972. - №1. - P. 193.

216. Pfeiffer Jr., J.J. A Rule-Based Visual Language for Small Mobile Robots. // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. – P. 162-163.

217. Pfeiffer Jr., J.J. Case Study: Developing a Rule-Based Language for Mobile Robots. / J.J. Pfeiffer Jr.// Proceedings of 1998 IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P. 204-209.

218. Pfeiffer Jr, J.J. Parsing Graphs Representing Two Dimensional Figures. / J.J. Pfeiffer Jr. // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 200-206.

219. Pratt, W.T. A Language Extension for Graph Processing and Its Formal Semantics. / W.T. Pratt, D. P. Friedman // Communications of the ACM. - July 1971, Vol. 14 Num. 7.

220. Pustell, J. A High Speed, High Capacity Homology Matrix: Zooming Through SV40 and Polyoma. / J. Pustell, F.C. Kafatos // Nucleic Acid Research №10, 15 1982. - P. 4765-4782.

221. Qiu. M.K. Spatial Graph Grammars for Web Information Transformation. / M.K. Qiu, G.L. Song, J. Kong, K. Zhang // Proceedings of IEEE 2003 International Symposia on Human-Centric Computing Languages and Environments, Oct. 28-31 2003 New Zeland, Aukland. – 2003. – P. 84-91.

222. Radyia, A. A Model of Human Approach to Describing Algorithms and Diagrams / A. Radyia, V. Radiya // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 261-263.

223. Raeder, G. A survey of current graphical programming techniques / G. Raeder // IEEE Computer. – 1985.- № 18(8). – P. 11-25.

224. Rau-Chaplin, A. A Graphical Language for Generating Architectural Forms. / A. Rau-Chaplin, T.J. Smedley // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. – P. 260-267.

225. Raymond, D.R. Characterizing Visual Languages. / D.R. Raymond // 1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.– P. 176-182.

226. Repenning, A. Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction / A. Repenning, W. Citrin // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 77-82.

227. Rekers, J. A graph-based framework for the implementation of visual environments. / J. Rekers, A. Schurr // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 148-155.

228. Rekers, J. Defining and Parsing Visual Languages with Layered Graph Grammars. / J. Rekers, A. Schürr // Journal of Visual Languages and Computing, №8(1). – 1997. – P. 27–55.

229. Roast, C. R. Formal Comparisons of Program Modification / Roast C. R., B. Khazaei, J. I. Siddiqi // Proceedings of 2000 IEEE Symposium on Visual Languages, Sep 10-13 2000, USA, WA, Seattle. – 2000. – P.165-171.

230. Rodgers, P. Visual Execution and Data Visualization in Natural Language Processing. / P. Rodgers, R. Gaizauskas, K. Humphreys, H. Gunningham // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. – 1997. – P. 338-343.

231. Rogers, G., Visual Programming with Objects and Relations. / G. Rogers // 1988 IEEE Workshop on Visual Languages, - 1988. – P. 29-36.

232. Sato, Y. An fMRI Analysis of the Efficacy of Euler Diagrams in Logical Reasoning/ Y. Sato, S. Masuda, Y. Someya, T. Tsujii, S. Watanabe //2015 IEEE Symposium on Visual Languages and Human-Centric Computing. - Atlanta, 2015. - P. 143-152.

233. Sen, T, Confidence and Accuracy in Judgements Using Computer Displayed Information / T. Sen, W.J. Boe // Behaviour and Information Technology. - № 10(1). – 1991. - P. 53-66.

234. Schneider H. J. Chomsky-Systeme fur partielle Ordnungen // Technical Report 3,3, Institut fur Mathematische Maschinen und Datenverarbeitung. - Erlangen. - 1970.

235. Selker, T. Elements of Visual Language / T. Selker, J. Koved //1988 IEEE Workshop on Visual Languages. – 1988. – P. 38-44.

236. Senay, H. Graphical Representations of Logic Programs and Their Behavior. / H. Senay, S. Lazzeri //1991 IEEE Workshop on Visual Languages, Sep. 1991 Japan, Kobe - 1991.- P.25-31.

237. Shaw, A.C. Parsing of Graph-Representable Pictures / A.C. Shaw // Journal of the ACM – 1970. – №3: Issue.17 - P .453-481.

238. Shneiderman, B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations / B. Shneiderman // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 336-343.

239. Shu, N. C., Visual Programming Languages: A Dimensional Analysis / N. C. Shu // Proceeding of the International Symposium on New Directions in Computing. – 1985. -№8. - Trondheim, Norway.

240. Shu, N. C. Visual Programming / N. C. Shu. - New York: Van Nostran Reinhold co. – 1988.

241. Sifer, M. Zooming in One Dimension can be better than Two: an Interface for Placing Search Results in Context with a Restricted Sitemap / M. Sifer, O. Liechti // Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 72-79.

242. Smedley, T.J. Graphical Parametrised Structural Descriptions of VLSI Devices. / T.J. Smedley, A.G. Jost // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. – 1993. – P. 282-286.

243. Smedley, T.J. A High-Level Visual Language for the Graphical Description of Digital Circuits. / T.J. Smedley // Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 77-82.

244. Smolander, K. "MetaEdit: A flexible graphical environment for methodology modelling" / K. Smolander, K. Lyytinen, V.-P. Tahvanainen, P. Marttiin //Proceedings of CAiSE'91, 3rd Intl. Conference on Advanced Information Systems Engineering, Springer Verlag, pp. 168–193, 1991.

245. Sommerer, C. VERBARIUM and LIFE SPACIES: Creating a Visual Language by Transcoding Text into Form on the Internet / C. Sommerer, L. Mignonneau// Proceedings of 1999 IEEE International Symposium on Visual Languages. 13-16 Sept. 1999 Tokyo, Japain – 1999. – P. 90-95.

246. Stasko, J.T. Understanding and Characterizing Software Visualization / J.T. Stasko, C. Patterson // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 3-10.

247. Stasko, J.T. Three-Dimensional Computation Visualization / J.T. Stasko, J.E. Wehrli// Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. - P. 100-107.

248. Stiles, R. Lingua Graphica: A Visual Language for Virtual Environments / R. Stiles, M. Pontecorvo // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 225-227.

249. Sugihara, K. Layout- by-Example: A Fuzzy Visual Language for Specifying Stereotypes of Diagram Layout. / K. Sugihara, K. Yamamoto, K. Takeda, M. Inaba // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 88-94.

250. Sugihara, K. An Approach to Animation of Software Specifications. / K. Sugihara, K. Takeda, M. Inaba // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. -P. 374-375.

251. Sutherland, W.R. On-line Graphical Specification of Computer Procedures. PhD thesis. / W.R. Sutherland. - MIT. - 1966.

252. Swenson, K.D. A Visual Language to Describe Collaborative Work. / K.D. Swenson// Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. –P. 315-321.

253. Suleiman, K.A. An International Visual Language / K.A. Suleiman, W.V. Citrin // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 141-147.

254. Sutherland W. On-line Graphical Specification of Computer Procedures, Ph. D Thesis, M.I.T. Cambridge, Mass. – 1966.

255. Takami, K. A Visual Programming for Telecommunication Services. / K. Takami, T. Ohta, N. Terashima // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993 . – P. 360-361.

256. Tanimoto, S.L. Introducing New Nouns in a Children's Visual Language / S.L. Tanimoto, C.L. Bernardelli // Proceedings of 1998 IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P. 74-75.

257. Tanimoto, S.L. Representation and Learnability in Visual Languages for Web-based Interpersonal Communication / S.L. Tanimoto // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. – P. 2-10.

258. Tucci, M. Parsing Non-linear Languages / M. Tucci, G. Vitiello, G. Costagliola // IEEE Trans. Soft. Eng. 20(9). - 1994. - P. 720-739.

259. Tudoreanu, M.E. Empirical Evidence that Algorithm Animation Promotes Understanding of Distributed Algorithms / M.E. Tudoreanu, R. Wu, A. Hamilton-Taylor, E. Kraemer // Proceedings of IEEE 2002 International Symposia on Human-Centric Computing Languages and Environments, Sep. 3-6 2002 USA, Virginia, Arlington. – 2002. – P. 236-243.

260. Uskudarli, S.M. Generating Visual Editors for Formally Specified Languages. / S.M. Uskudarli // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. - 1994. - P. 278-285.

261. Uskudarli, S.M. Towards a Visual Programming Environment Generator for Algebraic Specifications. / S.M. Uskudarli, T. B. Dinesh //Proceedings of 11th IEEE International Symposium on Visual Languages, Sep 5-9 1995, Germany, Darmstadt. - 1995. - P. 235-241.

262. Uskudarli, S.M. The VAS Formalism in VASE. / S.M. Uskudarli, T. B. Dinesh // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 140-147.

263. Wang, G. Applicability Checking in Visual Programming Languages / G. Wang, A. Ambler // Proceedings of 1994 IEEE Symposium on Visual Languages, Oct 1, 1994, USA, MO, St. Louis. – 1994. – P. 31-38.

264. Ware, C. Viewing a Graph in a Virtual Reality Display is Three Times as Good as a 2D Diagram / C. Ware, G. Franck // 1994 IEEE Symposium on Visual Languages. – 1994. – P. 182-183.

265. Weinreb D., Moon D. Lisp Machine Manual // Symbolics Inc. - July 1981. – 1981

266. Weitzman, L. Relational Grammars for Interactive Design / L. Weitzman, K. Wittenburg // Proceedings of 1993 IEEE Workshop on Visual Languages, Aug. 24-27, 1993 Norway, Bergen. - 1993. - P.4-11.

267. Wittenburg, K. Visual Grammars and Incremental Parsing for Interface Languages. / K. Wittenburg, L. Weitzman // 1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 111-118.

268. Wittenburg, K. Earley-style Parsing for Relational Grammars. / K. Wittenburg // Proceedings of 1992 IEEE Workshop on Visual Languages, Sept. 15-18, 1992, USA, WA, Seattle. - 1992. - P. 192-199.

269. Yang, S. Design Benchmarks for VPL Static Representations / S. Yang, E. DeKoven, M. Zloof // Proceedings of 12th IEEE International Symposium on Visual Languages, Sep 3-6 1996 USA, Colorado, Boulder - 1996. - P. 263-264.

270. Yu, B. A Fuzzy Visual Language Compiler / B. Yu, S.-K. Chang //1990 IEEE Workshop on Visual Languages, October 4-6 1990 USA, Illinois, Skokie. - 1990. – P. 162-177.

271. Zave, P. A Compositional Approach to Multiparadigm Programming / P. Zave // IEEE Software. - 1989. - №9. – P. 15-25.

272. Zhang, D.-Q. Reserved Graph Grammar: A Specification Tool for Diagrammatic VPLs / D.-Q. Zhang, K. Zhang // Proceedings of 1997 IEEE International Symposium on Visual Languages, Sep. 23-26 Italy, Capri. - 1997. - P. 260-267.

273. Zhang, D.-Q. VisPro: A Visual Language Generation Toolset. / D.-Q. Zhang, K. Zhang // Proceedings of 1998 IEEE International Symposium on Visual Languages, Sep 1-4, 1998 Canada, NS, Halifax. – 1998. – P. 88-89.

274. Zloof, M., Query-by-Example: A Database Language / M. Zloof // IBM Sys. Journal. – 1977. - № 16(4) – P. 324-343.

ПРИЛОЖЕНИЕ П1

Обоснование преимуществ визуальных языков перед текстовыми

Визуальные языки и связанные с ними технологии являются средством, которое призвано улучшить восприятие программ, а через это уменьшить количество ошибок и расширить круг людей, способных программировать [71]. Существует множество работ, посвященных психологии восприятия визуальных языков, которые могут быть подытожены следующим образом [240]:

a) Изображения более мощны, нежели слова, в плане передачи информации. Они могут нести больше смысла в меньшем выражении
b) Изображения помогают пониманию и запоминанию информации.

Эти утверждения, сделанные с точки зрения психологии восприятия, вызывают немедленные вопросы при реализации в инструментальных средствах. Например, в работе [139] на основании сравнительного анализа существовавших на то время языков, представляющих потоки вычислений, показано, что признак наиболее легко воспринимаемого языка не имеет ничего общего с его формальными свойствами. Ряд работ [140,174,233] ставит под сомнение достоинства визуальных языков, представляющих потоки данных в ряде задач, показывая, что они могут вызывать и обратный эффект – непонимание и увеличение числа ошибок пользователя. В работе [72] проведенные эксперименты показали, что символические изображения сложных процессов работы виртуальной машины не помогают лучшему их пониманию. В [229] показано, что визуальный язык может приводить к большей трудоемкости при управлении кодом. Даже сама концепция схематического изображения (т.н. “картинки-на-веревочках”[178]) подвергалась резкой критике. В [225] показано, что визуальные языки наиболее эффективно воспринимаются, когда они являются аналоговыми[147]

и работают с аналоговыми данными, и дискретность вычислительной техники и большинства понятий программирования является существенным препятствием на пути их внедрения.

Многие исследователи не оставляли попыток улучшить перцептивные качества визуальных языков. Шнайдерман [238] отмечает, что основной идеологией интерфейса визуальной системы должна быть т.н. “мантра” – “сперва общий обзор, потом приближение и фильтрация, затем детали по требованию”. Эффективной, по его мнению, может быть только система, которая поддерживает эти этапы работы пользователя. В связи с этим Шнайдерман вводит семь основных задач визуальной системы – просмотр, приближение, фильтрация, получение деталей, просмотр отношений, ведение истории и излечение по запросу. Также приводится классификация абстрактных систем по степени поддержки этих функций.

Танимото[257] высказывает идею о том, что перцептивные свойства языка зависят от таких категорий, как точность передаваемого смысла, интуитивность пиктограмм, сохранение контекста при изменении вида просмотра данных (принцип “минимального разрушения”),возможность трансляции в естественный язык и поддержка метаязыка, на котором люди могли бы описывать диаграмму, поддержка языков описания сценариев и программирования.

Группа исследователей[74] на основе теоретических знаний о пригодности программных средств к использованию, разработала ряд положений, следование которым существенно улучшает восприятие пользователем, например: “все одинаковые понятия должны отображаться одинаково”,”каждое действие пользователя должно поддерживаться визуальным объектом”, “изменение состояния объекта не должно происходить через промежуточные бессмысленные состояния”, “любые воздействия на диаграмму должны быть атомарными” и некоторые другие. Интересно отметить, что нарушение некоторых из этих положений в современной информационной индустрии является модным, например,

анимация открытия или сворачивания окон, меню и пр. Можно отметить также разработанную формальную модель “пригодности к использованию” визуального языка [75], которая объясняет то, каким образом пользователь воспринимает диаграммы.

