Жаринов В.Н.

ОПИСАНИЕ ДЕЯТЕЛЬНОСТИ НА ОСНОВЕ МЕТОДОЛОГИИ ДРАКОН

Вводный цикл (извлечение)

Версия 10.1

ВВЕДЕНИЕ В ДОКУМЕНТ

Общие положения

1. Файл содержит выполняемый автоматизированным способом (в форме машинного оригинала МО) <беловик|черновик> целевого документа или его части (неотъемлемой), выделенной для удобства работы.

Документ в целом, кроме основного содержания, может включать приложения. Содержание документа, приложения (его выделенной части) составляют текст и/или иллюстрации (графчасть).

Конкретное наполнение файла определяется по его имени (полный формат имен см. шаблон документа)[1].

2. Содержание документа, приложения подразделено на структурные элементы по иерархии; её высшие 4 уровня стандартны. Элементы обычно имеют многоуровневую нумерацию и заголовки-абзацы, входящие в оглавление; возможны также элементы без нумерации, в т.ч. не входящие в оглавление, в т.ч. с заголовками в тексте.

В тексте применяются типовые приемы оформления, описанные в п/р 1.1 документа|шаблона.

3. В файл части из документа, приложения выделяется элемент структуры стандартного уровня иерархии (или ряд соседних элементов одного уровня) целиком (с заголовками).

Для многофайлового МО в имени каждого файла указаны индексы входящих элементов (формат: разделы <ЧN>, подразделы <пPNN>, пункты <ПNNN>, подпункты <пПNNNN>); файл первой части является *головным*.

При наличии приложений их форму (способ выполнения) указывают в отметках о наличии в составе единственного (или головного) файла основного документа (виды способов и формат отметок см. шаблон).

Приложения в МО могут выполняться как отдельные файлы *ПрилN* (что указывается в их отметках о наличии).

При наличии иллюстраций в документе, приложении (части) они также м.б. выполнены разными способами. Подрисуночные подписи включаются в оглавление для удобства поиска рисунков в документе.

Иллюстрации в МО могут содержаться в отдельном файле графчасти *Рисунки*; тогда текст содержится в файле *Текст*, и в нём дублируется подпись к каждой иллюстрации по месту её упоминания для отсылки к графчасти.

4. Оригинал документа (части) выполнен как настоящий файл (имя см. поле внизу) и другие необходимые (детальный состав многофайлового документа см. п. 1.1.4 в <настоящем файле|головном файле *Ч.1 Введ.*>).

Текст подготовлен в среде OpenOffice.org Writer или иной программы, совместимой по файлам; иллюстрации выполнены в той же программе и/или иными средствами, включая захват машобразов для МО.

Подлинник выполняется как твёрдая копия с заменой и/или добавлением листов к твёрдой копии предыдущих версий, либо как машинный образ файлов оригинала по листам, с которого делаются твёрдые дубликаты.

Информация, содержащаяся в документе, получена из открытых источников, рассматриваемых автором как надёжные. Возможное наличие секретных, конфиденциальных, а равно иных сведений ограниченного доступа следует рассматривать как результат предположения на массивах открытых сведений. Имея в виду возможные человеческие и технические ошибки, автор не может гарантировать абсолютную точность и полноту приводимых сведений, и не несет ответственности за возможные последствия, связанные с их использованием.

---

[1] Переменные части текста даются как поля в '< >', заменяемые на описание; общая часть (корень) поля пишется как есть, а изменяемые части как '*'. Файлы МО с однокоренным именем относятся к одному элементу структуры.

[2] Федеральный закон № 230-ФЗ от 18 декабря 2006 г.

[3] Если цитата состоит полностью из сведений, цитирующих другой источник – дать ссылку на первоисточник.

Назначение, сведения о версиях, языковые соглашения

1. Документ предназначен для представления содержания страницы гипертекста (либо ряда страниц, на которые делится его содержание) в виде обычного инфордока.

2. Версии документа выпускаются по мере обновления содержания цикла.