Редер отмечает[223], что визуальные языки имеют два основных свойства, на которых необходимо сосредоточиться разработчикам: выразительность и эффективность. Преимуществами визуальных языков, по его мнению, является возможность представления нелинейных структур и процессов, использование форм изображений как выразительного средства, а также психологическая особенность восприятия человеком изображений, которые запоминаются обычно лучше, чем текст. Вместе с тем, Редер отмечает, что необходимо бороться с недостатками, такими как большие вычислительные трудозатраты, большее необходимое изображению количество пространства, отсутствие единых правил изображения и использование классов диаграмм в не предназначенных для этих классов областях.

Очевидным образом, все эти недостатки могут быть устранены. Например, ряд работ [135,222] предлагают стандартизировать элементы и команды управления способом, оптимальным с точки зрения человеческого восприятия. Некоторые работы стремятся найти новые формы представления информации, такие как ориентация в пространстве[213], структуризация [170] или трехмерность [131,247,264]. Рассматриваются подходы к визуализации, оптимальные по эффективности представления [107] и интерфейса взаимодействия с пользователем [95,226], обобщенного принятия решений о допустимости действий с диаграммой [263]. Вводятся специфичные для визуальных языков метрики [205,269]. Проводятся классификации визуальных элементов диаграмм [179]. Разрабатываются методы эффективного обучения визуальным языкам [141]. Вычислительные трудозатраты, в основном, сняты за счет развития современных компьютеров.

Вопросы визуального дизайна, и, самое главное, соответствия задачам, однако, по-прежнему являются ключевыми.

К сожалению, до сих пор не разработано удовлетворительного визуального языка общего назначения, хотя такая задача была сформулирована еще два десятилетия назад [136], зато огромное количество применений лежит в узкоспециализированных областях. Например, экспериментально подтверждено[253], что визуальные языки лучше подходят людям, слабо владеющим английским, на котором построено большинство обычных языков программирования. Визуальные языки также существенно улучшают понимание алгоритмов распределенных вычислений [259].

Некоторые авторы отстаивают противоположную точку зрения. Например, исследование [211] показало, что визуальный язык общего назначения Form/3 [83] показывает лучшие результаты в качестве решения обычных задач программирования, нежели традиционные языки, в частности, Pascal.

Таким образом, визуальные языки могут обеспечивать лучшие результаты, нежели традиционные текстовые языки, в ряде областей. При этом визуальные языки не являются универсальным средством улучшения перцептивных качеств средства решения задачи – при неправильном применении они могут ухудшить ситуацию, а не улучшить. Тем не менее, существует большое количество специализированных задач, при решении которых визуальные языки показали высокую эффективность.

ПРИЛОЖЕНИЕ П2

Обзор моделей представления визуальных языков

Исследователю, задавшемуся целью построить формальную модель визуального языка и диаграммы, прежде всего, необходимо ответить на вопрос, что же это такое. Обычно диаграмму определяют как структурированную систему связанных изображений (iconic system)[86]. Это упорядоченное множество изображений трактуют как «текст» или, скорее, «изображение» на некотором «языке», в свою очередь называемым «визуальным языком». Для визуального языка, также как и для текстового, выделяют синтаксический и семантический анализ, первый из которых отвечает на вопрос корректности построения диаграммы, а второй – смысла. В общем случае, исследователи обычно ссылаются на определение визуального языка как языка, “представляющего свои тексты двух или более мерным способом”. Кроме того, к выразительным средствам визуальных языков иногда относят цвет (в особенности для задач поиска изображений по содержимому[257]) и звук[82]. Существуют системы[177], в которых визуальное программирование осуществляется с помощью речи и ручного письма.

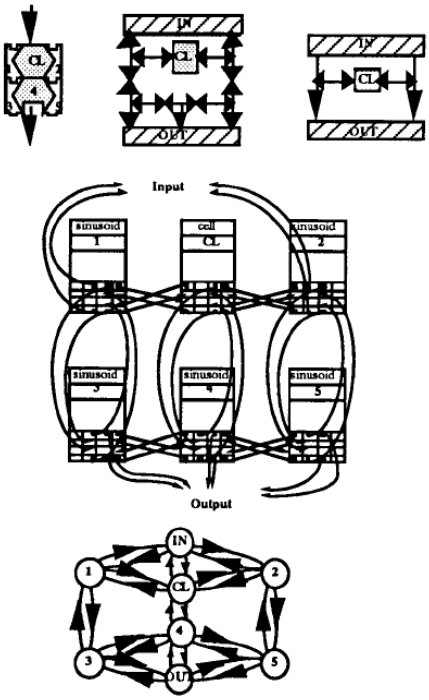


Рис. 1 Моделирование популяции клеток печени [136]

Визуальные языки используются в самых разных областях. Говоря об информационных технологиях, это, прежде всего, задачи проектирования ПО, в которых прочное место завоевали технологии UML[3], IDEF[54] и другие. Однако, визуальные языки находят применение и в областях, не обязательно связанных напрямую с программированием, так,

например, в музыке [98], медицине [66,68, 163,199,200], генетике [180,220], образовании [65, 81,138, 148,175,193,207], в том числе детей[161,256,210] и через Интернет[84,169], изобразительном искусстве [245], а также моделировании трехмерных сцен [248], поиске [113,114,115], распознавании [61] и обработке [70] изображений, поиске информации [116,166,206,241], добычи данных [69,92], иллюстрации математических выражений и теорий [78,117,158,202], телекоммуникациях[255], описаниях бизнес-процессов [142,252] и алгоритмов [80,122], проектировании устройств [242] и программного обеспечения[104], программировании устройств на низком уровне [64], цифровых схем [243], ведении архивов мультимедиа [112], веб-дизайне [196], управлении роботами[110,216,217], архитектуре[224], банковской деятельности [67],обработки текстов на естественных языках [230], управлении безопасностью [88] и даже для создания интерфейсов типа “командная строка”[134]. Можно отметить языки для программирования веб-приложений[157]. Количество же графических подходов к работе с базами данных, равно как и разнообразных расширений языка UML, не поддается исчислению – в каждой из категорий можно назвать десятки разнообразных работ.

В настоящее время существует огромное количество диаграмм и такое же количество визуальных языков. Для того, чтобы каким-то образом упорядочить их и производить анализ, существует насколько классификаций, наиболее известными из которых являются классификации Майерса [189], Шу [240] и Чанга [86].

Классификация Майерса разделяет все визуальные языки на три части в зависимости от класса задач, которые они решают. Майерс выделяет визуальные языки, предназначенные для программирования (Visual Programming), визуальные языки, предназначенные для визуализации данных (Program Visualisation) и шаблонное визуальное программирование (Program by Example). Языки, предназначенные для визуализации данных, являются наиболее известным классом языков. Они реализованы в современных Case –

средствах в виде нотаций UML[3], IDEF[54] и многих других. Все эти диаграммы изображают предметные области и обычно могут быть преобразованы в специфичные для этих областей описания, такие как тексты программ или команды базы данных. Также зачастую для них поддерживается обратное преобразование (реинжениринг).

Языки визуального программирования, начало которым было положено еще в 60х годах [251], имеют важное отличие от предыдущего класса. Диаграммы на этих языках могут исполняться так же, как обычные текстовые программы. Подобные языки имеют весьма широкое применение, например, для систем моделирования, когда система вначале изображается в виде диаграммы, а потом запускается на выполнение. Еще одним популярным применением является описание с помощью визуальных языков редакторов диаграмм, т.е. визуальные метаязыки.

Несмотря на высокую популярность языков визуального программирования, многие из которых вполне выдержали проверку временем (QBE [274], UML[3], IDEF[54] и другие), их применение обычно ограниченно узкоспециализированными областями. Еще в восьмидесятые годы ряд исследователей [79, 190] выдвинул предположение о неприменимости визуального подхода к решению обобщенных задач. Эта неприменимость объясняется тем, что с помощью изображений очень легко описывать конкретные объекты, но программирование по большей части абстрактно. В девяностые годы [137] сказывалось мнение, что основные области применения визуальных языков – параллельное программирование, а также мультимедиа-системы. Некоторые исследователи считали, что одна из основных областей применения визуальных языков - помощь слабовидящим людям. В результате исследований в этой области появились, например, такие системы поддержки слабовидящих людей, как Minspeak [109,90] или “язык визуальной трансляции” VT [176], преобразующий естественный язык в наборы изображений по принципу языка жестов. Другие исследователи не прекращали попытки создания такого языка. Например, в [73] была

предложена идея мультипарадигменного [151,271] визуального языка “Visual ToolSet”, позволяющего решать общие задачи за счет разделения на пять составляющих (парадигм) – средства определения данных, взаимодействий с базами данных, функций, потока вычислений и шаблонного программирования.

Шаблонное программирование является отдельным важным классом программирования. Это попытка построить с помощью методов искусственного интеллекта генератор, который на основании нескольких примеров использования целевой системы построит саму систему. Программирование с использованием таких генераторов позволяет уйти от традиционного труда программиста, однако же требует известного искусства и понимания технологии при выборе примеров использования. Применительно к визуальным языкам системы шаблонного программирования обычно пытаются строить генераторы диаграмм на основе нескольких примеров диаграмм. Встречаются и другие применения, например, построение интерфейсов программ (так называемых “демонстрационных интерфейсов”, в которых действия выполняются по шаблонам, заданным пользователем) [96,191]. Так же очень часто к этой технологии относят любые виды использования макросов пользователя [194].

Майерс рассматривает несколько других характеристик визуальных языков. Наиболее важным является динамический или статический вид диаграмм. Большинство известных диаграмм являются статическими (так как изначально были предназначены для рисования на бумаге), но также существуют диаграммы, визуализирующие данные, изменяющиеся со временем. Такая диаграмма может изменяться как на основании виртуального времени, определяемого системой, так и на основании данных, поступающих от внешнего интерфейса. Во втором случае такие системы применимы, например, для задач мониторинга.

Последний вид классификации, введенный Майерсом – подразделение диаграмм на интерактивные и пакетные (batch) по принципу исполнения.

Интерактивные диаграммы реагируют непосредственно на воздействия пользователя, в то время как пакетные реагируют асинхронно, по мере возможности. Типичным примером пакетной диаграммы является графический интерфейс с кнопками и меню, когда воздействия пользователя формируются в очередь сообщений, которая затем обрабатывается оконной функцией.

Классификация Шу [239] относится, скорее, к системам поддержки редактирования и исполнения диаграмм, и предлагает их оценивать по трем независимым параметрам: видимость (адекватности визуализации), уровню языка (адекватности представления процесса) и пределам языка (адекватности представления объектов для разных предметных областей). Выражаясь проще, в расчет берется то, насколько полученные диаграммы визуально соответствуют тем, которые необходимо получить, насколько работа с диаграммой отражает то, как с ней необходимо работать и насколько широкий класс применения способно поддерживать данное средство. Шу предлагает классифицировать редакторы диаграмм по этим параметрам, после чего производить поиск подходящего средства для каждой конкретной диаграммы путем ответа на три указанных вопроса и выбора средства на основе этих ответов

Классификация Чанга [86] делит все визуальные языки на четыре типа. Первый тип включает в себя языки, поддерживающие визуальное взаимодействие. Ко второму типу Чанг относит языки визуального программирования. Первые два типа обычно работают с логическими объектами, имеющими визуальное представление. Третий тип – языки обработки визуальной информации и четвертый тип – языки обработки схематической визуальной информации наоборот, работают с визуальными объектами, которые имеют логическую интерпретацию. Первый тип в общем повторяет соответствующие типы классификации Майерса. Второй, третий и четвертый типы являются в той или иной степени языками визуального программирования, ориентированными либо на программирование с

помощью логических понятий (второй тип), либо непосредственно визуальных образов (третий и четвертый). В своей работе [86] Чанг приводит несколько примеров языков, относящихся к каждой категории.

Существует некоторое количество классификационных подходов, которые используются исследователями для своих частных нужд. Так, в работе [155] приводится разделение диаграмм на активные и пассивные. К первым относятся всевозможные диаграммы, связанные с потоками данных (flowchart) [254], ко вторым – диаграммы, призванные отображать статические системы. Можно также отметить популярное разделение визуальных систем на диаграммные, в которых рисуются собственно диаграммы, и иконные, в которых используются символические изображения и отношения между ними [85]. В работе [174] языки делятся на обозначающие и аналоговые (первые должны отвечать ряду свойств, таких как непересекающиеся классы эквивалентности понятий, недвусмысленность и пр.). Классификация Кокса и Роман [108] разделяет языки по уровню абстракции представления предметной области, выделяя прямое отображение, структурное (отображается только структура), синтезированное (добавляется новая информация, полученная из имеющейся), аналитическое (выделяются только основные части) и поясняющее (добавляется новая информация, улучшающая восприятие старой). Классификация Стаско и Паттерсона [246] разделяет языки по признаку внешнего вида, абстрактности (насколько сильно внешний вид отличается от вида предметной области), анимированности и автоматичности (требуется ли и насколько сильное вмешательство программиста для создания необходимых представлений).

Таким образом, как было показано в этом разделе, визуальный язык – это обобщение понятия текстового языка на диаграммы. Для визуального языка выделяется понятие синтаксиса и семантики, а также возможно использование адаптированных технологий трансляции текстовых языков. Тем не менее, для визуальных языков характерны свойства, которые отсутствуют у текстовых языков, например, в качестве синтаксических и

семантических выразительных средств могут использоваться цвет или направление. Существует большое количество видов и классификаций визуальных языков, основной из которых является классификация Майерса, выделяющая три основных класса визуальных языков - языки визуального программирования, языки визуализации данных и языки для шаблонного программирования. В следующем разделе будет показано, каковы основные области применения визуальных языков этих видов.

Синтаксис визуальных языков

Сразу же после того, как было введено понятие визуального языка, появилась необходимость формализации подхода к описанию его синтаксиса и семантики. Существует несколько исследований на эту тему. Например, исследователи Селкер и Ковед [235] предложили следующую модель для описания визуального языка.

Визуальный язык рассматривается как технология визуального представления информации. В нем выделяется алфавит, состоящий из изображений, схематических изображений («иконок») и символов (например, символов языка). К алфавиту же относят свойства поверхностей (текстура, градиент, прозрачность и пр.) и типы рендеринга изображений (векторы, битовые карты, эллипсоидная графика и другие методы визуализации). Синтаксис языка на основе такого алфавита разделен на четыре категории: позиционный, размерный, временной и синтаксис правил. Каждая из категорий, в свою очередь, разделена на несколько мелких

Позиционный синтаксис делится на относительный (объекты расположены последовательно, на определенном расстоянии друг от друга, под определенным углом), соотносительный (объект встроен в другой объект, объект пересекает другой объект, объект соприкасается с другим объектом) и собственно позиционный (объект соединен с другим объектом, объект помечен меткой). Размерный синтаксис определяет, в основном,

относительные размеры объекта, что зачастую используется для определения порядка синтаксического разбора диаграммы. Временной синтаксис определяет реакции динамических диаграмм на изменение внутреннего времени диаграммы, а также раз общих свойств объектов диаграммы, как например, визуальные эффекты, связанные с миганием изображения. Синтаксис правил определяет алгоритмически описываемые визуальные отношения.

В дополнение к указанным категориям вводится дополнительные синтаксические понятия – синтаксис взаимодействия и структуры. Синтаксис взаимодействия определяет, каким именно образом диаграмма может взаимодействовать с пользователем. В нем выделяются три основных параметра: реакция (степень способности диаграммы реагировать на воздействия пользователя, аналогична делению диаграмм на интерактивные и пакетные Майерса), абстрактная интерактивность (способность диаграммы реагировать на воздействия пользователя в принципе; некоторые диаграммы являются статическими и не предназначены для воздействия вообще) и время реакции, определяющее с какой скоростью диаграмма реагирует на воздействия (например, диаграмма реального времени). Под структурным синтаксисом понимается возможность объединения диаграмм в комплексы, как например язык UML состоит из нескольких отдельных взаимосвязанных языков

Важным рассмотренным в работе [235] аспектом является “парадигма ввода”, а именно способы воздействия пользователя на диаграмму. Исследователи выделили следующие виды воздействия: ввод с клавиатуры, когда значения вводятся в виде текста или выбора из меню; Point and Pick (указание и выбор), когда производится выбор манипулятором из меню; Point and Move (указание и перемещение), когда манипулятором захватывается и перемещается объект и Point and Draw (указание и изображение), когда манипулятором производится рисование диаграммы. Отдельно рассматривается интерпретация движений, которая также может иметь

значение при вводе, например, в зависимости от скорости перемещения мыши диаграмма может по разному реагировать на вышеперечисленные действия.

Таким образом, работа [235] имеет важное значение, как достаточно полное описание аспектов синтаксиса диаграмм. Большинство существующих подходов к формализации визуальных языков могут быть рассмотрены в аспекте возможности поддержки указанных синтаксических элементов.

В работе [91] визуальный синтаксис рассматривается через призму операторов, заданных на визуальных токенах. Эти операторы подразделяются на операторы топологической композиции, упорядочивающие объекты относительно друг друга; операторы ограничения диаметра, ограничивающие размер; операторы несвязанной композиции, упорядочивающие объекты на основе данных, не связанных с другими объектами. С помощью примитивов и операторов создается топологическая композиция, представляющая собой кортеж из примитивных символов, составных символов, лексем и операторов упорядочивания. Интерес представляет то, что для данной синтаксической системы разработан визуальный парсер лексем, позволяющий преобразовывать изображения с известными визуальными символами в формализованные диаграммы.

Тем не менее, существуют альтернативные работы, например [184], классифицирующая синтаксические элементы сетевых диаграмм, или [153], где основную смысловую нагрузку несет наложение элементов диаграмм друг на друга. Также в [100] приводится классификация на основе поведения элементарных объектов.

С того момента, как было предложено понятие “визуального языка”, исследователи не оставляют попыток его формализовать. Существуют самые разные способы формализации: на основе текстовых и визуальных грамматик, лямбда-исчисления, шаблонов и пр. Далее эти методы будут рассмотрены подробнее.

Графовые грамматики

Графовые грамматики (ГГ) – суть попытка применения хорошо развитого формализма текстовых языков для графов. ГГ были предложены под названием «Web Grammar» (паутинные грамматики) в 1968 году исследователями Джоном Пфалцом и Азриелом Розенфельдом в их работе [214]. Общая идея «Web Grammar» заключалась в том, что к графам применим подход на основе грамматик, при этом рассматривался класс контекстно-зависимых грамматик, для которых было показано, что они могут описывать ациклические и «выпуклые» графы.

Попытка использовать текстовый формализм немедленно породила очевидные проблемы несоответствия предметных областей. Прежде всего, в отличие от текста у диаграммы нет начала, поэтому при разборе графа, даже если мы имеем правила его вывода, мы не знаем с какой его точки нужно начинать их применять. Таким образом, формализм предполагает детерминированность лишь для крайне узкого и практически не встречающегося в жизни класса языков.

Второй очевидной проблемой является многомерность графа. Обычный текст всегда одномерен. Наиболее распространенный формализм Машины Тьюринга предполагает интерпретацию текста, записанного на ленте. Одномерный граф является простейшим случаем диаграммы и в реальной жизни встречается крайне редко. Обычно элементы («терминальные символы» или «терминальные узлы») диаграммы вступают в отношения типа «слева», «справа», «сверху», «снизу», «под углом X к» и так далее. Кроме того, в большинстве диаграмм возможны циклические зависимости, не характерные для текстовых языков.

Последней важной особенностью диаграмм является то, что они несут достаточное количество нагрузки, не связанной со смыслом. Количество элементов, определяющих смысл изображения, обычно значительно меньше, чем количество элементов, направленных на восприятие пользователем,

однако важны и те, и другие. Вместе с тем описание визуальной нагрузки с помощью формализма ГГ представляется существенно избыточным, так как его не надо транслировать, зато на описание потребуется изрядное количество правил

Самой ранней из известных автору работ, связанных с описанием диаграмм формальными текстовыми языками, является работа Алана Шо (Shaw, 1970), выпущенная в 1970 году. Эта работа являлась развитием направления исследований [119,120,165,203,204], связанных с распознаванием рукописного текста с помощью визуальных языков. В работе [237] автор предложил при распознавании рукописного текста выделять в полученном изображении шаблоны (которые могут иметь произвольную форму, но при этом обязательно содержат две точки, называемые головой и хвостом, с помощью которых они могут соединяться друг с другом) и связывать их пространственными отношениями [235]. Полученный результат может быть записан в виде текста на введенном автором языке PDL (Picture Definition Language, язык описания изображений). Синтаксис языка автор описывает так:

$S \rightarrow p \mid S \varnothing S \mid (\sim S) \mid SL \mid (/SL)$

$SL \rightarrow S^{l} \mid (SL \varnothing SL) \mid (\sim SL) \mid (/SL)$

$\varnothing \rightarrow + \mid x \mid - \mid *$ (операции отношения примитивов: голова к хвосту, соединение хвостами, соединение головами, цикл)

$p \rightarrow$ примитивный класс

$l \rightarrow$ метка

В работе показано, что использование подобной техники позволяет преобразовать изображения в достаточно простой язык, который затем можно интерпретировать, предложены алгоритмы распознавания и анализа полученных графов. Можно отметить, что полученный язык является исключительно простым и, конечно же, не приспособленным для широкого класса диаграмм. В дальнейшем целый рад исследователей пытался применить текстовые языки к описанию диаграмм [89,162,203,231]. Видимо,

основной причиной подобной популярности является возможность применения к задаче хорошо формализованного аппарата текстовых языков.

Несмотря на вышеописанный анализ синтаксиса, для большинства формальных подходов к анализу визуальных языков характерно рассмотрение диаграмм без учета конкретных координат ее элементов. Одни из лучших описаний этого подхода находятся в [134,227,228]; подход заключается в том, что каждая диаграмма представляется в виде двух графов: абстрактного графа синтаксиса (АГС) и пространственного графа отношений. Несмотря на свое название, АГС несет, прежде всего смысловую нагрузку, так как выделяет только те отношения между объектами диаграммы, которые имеют смысл для анализа. Пространственный граф отношений предназначен для восстановления по АГС собственно изображения. Он также указывает на синтаксис диаграммы, но уже в терминах [235], т.е. с той или иной степенью упрощения позволяет рисовать конкретные диаграммы.

Несмотря на высокую сложность разбора визуальных языков по сравнению с текстовыми, графовые грамматики быстро завоевали популярность. Очевидно, прежде всего это было обусловлено соблазном использования отлично проработанного формального аппарата текстовых языков для диаграмм. Естественным шагом на этом пути явилось появление альтернативной [119] классификации Хомского [94] для визуальных языков и появление различного рода классов грамматик и алгоритмов, предназначенных для их разбора. Как и для текстовых языков, авторы алгоритмов обычно стараются ограничиться контекстно-свободными языками, в крайнем случае – подмножеством контекстно-зависимых языков. Некоторые классы визуальных грамматик будут рассмотрены ниже подробнее.

Грамматики упорядочивания изображений

Грамматики упорядочивания изображений (Picture Layout Grammars, PLG) были предложены в 1989 году исследователями Эриком Голином и Стивеном Рейсом [143] и являются вариантом контекстно-свободных грамматик [94]. Работа является расширением модели атрибуированных мультимножественных грамматик (Attributed Multiset Grammar, AMG), предложенных Эриком Голином ранее [144]. Мультимножественная грамматика близка к контекстно-свободной грамматике, за исключением того, что в правой части правил вывода содержатся неупорядоченные наборы символов. В результате с помощью таких продукций может быть выведено множество мультимножеств (т.е. множеств с возможно повторяющимися элементами). Атрибуированная грамматика [167] - грамматика, расширенная атрибутами, необходимыми для разбора. Таким образом, в AMG продукция представляет собой набор (R,S,C) где

R – продукционное правило вида N→M, где N – нетерминальный символ и M – мультимножество символов

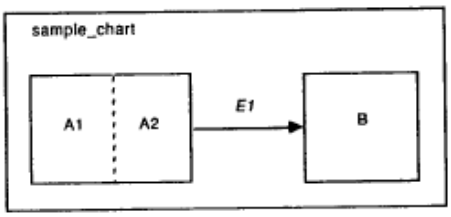
S – множество семантических функций, отображающих атрибуты из правой части продукции в атрибуты из левой части продукции

C – множество ограничений, заданных на атрибутах в правой части продукции.

PLG, являясь расширением AMG, добавляет в нее дополнительные свойства. В этих грамматиках атрибуты представляют собой информацию о пространственном расположении элементов диаграммы. Каждый символ содержит в себе четыре атрибута: lx, by, rx и ty, описывающие прямоугольную область, занимаемую объектом (lx – левая координата по x, rx – правая, by – нижняя по y, ty – верхняя по y). Дополнительные атрибуты могут определять цвет, толщину линий, градиент, стиль и пр. Шестнадцать различных семантических функций (поверх, слева, прилегает, содержит и другие), используя эту информацию, соединяют символы в единый визуальный объект.

В качестве примера в [143] разбирается диаграмма состояний [110]. Грамматика для поддержки этой диаграммы с помощью PLG состоит из 29 продукционных правил. Пример диаграммы и дерева разбора с помощью грамматики приведен на Рис. 2.

(13) STATE0 → contains(**rectangle**,STATE_INSIDE)
(14) STATE → STATE0
STATE.pos = 0
(15) STATE → touches_L(LABELLED_ARC,$STATE_1$)
STATE.pos = hash(LABELLED_ARC)
Where
$STATE_1$.pos < hash(LABELLED_ARC)
(16) LABELLED_ARC → points_to(LABELLED_ARROW,STATE0)
(17) LABELLED_ARROW → labels(**text**,**arrow**)
(18) DEFAULT → DEF_STATE
(19) DEFAULT → touches_L(UNLBLD_ARC_HIST,DEF_STATE)
(20) DEF_STATE → touches_L(UNLBLD_ARC_STATE,DEF_SYMBO
(21) DEF_SYMBOL → contains(**circle**,**text**)
Where
text.sval == "D"
(22) HISTORY → contains(**circle**,**text**)
Where
text.sval == "H"
(23) UNLBLD_ARC_HIST → points_to(**arrow**,HISTORY)
(24) UNLBLD_ARC_STATE → points_to(**arrow**,STATE0)
(25) XOR_GROUP → STATE
XOR_GROUP.default = 0
XOR_GROUP.history = 0
(26) XOR_GROUP → DEFAULT
XOR_GROUP.default = 1
XOR_GROUP.history = 0
(27) XOR_GROUP → HISTORY
XOR_GROUP.default = 0
XOR_GROUP.history = 1
(28) XOR_GROUP → adjacent_to(XOR_GROUP$_1$,XOR_GROUP$_2$)
XOR_GROUP.default = XOR_GROUP$_1$.default
+ XOR_GROUP$_2$.default
XOR_GROUP.history = XOR_GROUP$_1$.history
+ XOR_GROUP$_2$.history
Where
XOR_GROUP$_1$.default + XOR_GROUP$_2$.default ≤ 1
XOR_GROUP$_1$.history + XOR_GROUP$_2$.history ≤ 1



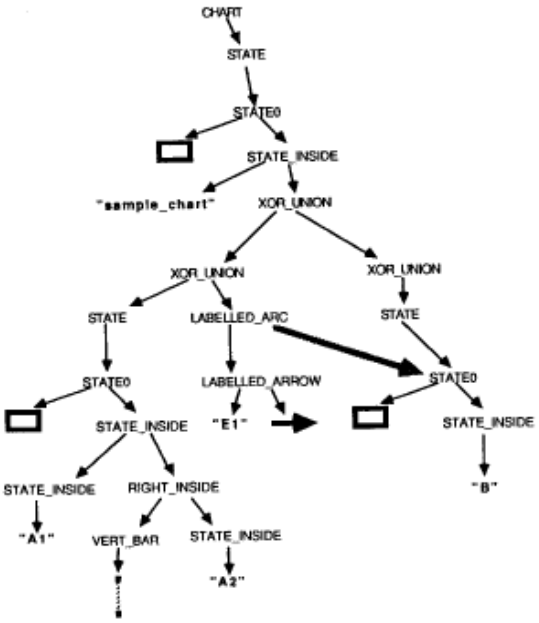


Рис. 2 Фрагмент грамматики, пример диаграммы состояний и соответствующее им дерево разбора

Авторы позиционируют их работу как математическую основу средства генерации визуальных трансляторов, на основе которой построено промышленное программное средство [145]. Кроме того, имеется объектное расширение, для которого реализован генератор компиляторов [146].

Расширенные позиционные грамматики

Расширенные позиционные грамматики (Extended Positional Grammars. XPG) впервые были предложены в работах [101,103] и являются расширением более раннего формализма позиционных грамматик [129, 258]. В данном подходе диаграммы рассматриваются как визуальные предложения,

состоящие из визуальных символов. Обобщенным визуальным символом называется такая тройка <M,S,L> в которой M указывает на изображение символа (форму, цвет, и т.п.), S – на взаимодействие с другими символами и L – на семантику. Например, M может представлять собой прямоугольник, S содержать ссылки на входящие и выходящие связи, а L указывать, что этот прямоугольник является элементом диаграммы состояний. Обобщенный алфавит является множеством обобщенных символов. Обобщенное предложение является множеством “инстанцированных” символов, принадлежащих обобщенному алфавиту, т.е. для которых полностью определены S и L (в этом месте авторы XPG допускают двусмысленность, так как формальные различия между символами алфавита и “инстанцированными” символами алфавита нигде не оговариваются, предполагается, что они интуитивно понятны).