3. В тексте употребляются следующие типовые обозначения и сокращения:

букв.	буквально;
в т.ч.	в том числе;
и т.д.	и так далее;
и т.п.	и тому подобное;
к.-л.	какой-либо;
напр.	например;
см.	смотри;
т.е.	то есть;
т. зр.	точка зрения;
т.о.	таким образом;
разд.	раздел (документа);
п/р	подраздел (документа);
п.	пункт (документа);
п/п	подпункт (документа);
пред.	предыдущий;
след.	следующий;
ТЛ	титульный лист;

Сведения о терминологии, обозначениях и сокращениях данного документа см. в п/р 1.3.

Оглавление

4. ОСНОВЫ ТЕХНОЛОГИИ ВИЗУАЛИЗАЦИИ ДЕЯТЕЛЬНОСТИ

Как уже говорилось, технология – вещь неформальная. Она логически вытекает из свойств объекта и возможностей инструмента, согласуя первые со вторыми и формируется во многом эмпирически, на базе опыта применения имеющегося инструмента к нужным объектам; далее опыт обобщается и вырабатываются теоретические основы получения конкретных результатов (включая инвариант структуры техпроцессов), а затем и формирования конкретных техпроцессов по инварианту. ТФЗ-ДРАКОН находится как раз на эмпирическом этапе, многое ещё нуждается в проработке, поэтому часть наших результатов суть не готовые решения, а задачи на перспективу. Мы начнём со структуризации; затем попытаемся построить неформальную технологию.

4.1. Структуризация предметной области

Состав сущностей ТФЗ в общих чертах следующий. Как и любую иную технологию, ТФЗ-ДРАКОН можно представить в виде процедуры из ряда этапов и шагов, составляемых из определённого набора техопераций. Имеется также набор объектов (предметов и средств труда) и логических отношений между ними. Построением техпроцесса и его протеканием управляют некоторые факторы, среди которых есть ключевые.

Основным типом объектов нашей технологии являются импер-модели (описания деятельности) с элементами деклар-моделей (описания объектов деятельности), вводимыми для сохранения целостности.

Первым среди ключевых факторов следует назвать **язык представления знаний о задаче** – точнее, как мы помним, систему подъязыков. Исходным будет естественный язык, текстовый в своей основе; на нём, как правило, сочинитель даёт качественное описание задачи, нестрогое и целостное. Доля графики здесь зависит от степени образности мышления сочинителя. Далее необходим язык для более строгого целостного описания. От такого языка требуется выразить выделение сущностей-объектов (предметов и средств труда) и сущностей-процессов труда с представлением связей между ними.

> Можно применять визуальный язык типа IDEF0, как сейчас принято при инжиниринге процессов; в нём процессы труда представлены как функциональные блоки, предметы труда описываются как входные, выходные и управляющие объекты, а средства труда – как механизмы функций.
>
> По-видимому, нужно уточнить, какие языки оптимальны для обобщённого описания.

По сути, здесь начинается математическая стадия формализации наших знаний; при дальнейшей формализации процесс описывается на импер-языках, а его объекты – на деклар-языках, о чём мы также говорили.

Далее мы конкретизируем описание процесса, раскладывая каждое действие и его объекты на составляющие, пока не перейдём к точным терминам формального языка исполнителя (репертуару и багажу человека и машины); при этом детализируются и маршруты. В итоге имеем командную модель, однозначно понимаемую каждым исполнителем с этим репертуаром.

> В любом случае ТФЗ-ДРАКОН исходит из непреложных правил, своего рода «альфы и омеги» ДРАКОН-методологии, как они понимаются автором данного документа:
>
> 1. С начала применения (на стадии информатического моделирования или раньше) дракон-схемы рассматриваются как единственный источник знания об алгоритме задачи.
>
> 2. На всех этапах применения используется единый графический маршрутный язык, что сводит процесс формализации импер-знаний к детализации и уточнению маршрутов деятельности; при этом строгость маршрутного языка удовлетворяет требованиям однозначного перевода его конструкций в машинную форму.
>
> 3. По мере формализации переходят от менее строгого текстового языка описания действий и сущностей (более привычного специалисту предметной области) к более строгому (алгоритмическому); дальнейший переход к машинному языку (процесс кодирования программ) м.б. автоматизирован за счёт свойств маршрутного языка.

Исходя из этих правил, импер-модель представляется как дракон-схема либо дракон-модель с минимальными текстовыми пояснениями (и необходимыми элементами КогниСтиль).