Согласно XPG, предложения на визуальном языке формируются путем установления отношений S между визуальными символами языка. Различают два вида отношений: отношения соединения, указывающие на то, что два объекта соприкасаются какими-либо областями, и пространственные отношения, задающие отношения между координатами объектов (выше, ниже, перекрываются и т.п.). Расширенная позиционная грамматика определяется как пара (G, PE), где PE – “позиционный вычислитель”, а G – контекстно-свободная аттрибуированная грамматика (N, T$\cup$POS, S,P), где

N – конечное непустое множество нетерминальных символов

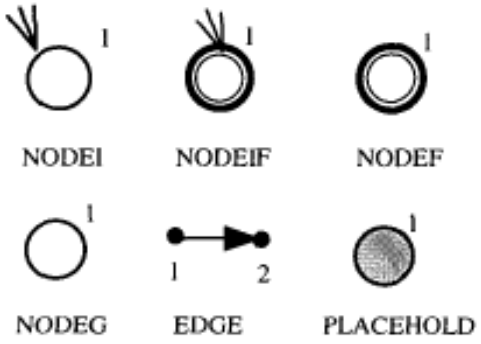
T – конечное непустое множество терминальных символов, непересекающееся с N

POS – конечное множество идентификаторов бинарных отношений, не пересекающееся ни с N, на с T

S – начальный символ грамматики, S$\in$N

P – конечное непустое множество продукций вида $A \rightarrow x_1R_1x_1R_2\ldots x_{m-1}R_{m-1}x_m$ Δ,Γ где $A \in N$, $\forall i\ x_i \in T \cup N$, а R_k представляет собой множество из двух типов отношений, входящих в POS, ведущих и тестирующих.

Ведущие отношения используются для синтаксического анализа полученного предложения. Δ представляет собой функцию, синтезирующую атрибуты для A и Γ –



(1) STDiagram → Graph
(2) Graph → NODEI
Δ: ($Graph_1$ = $NODEI_1$)
(3) Graph → NODEIF
Δ:($Graph_1$ = $NODEIF_1$);
(4) Graph → Graph' ⟨⟨*1_1*⟩,⟨*1_2*⟩⟩ EDGE *2_1* Node
Δ: ($Graph_1$ = Graph'$_1$ - $EDGE_1$),
Γ: { (PLACEHOLD; |$Node_1$|>1;
$PLACEHOLD_1$ = $Node_1$ - $EDGE_2$)}
(5) Graph → Graph' ⟨⟨*1_1*⟩,⟨*1_2*⟩⟩ EDGE
Δ: ($Graph_1$ = (Graph'$_1$ - $EDGE_1$) - $EDGE_2$);
(6) Graph → Graph' ⟨⟨*1_2*⟩,⟨*1_1*⟩⟩ EDGE *1_1* Node
Δ: ($Graph_1$ = Graph'$_1$ - $EDGE_2$),
Γ: { (PLACEHOLD; |$Node_1$|>1;
$PLACEHOLD_1$ = $Node_1$ - $EDGE_1$)}
(7) Graph → Graph' ⟨*any*⟩ PLACEHOLD
Δ: ($Graph_1$ = $PLACEHOLD_1$);
(8) Node → NODEG
Δ: ($Node_1$ = $NODEG_1$);
(9) Node → NODEF
Δ: ($Node_1$ = $NODEF_1$);
(10) Node → PLACEHOLD
Δ: ($Node_1$ = $PLACEHOLD_1$);

Рис. 3 Описание конечного автомата в XPG

множество троек вида (N,Cond, Δ), где N – терминальный символ, Cond – условие помещения его в предложение и Δ - генератор атрибутов. Множество Γ позволяет динамически добавлять новые терминальные символы в предложения. Наконец, позиционный вычислитель – абстракция, описывающая преобразователь диаграммы в формальный вид (т.е. выделение абстрактного графа синтаксиса).

На Рис. 3 приведен пример описания диаграммы, представляющей конечный автомат, в терминах XPG. В своих работах авторы в основном рассматривают применение XPG для представления различного рода автоматов и диаграмм состояний.

Важной особенностью позиционных грамматик является их хорошая теоретическая проработка. В частности, в работе [99] рассматриваются формальные механизмы разбора графов. Несомненным преимуществом данного подхода является возможность использования генератора компиляторов Yacc и модифицированного алгоритма Эрлея[118], продемонстрированная в [105], а также алгоритм LALR разбора, продемонстрированный в [208].

Грамматики графов соединений

В работе [63] предлагается модель “грамматики графов соединений” (connection graph grammar) как абстрактной модели для параллельных вычислений.

В качестве требований к абстрактной машине, производящей параллельные вычисления, автором выдвинуты следующие положения:

Модель должна соответствовать реальному оборудованию (т.е. поддерживать только те операции, которые умеет выполнять реальное оборудование).

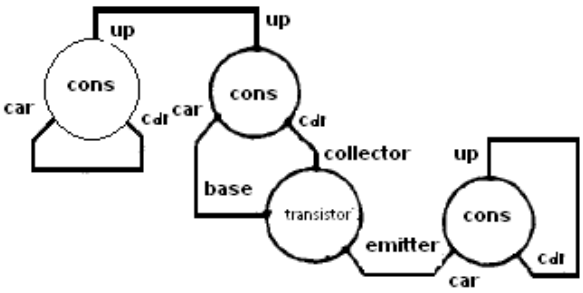


Рис. 4 Пример графа соединений - транзистор

Модель должна быть простой.

Модель должна поддерживать простую трансляцию в общепринятые языки.

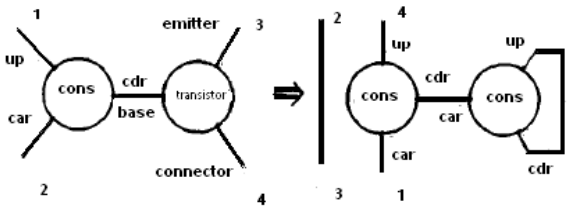


Рис. 5 Пример продукционного правила

Граф соединений во многом похож на электрическую цепь (Рис. 4). В нем выделяются узлы, которые содержат входы и выходы (терминалы), которые могут быть соединены с другими терминалами других узлов, причем каждый терминал должен быть соединен в точности с одним терминалом. Количество терминалов у узла называется “валентностью”. Тип терминала называется его меткой. Узлы с терминалами, помеченными метками, образуют алфавит языка.

Грамматика графов соединений образуется с помощью графовых продукционных правил, называемых методами (Рис. 5). Каждый метод описывает, как можно заменить некоторый фрагмент графа другим фрагментом графа. Рассматривается также возможность наличия нетерминальных символов в таком языке, однако только в качестве предложения.

Автор показывает, что подобный язык может описывать тексты программ на чистом языке Lisp [265]. Исполнение таких программ заключается в последовательном применении правил трансформации графа. Таким образом, предложенный формализм имеет прикладное подтверждение.

Реляционные грамматики

Реляционные грамматики [111] являются контекстно-свободными грамматиками, которые строятся на основе мультимножеств объектов, соединенных “реляционными сущностями” (relational instances). Они являются развитием идеи грамматик обработки изображений Чанга [87]

Определение.

Предложение w, заданное на алфавите терминальных символов VT и множестве реляционных символов VW является парой (T,RT) где T – мультимножество символов из VT а RT –множество реляционных сущностей из VW заданных на T.

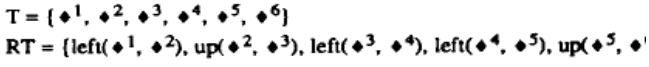


Рис. 6 Пример предложения для реляционной грамматики

Определение.

Контекстно-свободной реляционной грамматикой G называется кортеж G=(VN, VT,VR,S,P,R), где VT – множество терминальных символов, VN - нетерминальных символов, VR - множество реляционных символов, для каждого их которых указано количество мест, S - начальный символ грамматики P - конечное множество продукций в виде A::=M, N, где A – нетерминал, а N – ограничение (аналогично аттрибуированной грамматике [167]) и R – множество правил вычисления, используемых для обработки ограничений.

Таким образом, можно сделать вывод, что данная грамматика является несколько модифицированной под задачи визуальных языков аттрибуированной грамматикой [167] . Для грамматики разработан алгоритм разбора [128] с реализацией на языке Пролог. Позднее был предложен общий алгоритм разбора[268267] на основе алгоритма Эрлея[118], что делает подход приголным для любых контекстно-свободных визуальных грамматик, а также показано[102], что для построения парсера языка можно использовать систему YACC. Важным результатом является показанная в [266] применимость грамматик к задачам интерактивного дизайна, в частности, автоматического размещения объектов и проверки допустимости трансформаций. Также для некоторых подклассов контекстно-свободных реляционных грамматик, разработан упрощенный алгоритм разбора [129].

Развитием данного формализма являются символ - реляционные грамматики [130], в которых алфавит разделяется на терминалы,

представляющие объекты и терминалы, представляющие отношения и, соответственно, это разделение учитывается в продукционных правилах.

Графические функциональные грамматики

Майерс, автор известной классификации [189], является также соавтором графического формализма, известного как графическая функциональная грамматика [171]. В этом формализме алфавит состоит не из терминалов, а из функций, параметрами которых являются значения атрибутов. Например, если символ алфавита обозначает окружность, то он может представляться функцией, рисующей окружности, с параметрами – координатами центра и радиусом. Эта идея была первоначально высказана для естественных языков [212], однако нашла применение и для визуальных.

Графической функцией в данном формализме называется n-местная функция, рисующая графическое изображение. Графический алфавит – конечное множество графических функций. Графическое предложение – конечная последовательность графических функций из алфавита, формальным параметрам которых присвоены конкретные значения. Графическая функциональная грамматика представляет собой кортеж из графического алфавита, набора нетерминалов, стартовой функции и правил вывода, при этом правила вывода содержат ограничения в виде предикатов первого порядка. Таким образом, данная грамматика является аттрибуированной[167].

К сожалению, данный подход не свободен от проблем. На приведенном в [171] примере видно, что грамматика чувствительна к порядку графических символов, даже если сама диаграмма рассматривает их как равнозначные. Эта проблема решается за счет преобразований графических выражений. Грамматики, которые допускают такие перестановки без изменения результата, называются коммуникативными. Преобразование не всегда является тривиальной задачей, в особенности для языков, в которых

операторы могут иметь побочные эффекты. Тем не менее, доказывается, что для любого графического языка существует “коммуникативная” грамматика.

Для данного формализма разработан алгоритм разбора, дающий решение за $O(N^2)$ шагов. Таким образом, графические функциональные грамматики являются законченным формализмом для поддержки синтаксиса визуального языка.

Обобщенные двумерные контекстно-свободные грамматики

Обобщенные двумерные контекстно-свободные грамматики [218] (2D-CFG) являются развитием позиционных грамматик и отличаются тем, что в них присутствуют только два позиционных отношения – “выше” и “левее”.

Обобщенная 2D-CFG является кортежем $G = (E, N, P, S)$, где

C – множество терминальных символов,

N – множество нетерминальных символов,

P – множество продукций в форме $n \rightarrow r\ (\ a\ )$ где $n \in N$, $r \in R = \{$слева, сверху$\}$, и $a \in (N \cup E)^*$.

$S \in N$ - начальный символ грамматики.

Для грамматики выделяется нормальная форма, в которой соблюдается правило, что продукция имеет форму $n \rightarrow (n1\ r\ n2)$ или $n \rightarrow \sigma$ где $nl, n2 \in N$, $r \in R$, и $\sigma \in C$. Для грамматики разработана модификация алгоритма Кока-Казами-Янгера (CYK), что делает возможным разбор любого контекстно-свободного языка. Таким образом, применив существенные упрощения по отношению к

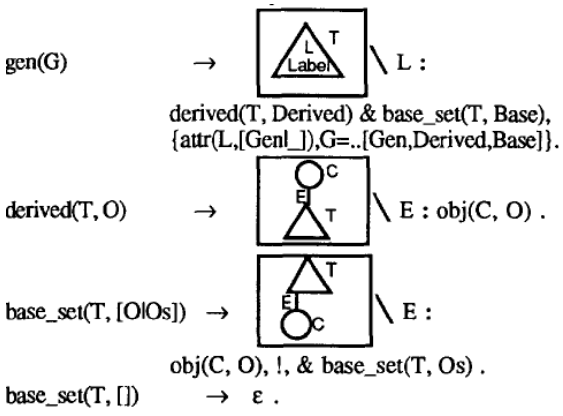


Рис. 7 Фрагмент описания ER-диаграммы с помощью PCG

возможностям диаграммы, был получен универсальный визуальный формализм.

Грамматики графических предложений

Грамматика графических предложений (Pictorial Clause Grammars, PCG) [188] является первым формальным метадиаграммным языком. При данном подходе изображения рассматриваются как символы некоторого логического языка. По определенным правилам может происходить унификация изображений и связывание графических переменных. Одновременно с этим, задается визуальная (контекстно-свободная) грамматика, в которой в качестве терминальных символов выступают изображения. С помощью атрибуирования грамматики на упомянутом выше языке, достигается возможность исполнения графических программ, описывающих другие графические языки.

Гиперграфовая грамматика

Гиперграф является обобщенным графом, в котором связь (гиперребро) может соединять произвольное количество вершин [197]. Формализм гиперграфовых грамматик строится на обычных контекстно-свободных визуальных грамматиках, атрибуированных с помощью неравенств. Грамматики содержат в себе терминальные и нетерминальные гиперребра, а также списки узлов, к которым они относятся. Продукционное правило содержит в левой части гиперребро и список вершин, а в правой –

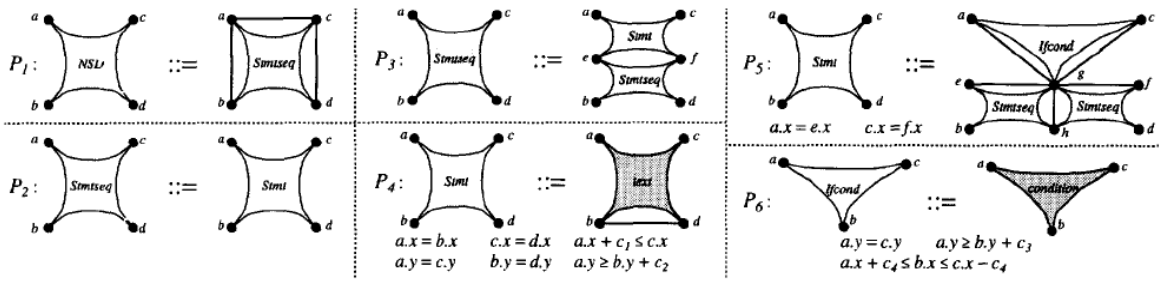


Рис. 8 Гиперграфовая грамматика для нотации Насси-Шнейдермана

произвольный гиперграф, возможно, также содержащий в себе гиперребра. Одно из гиперребер выделяется в начальный граф языка. Неравенства, с помощью которых атрибуированы продукционные правила, описывают правила пространственно упорядочивания узлов и гиперребер. Таким образом, принципиальное отличие от других формализмов заключается лишь в методе автоматического расположения объектов на экране. На основе данного формализма создан генератор диаграммеров DiaGen[195, 198].

Грамматика условной подстановки множеств

Система условной подстановки множеств для визуального языка (Conditional Set Rewriting System, CSRS) предложена в [201]. Грамматика представляет собой набор продукционных правил, аттрибуированных хорновскими условиями. Важным отличием данного формализма является возможность представления в левой части продукционных правил произвольного количества терминальных и нетерминальных символов. С другой стороны, CSRS вводит упорядочивание для продукционных правил. Грамматика предназначена для задач преобразования визуальных нотаций и трансляции графических изображений в текст.

Грамматики условных мультимножеств

Грамматика условных мультимножеств (constrained multiset grammar) была впервые неформально предложена в [152] и позже формально описана в [182]. При этом подходе нетерминальным символам грамматики присваиваются атрибуты. В качестве атрибутов продукционного правила рассматриваются условия существование определенных нетерминальных символов, атрибуты которых удовлетворяют условию применимости продукционного правила. Результатом разбора грамматики является логическая программа, составленная из этих условий. В данном формализме в левой части также может стоять только один нетерминал. Для данной

грамматики показано [182], что ее разбор выполняется за полиномиальное время. Для данного класса грамматик существует генератор компиляторов [93].

Подход также интересен тем, что для него разработана альтернативная Хомскому классификация грамматик [183] в зависимости от сложности разбора данной грамматики.

Графическая условная грамматика

Графическая условная грамматика (graphics constraint grammar) является формализмом, основанном на текстовом преставлении графов [132,133]. В этом формализме продукционное правило состоит из левой части, правой части и “тела”. Тело продукции представляет собой набор условий, в том числе геометрических отношений между объектами. Эти условия могут накладываться на широкий спектр свойств графических объектов, включая координаты, размеры и пр. а так же могут обращаться к свойствам множеств объектов. Например, правило может требовать, чтобы группа объектов была выровнена по горизонтали. Особенностью формализма является то, что правая часть, фактически, содержит логически не связанное множество объектов, отношения между которыми выстраиваются на основе условий. Естественно, что за счет такого упрощения структуры языка достигается существенное увеличение скорости разбора грамматики.

Грамматики смежности

Грамматики смежности (adjacency grammars) [161] расширяют формализм грамматик упорядочивания изображений, рассмотренных ранее. Основным их отличием является вид продукции, представимой в форме

$$\propto \rightarrow \{\beta_1 \cdots \beta_n\}\ if\ \Gamma\ (\beta_1 \cdots \beta_n)\ after\ \Delta$$

В этом продукционном правиле атрибут состоит из двух частей. Сперва выполняется часть после after, предназначенная для вычисления значений других атрибутов. Затем вычисляется собственно условие Г. Условие содержит в себе “операторы смежности”, которые определяют отношения, в том числе как пространственные, так и смысловые, между терминалами и/или нетерминалами. Для грамматики существует алгоритм разбора со сложностью O(n), который, однако, применим не ко всему классу контекстно-свободных языков.

Резервированные графовые грамматики

Резервированные графовые грамматики (Reserved Graph Grammars) являются одним из немногих представителей контекстно-зависимых грамматик. [272]. Этот формализм был разработан с целью упрощения инкрементарного разбора диаграмм. В данном формализме граф представляется в двухуровневом виде. Граф состоит из узлов, соединенных ребрами. Каждый узел содержит в себе вершины, соединенные друг с другом дугами. Некоторые “внешние” вершины, содержащихся в узле, называются “супервершинами” и представляют узел для операций логического соединения. Продукционные правила в этом формализме представляют собой правила трансформации одного графа в другой. Граф в левой части содержит узлы, помеченные специальными идентификаторами, которые используются при сопоставлении при трансформации графов. При этом продукционное правило симметрично, т.е. может быть применено в обе стороны. Сопоставление фрагментов графа с продукционными правилами осуществляется на основе меток вершин,

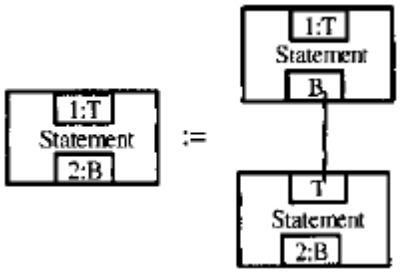


Рис. 9 Продукционное правило для резервированных грамматик

входящих в его узлы. Для данного формализма разработан алгоритм разбора за полиномиальное время, что выгодно отличает его от других контекстно-зависимых графовых грамматик. Также авторами разработан общий, т.е. пригодный и для других классов грамматик, алгоритм проверки корректности пространственного расположения со сложностью O(n4) [172]

Для данного формализма разработан визуальный аналог Yacc – VisPro[273].

Развитием данного формализма является пространственная графовая грамматика[221], которая содержит расширения, предназначенные для поддержки информации о пространственном расположении узлов. Эта информация бывает трех видов – направление, топология и взаиморасположение. Основным механизмом является матрица 3 на 3 поля, представляющая стороны (север, юг, запад, восток и промежуточные), сопоставленная с супервершиной. Центральное поле представляет саму супервершину, сопоставление с каким-либо из оставшихся полей указывает на относительное расположение объектов. За счет вступления в отношения с различными элементами этой матрицы можно указать взаимное расположение, перекрывание и направление. Продукционные правила также могут использовать трансформации этой матрицы для реализации операций, связанных с перемещением объектов.

NCE – грамматики

NCE (Neighborhood Controlled Embedding) – грамматики[59,60] являются формализмом, рассчитанным на поддержку контекстно-зависимых грамматик. Основным отличием данного формализма является вид продукционных правил. Они представляются в виде ((A;X) ::= (B; Y); C), где A и B – некоторые графы, а X и Y – соответственно, некоторые их подграфы. C – множество отношений соединения. Четыре графа A, X, B, и Y удовлетворяют условию A = X and B = Y либо существует такой подграф K(называемый контекстом) что K = A − X = B − Y. Смысл продукционного

правила заключается в нахождении в графе подграфа B и последующей замене подграфа X на подграф Y с одновременной установкой отношений C. Для данного формализма показана его применимость к нескольким визуальным языкам, а так же разработан генератор диаграммеров GRAVIS.

Функциональный подход

Вероятно, первой работой, в которой делается попытка представить граф как применение операции добавления узла к существующему графу, является работа [219]. В этой работе был предложен язык Graspe, являющийся, фактически, библиотекой Lisp, и позволяющий конструировать графы с помощью формальных операций. Все операции разделены на три типа: ссылочные (возвращающие часть графа), конструирующие (добавляющие узлы и дуги в граф) и разрушающие (удаляющие узлы и дуги из графа). Всего в работе вводится шестнадцать ссылочных, одиннадцать разрушающих и пять конструирующих операций.

Работа также интересна попыткой предложить формальную семантику полученного языка.

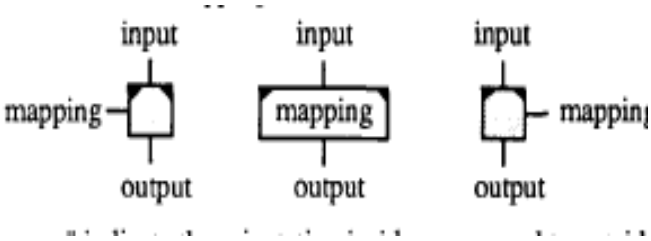
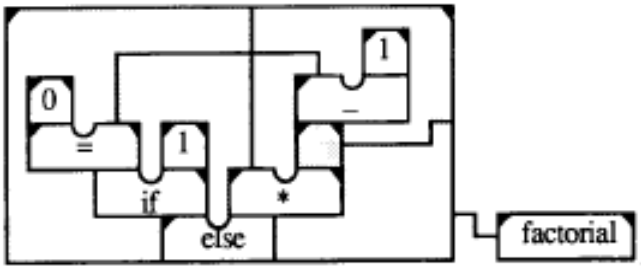


Рис. 10 Основные конструкции языка Viz

Другим исследователем, предложившим обратный подход – визуализацию функциональных языков, был Крис Холт [41,88]. В своей работе он рассматривал язык Viz, предназначенный для визуализации лямбда-выражений. Лямбда - применения в этом языке



factorial(n) = (1 if n=0) else n*factorial(n-1).

Рис. 11 Вычисление факториала на языке Viz

представляются как прямоугольники, наверху которых находится входной аргумент, внизу – результирующее выражение, а сбоку, либо собственно в прямоугольнике, применяемое отображение. Для правильной ориентации выражения в прямоугольниках обозначен верх (закрашенными уголками). Язык поддерживает также значительно более сложные выражения, такие как условия и циклы. Пример вычисления факториала на языке Viz приведен на Рис. 11.

Формальный метод описания визуальных языков с помощью лямбда – исчисления предложен Мартином Эрвигом [122,123,124]. Эрвиг вводит понятие индуктивного графа. В этом случае граф рассматривается в стиле алгебраических типов данных, заданных на таких языках, как ML или Haskell: граф может быть пустым, или может быть сконструирован из графа g и нового узла v, с рёбрами, соединяющими v с узлами из g. Здесь рассматривается число визуальный язык, то есть язык, в котором не используются метки. Таким образом, мы можем определять графовые выражения, используя оператор Empty, т.е. пустой граф, и оператор добавления узла в граф N (источники_соединений, идентификатор_узла, приёмники_соединений), где источники соединений – узлы, выходящие из которых дуги соединяют узлы из v (некоторый граф, к которому применяют этот оператор), приёмники соединений – узлы, в которые входят дуги, выходящие из v, и идентификатор узла – некоторый уникальный описатель, за которым может следовать метка (например, d:@). Эти три параметра называются также контекстом узла. В частности, символ [d›fun, e] при обозначении контекста узла указывает на список из двух узлов-источников d и e, причём ребро, выходящее из d, имеет метку fun, а ребро, выходящее

из e, метки не имеет. Точно также [par›a, body›a] обозначает узел-приёмник a, в который входят два разных ребра.

В качестве иллюстрации Эрвиг использует язык VEX. VEX [97,126]

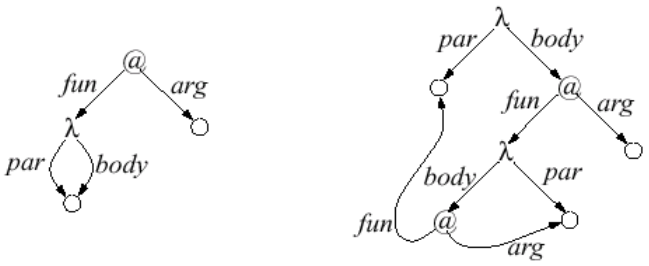


Рис. 12 Абстрактный граф синтаксиса языка VEX

является чисто визуальным языком – каждый идентификатор в нём представляется в виде окружности, которая соединена прямой линией с так называемым корневым узлом. Корневой узел является ещё одной окружностью, которая соединена с одной или несколькими линиями, ведущими ко всем идентификаторам с одним именем. Корневой узел может быть вписан в другую окружность, тем самым, указывая на параметр некоторой абстракции, в противном случае он описывает свободную переменную. Абстракция может иметь также выражение, описывающее её тело. Применение выражения изображается двумя касающимися друг друга окружностями со стрелкой в точке касания. Острие стрелки лежит внутри аргумента, а хвост находится внутри применяемой абстракции. Порядок применения может задаваться путём пометки стрелок их приоритетами, что в наших примерах будет для простоты игнорироваться.

На Рис. 12 представлен абстрактный граф синтаксиса языка VEX, заданный следующим выражением:

N ([], d:@, [fun›b, arg›c]) (N ([], c, []) (N ([], b:λ, [par›a, body›a]) (N ([], a, []) Empty)))

Здесь a,b,c и d являются произвольными, попарно различными идентификаторами узлов. В представлении синтаксиса содержатся также понятия применения, абстракции и узла-переменной, которые обозначены символами @, λ и O. Символами fun и arg на дугах обозначены применения функций и аргументов соответственно. Символами par и body обозначены параметры и тело абстракции.

В продолжение будем использовать следующие два сокращения: будем исключать пустые последовательности, и суперпозиция нескольких N-операторов может быть заменена на N* оператор. Тогда описание графа можно упростить:

N* (d:@, [fun›b, arg›c]) (c) (b:l, [par›a, body›a]) (a) Empty

Как обычно для подобных систем, один и тот же граф могут описывать несколько графовых выражений, например, следующее выражение эквивалентно предыдущему (так как подучено просто путём изменения порядка конструирования графа):

N* ([d›fun], b:l, [par›a, body›a]) (d:@, [arg›c]) (c) (a) Empty

Данные графовые выражения представляют собой последовательные применения конструирующих формул, что сближает их с функциональными языками. Поэтому и описание диаграмм в рамках этой модели названо функциональным. Диаграмма, заданная определением 2 (в

терминах предложенной здесь модели – “мультиграф”) всегда имеет представление в виде графовых выражений, что формально доказано [125]

На основании предложенного описания создаётся и описание синтаксиса и семантики языка. Конструирующие формулы используются для нахождения шаблонов внутри графа, которые могут рассматриваться как нетерминалы.

В случае рассмотренного языка VEX мы можем отобразить каждое графовое выражение в некоторое значение из домена D, подходящего для лямбда-исчисления [62]. Пусть d является переменной, представляющей значения из D. Определим семантику как некоторое движение по управляемому пути в абстрактном графе, то есть, семантика задаётся относительно контекстов некоторых узлов графа, и в терминах рекурсивных определений семантики, например, для узла v, семантическая функция S’ применяется к контексту узлов, имеющих выходящие рёбра, заканчивающиеся в v. Таким образом, S’ принимает два параметра – граф и узел, определяющий контекст. Преимуществом данного подхода является то, что мы можем разложить граф в текстовое представление на основе лямбда-выражений. Таким образом, для абстрактной семантики визуального языка VEX можно записать:

S’[v, N (v:d) g] = d

S’[v, N (v:@, [fun›f, arg›a]) g] = S’[f, g] (S’[a, g])

S’[v, N* (v:λ, [par›p, body›b]) (r, p) g] = Λd.S’[b, N (r, p:d, [])g]

Иными словами, семантическая функция, заданная на узле-“переменной”, не имеющей выходящих дуг, есть собственно значение этого узла. Семантическая функция, заданная на узле-“применении”, из которого выходят дуги применения функции и аргумента, есть суперпозиция семантических функций от узла, в который входят дуга функции, и узла, в который входит дуга аргумента.Семантическая функция, заданная на узле-

абстракции, из которого выходят дуги параметра и тела абстракции, есть функциональное значение (где Λ суть абстрактная семантическая функция), отображающее любое значение d в абстракцию, заданную некоторым телом, в том случае, если дуга-параметр задана как d.