Понятно, что как маршрутный импер-подъязык мы употребляем ДРАКОН; а как с командным? Он служит для спецификации действий и при качественной формализации представляет их в виде текстов на ограниченном (структурированном) естественном языке.

> Предложения по ограничению качественного языка мы оформили как РБНФ-выражения внутри икон алфавита техноязыка в Приложении 1.

Читатель не найдёт определённых ограничений для текста действий; этим мы хотели сказать, что на данной стадии их можно описывать так, как удобнее сочинителю. Желательно сразу придерживаться императивного стиля, т.е. описывать действия на основе глаголов.

> Примерами также могут служить тексты дракон-схем из /1, Гл.6...13/.

Далее, на математической стадии командный язык должен представлять действия как некие выражения. Содержание выражений зависит от характера процесса:

- информатический – перерабатывает одни данные в другие (Паронджанов называет его алгоритмом); так решаются т.н. информационные или вычислительные задачи;
- физический – перерабатывает одни матобъекты (предметы, материалы, энергии) в другие (Паронджанов называет его техпроцессом); так решаются производственные задачи.

Для первого рода процессов выражения содержат привычные математические операции (простейшие – арифметические и/или булево-логические – или построенные из них сложные).

Для процессов второго рода матопределение операций кажется не столь очевидным; конечно, можно свести, скажем, операции формообразования деталей из заготовок к геометрическим вычислениям аналогично тому, как это делается в машинной графике, но носители знаний (допустим, рабочие и технологи машиностроения) представляют себе эти операции, мягко говоря, несколько иначе:))

> Отталкиваясь «от жизни», мы предлагаем вводить для физической операции функцию, как показано в п/п 2.1.3.2 и текстах действий *шампур-блоков 1а* и *1б* (см. п. 2.2.1) ; определением этой функции в символическом представлении будет комплект технологической документации на операцию, а в предметном – пакет программ управления рабочими машинами, участвующими в ней.

Процесс описывается как следование функций. По сути, мы сохраняем язык предметной области, делая его частью командного подъязыка.

Собственно предметная форма описания образуется позже, на информатической стадии моделирования. Здесь матемвыражения принимают вид операторов присваивания, а все объекты – вид типизированных величин, т.е. конечных структур данных из багажа исполнителя.

Понятно, что для физических объектов величины есть их информодели; исполнитель должен по каналам ввода получать текущие значения этих величин, измеренные на реальных объектах.

Также появляются формально-логические условия выбора маршрутов в нелинейных структурах процессов; в этих условиях фигурируют те или иные алгоритмические величины.

Далее описание можно транслировать в коды команд и операндов конкретной информашины; для этого любая икона (её текст и смысл графики) д.б. выразима как та или иная последовательность машинных команд; возможно, что есть не один вариант такого выражения.

Для автоматической трансляции нужно указывать критерии выбора вариантов выражения.

Итак, **характер формализуемого процесса** есть один из ключевых факторов ТФЗ. Более широко в сфере научного изучения деятельности указывалось, что[4]: «организационно-техническая система реализует три резко отличных типа процессов:

- вещественно-энергетические преобразования или перемещения;
- обработка информации;

[4] Ломов Б.Ф. Инженерная психология в военном деле. – М.:,1977.

- информационные модели<рование> процессов первых двух видов на естественных и искусственных языках.»

Первый вид процессов мы вслед за Пароджановым определили как «техпроцессы».

Процессы второго вида суть информационные; следуя Фридланду, мы выделяем в них информатическую часть, определяемую как переработка исходных данных в результатные вне аппарата мышления человека (посредством инфортехники) для внешнего потребления.

Процессы третьего вида тоже информационные, но «для внутреннего потребления», в их результате оргсистема «познаёт саму себя» усилиями людей (в первую очередь её сотрудников); сюда относится и модный ныне [ре]инжиниринг бизнес-процессов (к которому могут привлекаются также сторонние лица-аналитики).

> Наша ТФЗ-ДРАКОН (как и любая ТФЗ вообще) как раз относится к третьему типу. Сам же ДРАКОН в силу универсальности маршрутного подъязыка считаем применимым для формализации процессов любых типов (возможно, с вариациями других подъязыков).
>
> Уточним, что термин «организационно-техническая» система (а кратко – оргсистема) сегодня имеет синонимы «социотехническая», «эргатическая» система. Распространённые термины «предприятие» и «учреждение» будем понимать как частные случаи реализации оргсистемы, а термин «человеко-машинная система» – как частный случай её масштаба – рабочее место (индивидуальное или коллективное), к которому сводится оргсистема либо которое выделяется в ней для дальнейшего рассмотрения.