Таким образом, рассмотренный формализм позволяет преобразовывать некоторые графы в текстовый вид, путём представления их в виде применения лямбда-выражений к другим графам.

Формализм на основе предикатов первого порядка предложен в [159]. Формализм предназначен для поддержки синтаксиса аттрибуированного графа, т.е. обыкновенного графа, узлам которого сопоставлены некоторые атрибуты. Описание языка состоит из двух частей – описание внешнего вида элементов языка и описание правил проверки корректности. Для последнего граф представляется в виде текста в соответствии со спецификацией GTDL (Graph Type Definition Language) [160] и с помощью предикатов на граф накладываются синтаксические ограничения. Для поддержки формализма разработано инструментальное средство MOSES[127]. Синтаксический разбор программ на MOSES осуществляется с помощью абстрактного автомата, т.е. семантика является операционной. Таким образом, вопрос о том, какого класса языки могут поддерживаться таким средством, остается открытым.

Подходы на основе искусственного интеллекта

Одной из первых работ, в которых была предпринята попытка создать парсер визуального языка с помощью искусственного интеллекта, была работа [270], в которой описывался компилятор, работающий на нечеткой логике.

Основой компилятора является обычная графовая грамматика. Однако, для решения проблемы недетерминированности разбора графа в грамматику вводятся “нечеткие функции членства”, которые для каждого

грамматического правила определяют его пригодность в контексте предметной области. Прежде всего, для каждого терминала функции возвращают значения, равные количеству его интерпретаций в контексте предметной области. Для сложных изображений, сконструированных с помощью грамматических правил, функции используют различные критерии оценки, такие как синтез на основе оценок элементарных изображений или оценка на основе оценок операций над элементарными изображениями. Также авторы предлагают более сложные алгоритмы, такие как оценка на основе расстояний между различными элементарными изображениями, а также “доверие” грамматическому правилу. С помощью таких оценок визуальный парсер выбирает деревья разбора не произвольным образом, а в соответствии со значениями оценок этих путей. Таким образом, при правильном подборе функций достигается оптимизация работы и, соответственно, существенное увеличение скорости разбора.

Язык MERA[249] является развитием этой идеи, обобщенной для задачи автоматического расположения элементов. Основным отличием его является то, что диаграммный язык относится к какому-либо стереотипу, для которого заранее заданы некоторые правила нечеткой логики, связанные с отображением элементов (например, все деревья рисуются примерно одинаково, поэтому правила их рисования одни и те же). Таким образом, с пользователя снимается значительная нагрузка по спецификации поведения диаграмм. Этот подход, по аналогии с шаблонным программированием, получил название “шаблонного упорядочивания”. На основе этого языка, в частности, была разработана система графического представления требований к программному обеспечению [250].

В работе [181] предпринята попытка использовать для нужд визуальных языков генетические алгоритмы. Эти алгоритмы применяются для построения графов, в которых минимизировано количество перекрытий и пересечений. В работе приводятся примеры достаточно сложных графов,

оптимизированных данным способом. Вместе с тем, данный подход, естественно, является чисто интерфейсным.

Язык GALAPAGOS [185] также призван решить проблему упорядочивания любых ориентированных графов за счет генетических алгоритмов, которые способны учитывать предпочтения пользователя. Язык использует критерии вида “начало графа должно быть выше окончания (для деревьев)”, “расстояние между узлами должно быть не меньше n”, “число пересечений должно быть минимальным”, “угол между связями должен быть больше m” и другие. Основным преимуществом подхода является его универсальность: будучи направленным на упорядочивание произвольного графа, он годится прочти для любых диаграмм.

Другие формальные подходы

Существует большое количество других попыток формализации визуальных языков, не попадающих в вышеприведенные классы. Ниже они будут рассмотрены подробнее.

В работе [149] приводится пример формального подхода, основанного на упорядочивании произвольных изображений, аналогичном языку TEX [168]. При этом подходе пространство разбивается на подпространства, внутри которых задается некоторый алгоритм пространственного упорядочивания. За счет развитой системы алгоритмов и взаимодействия между подпространствами, система позволяет получать достаточно сложные схемы. Данный подход, вообще говоря, не является формальным подходом к описанию языка, так как ориентирован исключительно на отображение диаграмм, а не на взаимодействие с ними. Преимуществом данного подхода является низкая вычислительная сложность алгоритма упорядочивания изображений: гарантируется, что любой граф может быть нарисован за O(n) шагов.

Работа [154] посвящена созданию формального языка описания генератора диаграмм. Язык, являющийся надстройкой над Lisp, позволяет создавать новые визуальные языки программирования на основе достаточно простых текстовых описаний.

Работа [186] представляет собой попытку создания среды описания диаграммеров на основе классификации элементов, некоторых допущений о правилах взаимодействия с языком и языка описания сценариев, а также обработки передачи сообщений. Так же как и в предыдущем случае, среда не основана на каком-либо формализме. В результате этого подхода была создана библиотека классов Escalante [187], реализующая обобщенные элементы диаграмм и позволяющая разрабатывать пользовательские диаграммеры.

Языки VODL и EDL [260,261] являются средством описания диаграммеров на текстовом языке, относительно которого авторы утверждают, что он способен описывать контекстно-свободные языки. В качестве базы для семантики используется алгебраический формализм описания языков ASF+SDF, который представляет собой набор выражений, заданных на переменных, описанных в секции описания синтаксиса языка. Для формализма существует программный прототип VASE [262].

Интересный формализм предложен Хольтом [156]. Он заключается в том, что объекты рассматриваются как символы исчисления, а отношения между ними – как отношения исчисления, в частности, соприкосновение можно рассматривать как вхождение, смежность – как применение, а разделение – как суперпозицию. Для данного исчисления был предложен ряд аксиом и операторов, а также рассмотрены основные отношения между объектами.

Формализм, предложенный в работе [76], предполагает представление диаграмм как визуальных предложений, составленных из изображений и их атрибутов. При этом пространственное расположение не рассматривается. Данный формализм имеет частное применение в медицинских задачах.

Дальнейшим его развитием явился формализм “CARW” (Visual Conditional Attributed ReWriting System) [77], в котором он приведен к виду, более похожему на графовые грамматики.

Еще один интересный формализм [173] предлагает описывать диаграммы на основе пи-исчисления [209], математического формализма для описания динамического поведения сетей. Тем самым достигается несколько преимуществ, например, возможность анализа изоморфизма для различных диаграмм. Вместе с тем, конечное визуальное отображение не рассматривается.

ПРИЛОЖЕНИЕ П3

XML СХЕМА ДЕСКРИПТОРА ВИЗУАЛЬНОГО ЯЗЫКА

```
<?xml version="1.0"?>
<xs:schema version="1.0"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

   <xs:element name="definition">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="language">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="menus" minOccurs="0" maxOccurs="1"
type="MenuItemType">
                     </xs:element>
                     <xs:element name="object" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:all>
                              <xs:element name="variables" type="VariablesType">
                              </xs:element>
                              <xs:element name="visuals" type="VisualsType">
                              </xs:element>
                              <xs:element name="clauses">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="clause" minOccurs="0"
maxOccurs="unbounded" type="xs:string">
```

```
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="constraints">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="constraint" minOccurs="0"
maxOccurs="unbounded" type="xs:string">
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="init">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="clause" minOccurs="0"
maxOccurs="unbounded" type="xs:string">
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name = "insert-at-x" type="xs:string"/>
                  <xs:attribute name = "insert-at-y" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:all>
            <xs:attribute name="name" type="xs:string"></xs:attribute>
            <xs:attribute name="insertion-type" >
              <xs:simpleType >
```

```
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="Standalone"/>
                            <xs:enumeration value="Relational"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
        <xs:element name="connection" minOccurs="0"
maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="roles">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="role" minOccurs="1"
maxOccurs="unbounded" >
                                    <xs:complexType>
                                        <xs:attribute name="token"
type="xs:string"></xs:attribute>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="variables" type="VariablesType">
                    </xs:element>
                    <xs:element name="visuals" type="VisualsType">
```

```
                    </xs:element>
                    <xs:element name="clauses">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="clause" minOccurs="0"
maxOccurs="unbounded" type="xs:string">
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="constraints">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="constraint" minOccurs="0"
maxOccurs="unbounded" type="xs:string">
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="init">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="clause" minOccurs="0"
maxOccurs="unbounded" type="xs:string">
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute name = "insert-at-x" type="xs:string"/>
                        <xs:attribute name = "insert-at-y" type="xs:string"/>
```

```
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="name" type="xs:string"></xs:attribute>
              </xs:complexType>
            </xs:element>
            <xs:element name="commands">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="command" minOccurs="0"
maxOccurs="unbounded" type="CommandElement">
                  </xs:element>
                </xs:sequence>

              </xs:complexType>
            </xs:element>
            <xs:element name="script">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="interface" minOccurs="0"
maxOccurs="unbounded" type="FunctionElement">
                  </xs:element>
                  <xs:element name="code" type="ScriptElement">
                  </xs:element>
                </xs:sequence>

              </xs:complexType>
            </xs:element>
```

```
            </xs:sequence>
            <xs:attribute name="name" type="xs:string"/>
            <xs:attribute name="fullname" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- User types-->

  <xs:complexType name="VariablesType">
    <xs:sequence>
      <xs:element name="variable" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name= "name" type="xs:string"/>
          <xs:attribute name= "type">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="string"/>
                <xs:enumeration value="integer"/>
                <xs:enumeration value="boolean"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
```

```
    <xs:attribute name = "name" type="xs:string"/>
  </xs:complexType>


  <xs:complexType name="VisualsType">
    <xs:sequence>
      <xs:element name="visual" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name = "role" maxOccurs="1" minOccurs="0">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name = "relationship" minOccurs="0"
maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="match" minOccurs="0"
maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:attribute name = "name" type="xs:string"/>
                            <xs:attribute name = "value" type="xs:string"/>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                      <xs:attribute name = "token" type="xs:string"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
```

```
                </xs:complexType>
              </xs:element>
              <xs:element name="popups" minOccurs="0" maxOccurs="1"
type="PopupsType">
              </xs:element>
            </xs:sequence>
            <xs:attribute name= "type" >
              <xs:simpleType >
                <xs:restriction base="xs:string">
                  <xs:enumeration value="ActivePoint"/>
                  <xs:enumeration value="Line"/>
                  <xs:enumeration value="ActiveArea"/>
                  <xs:enumeration value="AcceptPoint"/>
                  <xs:enumeration value="AcceptArea"/>
                  <xs:enumeration value="Text"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name= "name" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name = "name" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="PopupsType">
      <xs:sequence>
        <xs:element name="popup" minOccurs="1" maxOccurs="unbounded">
```

```
            <xs:complexType>
              <xs:sequence>
                <xs:element name="clause" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
              </xs:sequence>
              <xs:attribute name = "text" type="xs:string"/>
              <xs:attribute name = "condition" type="xs:string"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>

      <xs:complexType name="MenuItemType">
        <xs:sequence>
          <xs:element name="menuitem" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="clause" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
              </xs:sequence>
              <xs:attribute name = "text" type="xs:string"/>
              <xs:attribute name = "condition" type="xs:string"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
```

```
  <xs:complexType name="CommandElement">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="engine" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="FunctionElement">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


  <xs:complexType name="ScriptElement">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="engine" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

ПРИЛОЖЕНИЕ П4

XML СХЕМА ДЕСКРИПТОРА ДАННЫХ ДИАГРАММЫ

```
<?xml version="1.0"?>

<xs:schema version="1.0"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

   <xs:element name="asg">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="object" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:all>
                     <xs:element name="attributes" type="AttributeListType">
                     </xs:element>
                  </xs:all>
                  <xs:attribute name="name" type="xs:string"></xs:attribute>
                  <xs:attribute name="type" type="xs:string"></xs:attribute>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="name" type="xs:string"/>
         <xs:attribute name="fullname" type="xs:string"/>
      </xs:complexType>
   </xs:element>
```

```
  <xs:complexType name="AttributeListType">
    <xs:sequence>
      <xs:element name="attribute" minOccurs="0" maxOccurs="unbounded"
type="AttributeType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name = "AttributeType">
    <xs:all>
      <xs:element name="listvalue" type="ListType"/>
    </xs:all>

    <xs:attribute name= "name" type="xs:string"/>
    <xs:attribute name= "internalname" type="xs:string"/>
    <xs:attribute name= "value" type="xs:string"/>
    <xs:attribute name= "type">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="string"/>
          <xs:enumeration value="integer"/>
          <xs:enumeration value="boolean"/>
          <xs:enumeration value="list"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
```

```
  <xs:complexType name="ListType">
    <xs:sequence>
      <xs:element name="value" minOccurs="0" maxOccurs="unbounded"
type="AttributeType"/>
    </xs:sequence>
  </xs:complexType>

                                                              </xs:schema>
```

ПРИЛОЖЕНИЕ П5

ОПИСАНИЕ ER-ДИАГРАММЫ СРЕДСТВАМИ ВЫЧИСЛИТЕЛЬНОЙ МОДЕЛИ ВИЗУАЛЬНОГО ЯЗЫКА (ФРАГМЕНТ)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Simple ER-Diagram definition
-->

<definition     xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'

xsi:noNamespaceSchemaLocation='http://stepanoff.info/diagen/LanguageDefiniti
on.xsd'>
   <language name = "ERD" fullname="Entity-Relationship Diagram" >
      <menus>
         <menuitem text="-" condition="$true"/>
         <menuitem text="Toggle active points" condition="$true">
            <clause>$drawactivepoint=!$drawactivepoint;</clause>
         </menuitem>
      </menus>
      <object name = "Entity" insertion-type="Standalone">
         <variables>
            <variable name = "attributes" type="string"></variable>
            <variable name = "displayedattributes" type="string"></variable>
            <variable name = "tableschema" type="string"></variable>
            <variable name = "displayedtableschema" type="string"></variable>
            <variable name = "foreignkeys" type="string"></variable>
```

```
      <variable name = "rawforeignkeys" type="string"></variable>
      <variable name = "ansisqltypes" type="string"></variable>
      <variable name = "sqlfieldoptions" type="string"></variable>
      <variable name = "entitycaption" type="string"></variable>
      <variable name = "captionmargin" type="string"></variable>
    </variables>
    <visuals>
      <visual type = "Text" name = "caption"></visual>
      <visual type = "Text" name = "itemlist"></visual>
      <visual type = "ActivePoint" name = "topleftAP"></visual>
      <visual type = "ActivePoint" name = "toprightAP"></visual>
      <visual type = "ActivePoint" name = "bottomleftAP"></visual>
      <visual type = "ActivePoint" name = "bottomrightAP"></visual>
      <visual type = "ActiveArea" name = "area">
        <popups>
          <popup text="Delete object" condition="$true">
            <clause>$delete(this)</clause>
          </popup>
          <popup text="Object properties" condition="$true">
            <clause>$initDialog("Object
properties");$stringValueToDialog("Caption",caption.text);$stringTableToDialog("
Attributes",attributes,tableschema);$setBooleanColumn(0);$setListColumn(2,ansi
sqltypes);$setListColumn(3,sqlfieldoptions);$showDialog();</clause>
          </popup>
        </popups>
      </visual>
      <visual type = "Line" name = "topborder"></visual>
      <visual type = "Line" name = "bottomborder"></visual>
```

```
    <visual type = "Line" name = "leftborder"></visual>
    <visual type = "Line" name = "rightborder"></visual>
    <visual type = "Line" name = "captionborder"></visual>

    <visual type = "AcceptArea" name = "accarea">
      <role>
        <relationship token = "source">
          <match name="source" value="area"/>
          <match name="sourceattributes" value="attributes"/>
          <match name="destforeignkeys" value="rawforeignkeys"/>
          <match name="this1" value="this"/>
        </relationship>
        <relationship token = "dest">
          <match name="dest" value="area"/>
          <match name="destattributes" value="attributes"/>
          <match name="this2" value="this"/>
        </relationship>
      </role>
    </visual>
  </visuals>
  <clauses>
    <clause>entitycaption = caption.text</clause>
    <clause>caption.text=entitycaption</clause>
    <clause>topleftAP.x = bottomleftAP.x</clause>
    <clause>bottomleftAP.x = topleftAP.x</clause>
    <clause>topleftAP.x = area.x</clause>
    <clause>area.x = topleftAP.x</clause>
    <clause>toprightAP.x = bottomrightAP.x</clause>
```

<clause>bottomrightAP.x = toprightAP.x</clause>

<clause>cond(!$initialEvent(area.x))area.width = toprightAP.x - area.x</clause>

<clause>cond($initialEvent(area.x))toprightAP.x = area.x + area.width</clause>

<clause>topleftAP.y = toprightAP.y</clause>

<clause>toprightAP.y = topleftAP.y</clause>

<clause>topleftAP.y = area.y</clause>

<clause>area.y = topleftAP.y</clause>

<clause>bottomleftAP.y=bottomrightAP.y</clause>

<clause>bottomrightAP.y=bottomleftAP.y</clause>

<clause>cond(!$initialEvent(area.y))area.height = bottomleftAP.y - area.y</clause>

<clause>cond($initialEvent(area.y))bottomleftAP.y = area.y + area.height</clause>

<clause>topleftAP.y = topborder.y1</clause>

<clause>topborder.y1 = topleftAP.y</clause>

<clause>topleftAP.x = topborder.x1</clause>

<clause>topborder.x1 = topleftAP.x</clause>

<clause>captionborder.y1 = topleftAP.y+captionmargin</clause>

<clause>captionborder.y2 = toprightAP.y+captionmargin</clause>

<clause>captionborder.x1 = topleftAP.x</clause>

<clause>captionborder.x2 = toprightAP.x</clause>

<clause>toprightAP.y = topborder.y2</clause>

<clause>topborder.y2 = toprightAP.y</clause>

<clause>toprightAP.x = topborder.x2</clause>

<clause>topborder.x2 = toprightAP.x</clause>

<clause>bottomleftAP.y = bottomborder.y1</clause>

<clause>bottomborder.y1 = bottomleftAP.y</clause>

<clause>bottomleftAP.x = bottomborder.x1</clause>

<clause>bottomborder.x1 = bottomleftAP.x</clause>

<clause>bottomrightAP.y = bottomborder.y2</clause>

<clause>bottomborder.y2 = bottomrightAP.y</clause>

<clause>bottomrightAP.x = bottomborder.x2</clause>

<clause>bottomborder.x2 = bottomrightAP.x</clause>

<clause>topleftAP.y = leftborder.y1</clause>

<clause>leftborder.y1 = topleftAP.y</clause>

<clause>topleftAP.x = leftborder.x1</clause>

<clause>leftborder.x1 = topleftAP.x</clause>

<clause>bottomleftAP.y = leftborder.y2</clause>

<clause>leftborder.y2 =bottomleftAP.y</clause>

<clause>bottomleftAP.x = leftborder.x2</clause>

<clause>leftborder.x2 =bottomleftAP.x</clause>

<clause>toprightAP.y = rightborder.y1</clause>

<clause>rightborder.y1 = toprightAP.y</clause>

<clause>toprightAP.x = rightborder.x1</clause>

<clause>rightborder.x1 = toprightAP.x</clause>

<clause>bottomrightAP.y = rightborder.y2</clause>

<clause>rightborder.y2 = bottomrightAP.y</clause>

<clause>bottomrightAP.x = rightborder.x2</clause>

<clause>rightborder.x2 = bottomrightAP.x</clause>

<clause>caption.x=area.x+(area.width-caption.measuredWidth)/2</clause>

<clause>caption.y=topleftAP.y+captionmargin-caption.measuredHeight/2</clause>

<clause>itemlist.x=area.x</clause>

```
            <clause>itemlist.y=topleftAP.y+captionmargin</clause>
            <clause>itemlist.width=area.width</clause>
            <clause>itemlist.height=area.height-captionmargin</clause>
            <clause>itemlist.text=displayedattributes</clause>
            <clause>displayedattributes=atributesToWidget(attributes,
plainList(plainList(rawforeignkeys)))</clause>
            <clause>accarea.x=area.x</clause>
            <clause>accarea.y=area.y+captionmargin</clause>
            <clause>accarea.height=area.height-captionmargin</clause>
            <clause>accarea.width=area.width</clause>
        </clauses>

        <init insert-at-x="x" insert-at-y="y">
            <clause>area.x=x</clause>
            <clause>area.y=y</clause>
            <clause>area.width=180</clause>
            <clause>area.height=250</clause>
            <clause>caption.text="Entity"</clause>
            <clause>entitycaption = caption.text</clause>
            <clause>attributes=[[]]</clause>
            <clause>displayedtableschema=["name", "type",  "pk/fk"]</clause>
            <clause>tableschema=["key", "name", "type",  "can be null"]</clause>

<clause>ansisqltypes=["CHAR","VARCHAR","INTEGER","TINYTEXT","TEXT","BLOB"
,"MEDIUMTEXT","LONGBLOB","TINYINT","SMALLINT","MEDIUMINT","INT","BIGIN
T","FLOAT","DOUBLE","DECIMAL","DATE","DATETIME","TIMESTAMP","TIME","YE
AR"]</clause>
```

```
            <clause>sqlfieldoptions=["NULL","NOT
NULL","UNIQUE","CHECK"]</clause>
            <clause>captionmargin=30</clause>
            <clause>foreignkeys=[]</clause>
            <clause>rawforeignkeys=[]</clause>
         </init>

         <constraints>
            <constraint>topleftAP.x > toprightAP.x</constraint>
            <constraint>bottomleftAP.x > bottomrightAP.x</constraint>
            <constraint>topleftAP.y > bottomleftAP.y</constraint>
            <constraint>toprightAP.y > bottomrightAP.y</constraint>
         </constraints>
      </object>

      <connection name="Relationship">
         <roles>
            <role token="source"></role>
            <role token="dest"></role>
         </roles>
         <variables>
            <variable name = "keys" type="string"></variable>
            <variable name = "foreignkeys" type="string"></variable>
            <variable name = "keymatches" type="string"></variable>
            <variable name = "keysschema" type="string"></variable>
            <variable name = "tableschema" type="string"></variable>
            <variable name = "attributes" type="string"></variable>
            <variable name = "tableschema" type="string"></variable>
```

```
        <variable name = "innerrole" type="string"></variable>
        <variable name = "outerrole" type="string"></variable>

    </variables>
    <visuals>
        <visual type = "Line" name = "source1"></visual>
        <visual type = "Line" name = "source2"></visual>
        <visual type = "Line" name = "sourcetomiddlehoriz"></visual>
        <visual type = "Line" name = "sourcetomiddlevert"></visual>
        <visual type = "Line" name = "middletodesthoriz"></visual>
        <visual type = "Line" name = "middletodestvert"></visual>
        <visual type = "Line" name = "dest1"></visual>
        <visual type = "Line" name = "dest2"></visual>
        <visual type = "Line" name = "arrow1"></visual>
        <visual type = "Line" name = "arrow2"></visual>

        <visual type = "Text" name = "innerroletext"></visual>
        <visual type = "Text" name = "outerroletext"></visual>

        <visual type = "ActivePoint" name = "sourceAP">
            <popups>
                <popup text="Delete object" condition="$true">
                    <clause>destforeignkeys = mapRemove(destforeignkeys,
this);$delete(this)</clause>
                </popup>
                <popup text="Object properties" condition="$true">
                    <clause>$initDialog("Relation
properties");$stringValueToDialog("Inner
```

role",innerrole);$stringValueToDialog("Outer
role",outerrole);$stringTableToDialog("Foreign keys",keys,keysschema,
$false);$setColumnEditable(0,$false);$setColumnEditable(1,$false);$setColumnEd
itable(2,$false);$setListColumn(3,keymatches);$showDialog();</clause>
</popup>
</popups>
</visual>
<visual type = "ActivePoint" name = "middleAP">
<popups>
<popup text="Delete object" condition="$true">
<clause>destforeignkeys = mapRemove(destforeignkeys,
this);$delete(this)</clause>
</popup>
<popup text="Object properties" condition="$true">
<clause>$initDialog("Relation
properties");$stringValueToDialog("Inner
role",innerrole);$stringValueToDialog("Outer
role",outerrole);$stringTableToDialog("Foreign keys",keys,keysschema,
$false);$setColumnEditable(0,$false);$setColumnEditable(1,$false);$setColumnEd
itable(2,$false);$setListColumn(3,keymatches);$showDialog();</clause>
</popup>
</popups>
</visual>
<visual type = "ActivePoint" name = "destAP">
<popups>
<popup text="Delete object" condition="$true">
<clause>destforeignkeys = mapRemove(destforeignkeys,
this);$delete(this)</clause>

```
                </popup>
                <popup text="Object properties" condition="$true">
                  <clause>$initDialog("Relation
properties");$stringValueToDialog("Inner
role",innerrole);$stringValueToDialog("Outer
role",outerrole);$stringTableToDialog("Foreign keys",keys,keysschema,
$false);$setColumnEditable(0,$false);$setColumnEditable(1,$false);$setColumnEd
itable(2,$false);$setListColumn(3,keymatches);$showDialog();</clause>
                </popup>
              </popups>
            </visual>
          </visuals>
<!—фразмент вырезан-->
      </connection>
      <commands>
        <command name="-"/>
        <command name="Translate to SQL" engine="simplecompiler">
          <![CDATA[
            foreach (X:getObjects(type=="Entity"))
            {
              write("create table ");
              write(X.getAttribute("entitycaption"));
              write("(");
              writeln("");

              var attribute = X.getAttribute("attributes");
```

```
    foreach(Y:attribute){
        write(Y.getListElement(1));
        write("  ");
        write(Y.getListElement(2));
        write("  ");
        write(Y.getListElement(3));
        if (selector.hasNext()){
            writeln(",");
        };
    }

    writeln(")");
    writeln("go");
}


foreach (X:getObjects(type=="Entity"))
{

    var attribute = X.getAttribute("attributes");
    var hasPrimaryKey = false;

    foreach(Y:attribute){
        if (Y.getListElement(0))
            hasPrimaryKey = true;
    }

    if (hasPrimaryKey){
```

```
      write("alter table ");
      write(X.getAttribute("entitycaption"));
      write("(");
      writeln("");

      var comma = false;

      write("add primary key (");

      foreach(Y:attribute){
         if (Y.getListElement(0)){

               if (comma){
            write (",");
           } else comma = true;

          write(Y.getListElement(1));
                }
      }

      writeln(")");
      writeln("go");
   }
}

foreach (X:getObjects(type=="connection"))
{
   write("alter table ");
```

```
            var toname = X.getAttribute("to");
            var fromname = X.getAttribute("from");
            var fkfield = X.getAttribute("fkfield");
            writeln(toname);
            write("add foreign key (");
            foreach (Y:getObjects(name==toname))
            {
               var attribute = Y.getAttribute("content");
                     if (attribute)
                  foreach(Z:attribute){
                  var modifier = Z.getListElement(0);
                  if (modifier == "primary key"){
                                    write(Z.getListElement(2));
                  };
                  }
            }
            writeln(")");
            write ("references ");
            write(fromname);
            write ("(");
            write(fkfield);
            writeln (")");
            writeln ("go");
         }
      ]]>
   </command>
</commands>
<script>
```

```
<interface name="atributesToWidget"/>
<interface name="atributesToKeys"/>
<interface name="appendColumn"/>
<interface name="getColumn"/>
<interface name="plainList"/>

<interface name="mapGet"/>
<interface name="mapPut"/>
<interface name="mapRemove"/>
<code>
<![CDATA[
   var atributesToWidget = function(object, foreignKeys)
    {
      var ModelValue= Java.type('diagen.model.ModelValue');
      var PropertyType= Java.type('diagen.model.PropertyType');
      var modelValue = new ModelValue(PropertyType.LIST);
      modelValue.setEmptyListValue();
      if (!object.isVector())
         return modelValue;
      if (object.isEmptyList())
         return modelValue;
      if (object.getValueAt(0).isEmptyList())
         return modelValue;
      var iterator = object.iterator();
      while (iterator.hasNext())
      {
         var line = iterator.next();
         var lineValue = new ModelValue(PropertyType.LIST);
```

```
        lineValue.addValueToList(line.getValueAt(1));
        lineValue.addValueToList(line.getValueAt(2));
        if (line.getValueAt(0).getBooleanValue())
        {
           var pkValue = new ModelValue(PropertyType.STRING);
           if (listContains(foreignKeys, line.getValueAt(1)))
           pkValue.setStringValue("<pk> <fk>");
           else pkValue.setStringValue("<pk>");
           lineValue.addValueToList(pkValue);
        } else
        {
           var pkValue = new ModelValue(PropertyType.STRING);
           if (listContains(foreignKeys, line.getValueAt(1)))
           pkValue.setStringValue("<fk>");
           else pkValue.setStringValue("");
           lineValue.addValueToList(pkValue);
        };
        modelValue.addValueToList(lineValue);
     }
     return modelValue;
};

var atributesToKeys = function(object)
 {
     var ModelValue= Java.type('diagen.model.ModelValue');
     var PropertyType= Java.type('diagen.model.PropertyType');
     var modelValue = new ModelValue(PropertyType.LIST);
     modelValue.setEmptyListValue();
```

```
   if (!object.isVector())
      return modelValue;
   if (object.isEmptyList())
      return modelValue;
   if (object.getValueAt(0).isEmptyList())
      return modelValue;
   var iterator = object.iterator();
   while (iterator.hasNext())
   {
      var line = iterator.next();
      if (line.getValueAt(0).getBooleanValue()) {

         var lineValue = new ModelValue(PropertyType.LIST);
         lineValue.addValueToList(line.getValueAt(1));
         lineValue.addValueToList(line.getValueAt(2));
         lineValue.addValueToList(line.getValueAt(3));
         modelValue.addValueToList(lineValue);
      }
   }
   return modelValue;
};

var appendColumn = function(list)
{
   var ModelValue= Java.type('diagen.model.ModelValue');
   var PropertyType= Java.type('diagen.model.PropertyType');
   var modelValue = new ModelValue(PropertyType.LIST);
   modelValue.setEmptyListValue();
```

```
   var iterator = list.iterator();
   while (iterator.hasNext())
   {
      var newLine = new ModelValue(PropertyType.LIST);
      newLine.setEmptyListValue();
      var line = iterator.next();
      var lineIterator = line.iterator();
      while (lineIterator.hasNext())
         newLine.addValueToList(lineIterator.next());

      var matchValue = new ModelValue(PropertyType.STRING);
      matchValue.setStringValue("<select column>");
      newLine.addValueToList(matchValue);
      modelValue.addValueToList(newLine);

   }
   return modelValue;
};

var getColumn = function(list, column)
{
   var ModelValue= Java.type('diagen.model.ModelValue');
   var PropertyType= Java.type('diagen.model.PropertyType');
   var modelValue = new ModelValue(PropertyType.LIST);
   modelValue.setEmptyListValue();

   var iterator = list.iterator();
```

```
    while (iterator.hasNext())
    {
        var line = iterator.next();
        var lineIterator = line.iterator();
        var curColumn = 0;
        while (lineIterator.hasNext()){
            if(curColumn++ == column.getNumberValue().intValue())
                modelValue.addValueToList(lineIterator.next());
            else lineIterator.next();
            }
            print(modelValue);
    }
    return modelValue;
};


var mapGet = function(list,key)
{
    var ModelValue= Java.type('diagen.model.ModelValue');
    var PropertyType= Java.type('diagen.model.PropertyType');
    var modelValue = new ModelValue(PropertyType.LIST);

    var result = null;

    var iterator = list.iterator();
    while (iterator.hasNext())
    {
        var pair = iterator.next();
```

```
      if (pair.getValueAt(0).equals(key))
         return pair.getValueAt(1);

   }
   return null;
};

var mapPut = function(list,key, value)
{
   var ModelValue= Java.type('diagen.model.ModelValue');
   var PropertyType= Java.type('diagen.model.PropertyType');
   var modelValue = new ModelValue(PropertyType.LIST);
   modelValue.setEmptyListValue();

   var iterator = list.iterator();
   while (iterator.hasNext())
   {
      var pair = iterator.next();
      if (pair.getValueAt(0).equals(key)){
         continue;
      }
      else modelValue.addValueToList(pair);
   }

   var newPair = new ModelValue(PropertyType.LIST);
   newPair.setEmptyListValue();
   newPair.addValueToList(key);
   newPair.addValueToList(value);
```

```
    modelValue.addValueToList(newPair);
    return  modelValue;
};
var mapRemove = function(list,key)
{
    var ModelValue= Java.type('diagen.model.ModelValue');
    var PropertyType= Java.type('diagen.model.PropertyType');
    var modelValue = new ModelValue(PropertyType.LIST);
    modelValue.setEmptyListValue();

    var iterator = list.iterator();
    while (iterator.hasNext())
    {
        var pair = iterator.next();
        if (pair.getValueAt(0).equals(key)){
            continue;
        }
        else modelValue.addValueToList(pair);
    };
    return modelValue;
};
var listContains = function(list, value)
{
    var ModelValue= Java.type('diagen.model.ModelValue');
    var PropertyType= Java.type('diagen.model.PropertyType');
    var modelValue = new ModelValue(PropertyType.LIST);

    var result = null;
```

```
    var iterator = list.iterator();
    while (iterator.hasNext())
    {
        var element = iterator.next();
        if (element.equals(value))
            return true;
    }
    return false;
};
var appendList = function(list1, list2)
{
    var ModelValue= Java.type('diagen.model.ModelValue');
    var PropertyType= Java.type('diagen.model.PropertyType');
    var modelValue = new ModelValue(PropertyType.LIST);
    modelValue.setEmptyListValue();
    var result = null;
    var iterator1 = list1.iterator();
    while (iterator1.hasNext())
    {
        var element = iterator1.next();
        modelValue.addValueToList(element);
    }

    var iterator2 = list2.iterator();
    while (iterator2.hasNext())
    {
        var element = iterator2.next();
```

```
            modelValue.addValueToList(element);
        }
        return modelValue;
    };

    var plainList = function(list)
    {
        var ModelValue= Java.type('diagen.model.ModelValue');
        var PropertyType= Java.type('diagen.model.PropertyType');
        var modelValue = new ModelValue(PropertyType.LIST);
        modelValue.setEmptyListValue();
        var result = null;
        var iterator = list.iterator();
        while (iterator.hasNext())
        {
            var value = iterator.next();
            if (value.isVector())
                modelValue = appendList(modelValue, value);
                else modelValue.addValueToList(value);
        }
        return modelValue;
    };

    ]]>
  </code>
 </script>
</language>
</definition>
```

ПРИЛОЖЕНИЕ П6

ВНЕШНИЙ ВИД И РЕЗУЛЬТАТЫ ТРАНСЛЯЦИИ ER-ДИАГРАММЫ

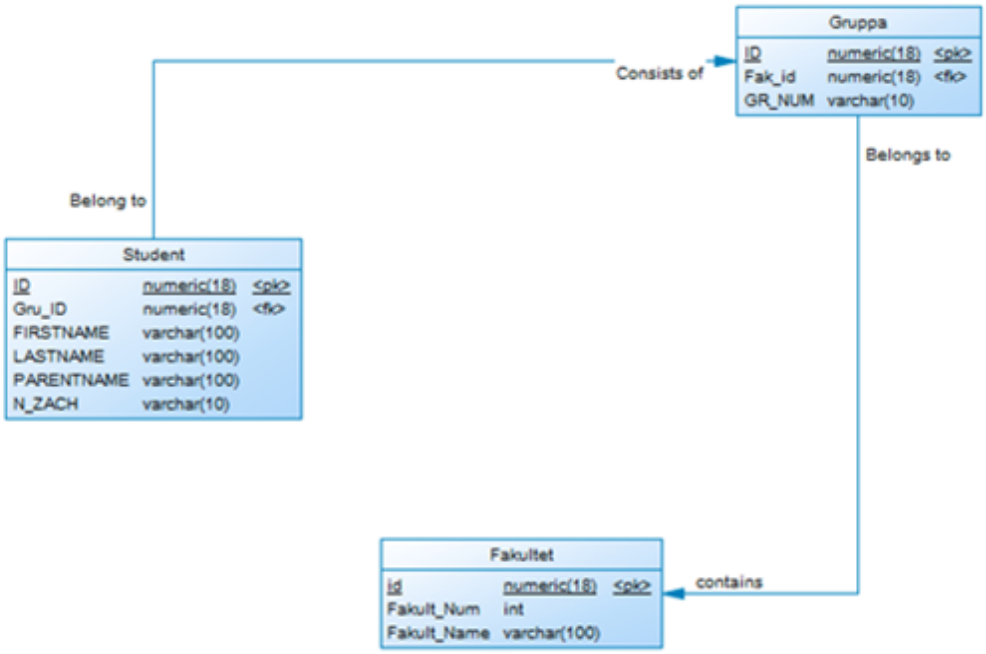


Рис. 13 Пример диаграммы в средстве моделирования Power Designer

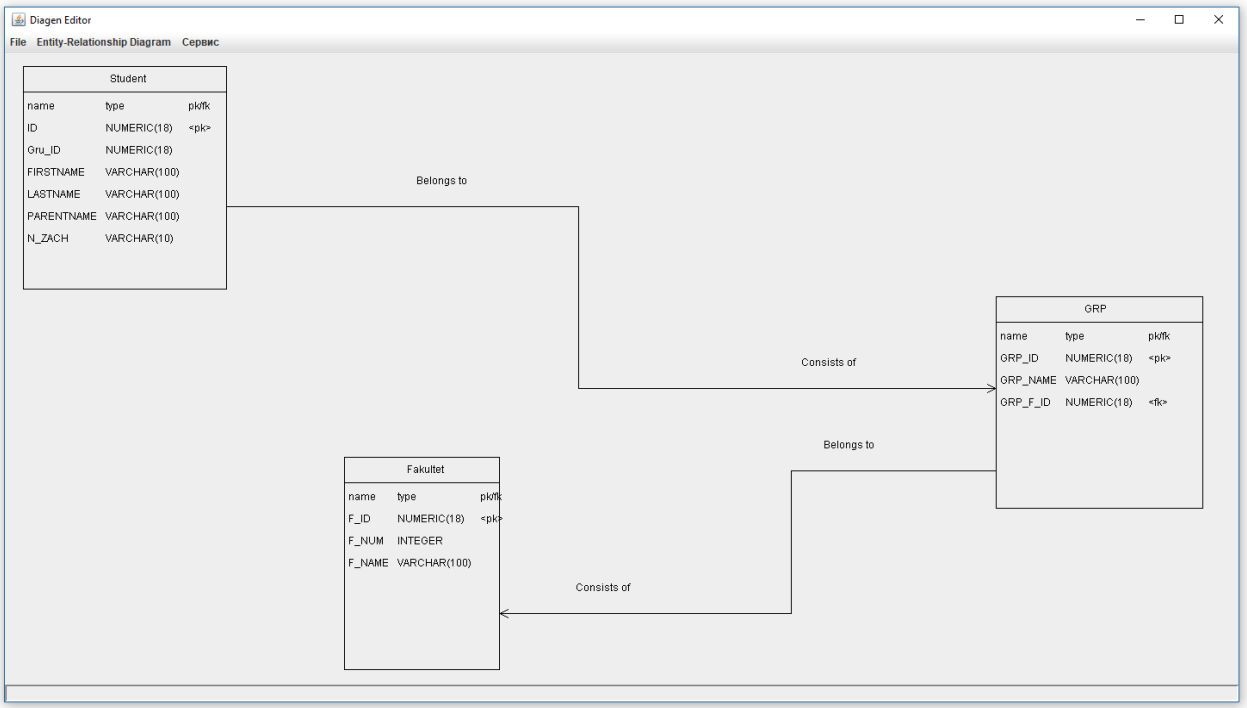


Рис. 14 Диаграмма с Рис. 13, реализованная средствами вычислительной модели визуального языка

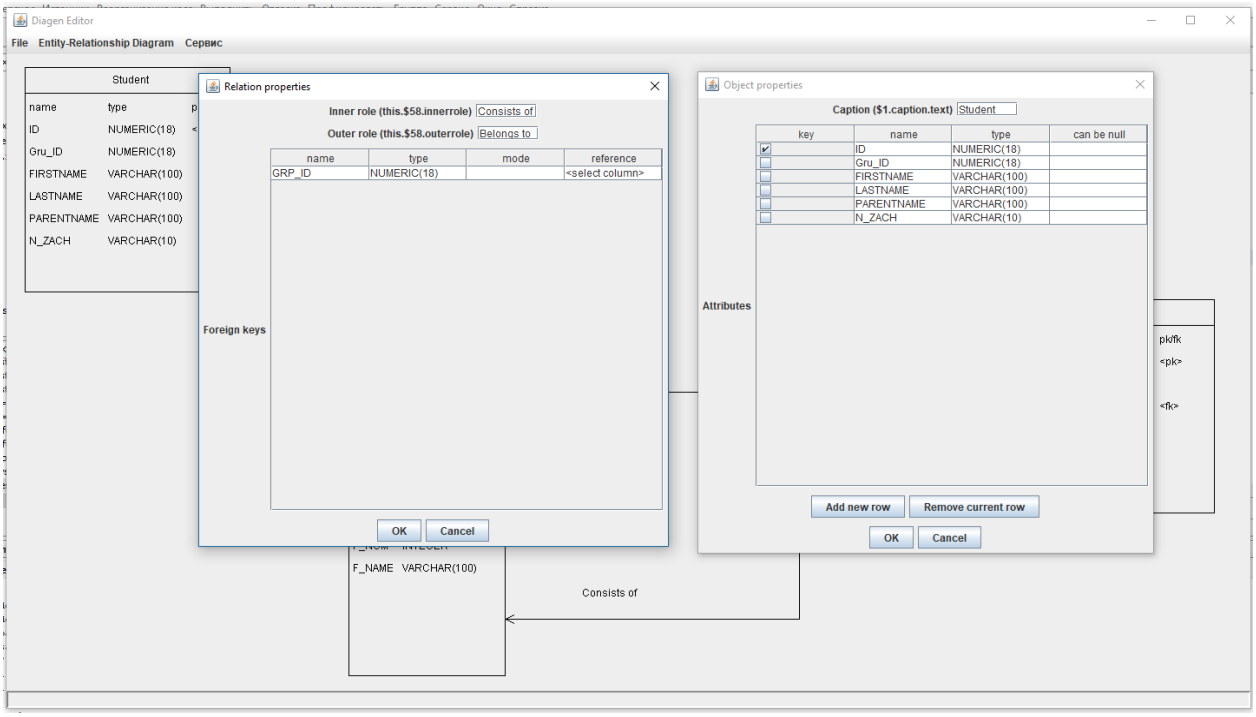


Рис. 15 Диалоговые окна для диаграммы с Рис. 13, реализованные средствами вычислительной модели визуального языка

```
create table FAKULTET (
F_ID  NUMERIC(18),
F_NUM  INTEGER,
F_NAME  VARCHAR(100))
go
create table STUDENT(
ID  NUMERIC(18),
Gru_ID  NUMERIC(18),
FIRSTNAME  VARCHAR(100),
LASTNAME  VARCHAR(100),
PARENTNAME  VARCHAR(100),
N_ZACH  VARCHAR(10))
go
create table GRP(
GRP_ID  NUMERIC(18),
GRP_NAME  VARCHAR(100),
GRP_F_ID  NUMERIC(18))
go
alter table FAKULTET(
add primary key (F_ID)
go
alter table STUDENT(
add primary key (ID)
```

```
go
alter table GRP(
add primary key (GRP_ID)
go
alter table STUDENT(
add foreign key (Gru_ID) references GRP(GRP_ID)
go
alter table GRP(
add foreign key (GRP_F_ID) references FAKULTET(F_ID)
go
```