Взаимосвязанным существенным фактором является **назначение процесса визуализации**. Для информационных процессов Фридланд в /4, п. 4.3.1/ выделяет три класса назначений: познание, управление, обучение. Мы создаём технологию, инвариантную к назначению процесса; однако в разных классах, по-видимому, будут свои особенности для отдельных этапов и шагов.

> Важно, что ДРАКОН может применяться в качестве и учебного, и практического (в терминологии Ершова – «производственного»), и познавательного языка (для последнего назначения мы фактически доопределили его алфавит обобщёнными блоками и вершинами).

По сути, говоря конструкторским языком, вначале мы составляем эскиз (аванпроект) будущего визуала (дракон-модели), где содержание деятельности изложено обобщённо и неформально, привычным носителю знаний языком. По мере детализации маршрутов и уточнения текста описание превращается из эскизов в исходные чертежи инструкций человеку (программ для информашины).

> Командный подъязык, т.о., будет иметь три диалекта для разных стадий формализации: текстовый, математический и информатический. Пример последнего для рабочих операторов показан в *шампур-блоках 1а* и *1б* (см. п. 2.2.1); математический отличается в первую очередь употреблением между частями выражений равенства вместо присваивания. Для условных операторов командный текст математически записывается как логические выражения; принципиальное отличие информатической записи м.б. в том, что эти выражения преобразуются к стандарту прогязыка, для которого пишется дракон-программа.

Таковы наши предложения по командному подъязыку визуалов; разумеется, они развиваются и уточняются применительно к конкретной предметной области сочинителями.

Итак, **степень обобщённости текста** – величина переменная и также служит фактором, управляющим технологией визуализации; меняется как объём текста (а значит, требования к габаритам содержащих его фигур), так и алфавит текста (различные спецсимволы, принятые в предметной области, должны поддерживаться приложениями редактирования дракон-схем).

Ещё один фактор – **степень содержательности визуалов дракон-модели** – может меняться в зависимости от стратегии подстановок.

Можно построить алгоритм, действия в котором полностью замещены иконами *Вставка*; такой визуал Пароджанов называет *алгоритмом-концепцией*. Противоположный случай, когда

вставки не употребляются, назовем *конкретным* визуалом. Очевидно, можно ограничить уровень конкретизации, допустив подстановки лишь вместо части шампуров визуала.

> Допускаем переменный уровень конкретизации визуалов дракон-модели.

Теперь обратимся к **деклар-языку**. Чтобы получать модель деятельности, практически полезную на всех этапах формализации, необходим адекватный ДРАКОНу по формальности и эргономичности набор деклар-языков; они д.б. реализованы в рамках ИСП-ДРАКОН (иначе говоря, дракон-среды). В настоящее время можно говорить о следующих языках такого рода:

- ФЛОКС – разработан в рамках ИСП ГРАФИТ-ФЛОКС под руководством создателя техноязыка;
- язык, предложенный Я. Романченко в рамках его проекта ИСП ДРАКОН-Оберон (ДРОН).

ФЛОКС имеет узкопредметную направленность; о других языках судить трудно, ибо они сегодня ещё формируются. В основном эта работа также ведётся силами участников упомянутого форума.

Кроме того, предполагается использование языков обобщённой визуальной формализации (схематизации) в качестве «эрзац-деклар-языков». Так, ГНОМ использован Г. Тышовым в рамках его проекта ИСП-ДРАКОН (авторское название "и.с. ДРАКОН"). Судя по отзывам пользователей-программистов, существенного эффекта это не дало.

Возможно, следует развивать предложения, содержащиеся в п. 2.1.1.4.

Ещё один фактор – **характер исполнителя**. В качестве крайних случаев м.б. только человек (когда описывается ручной процесс) или только машина (полностью автоматизированный); в последнем случае имеем информашину, управляющую рабочей машиной, т.е. косавт; в процессах переработки данных рабочие машины сводятся к периферийным устройствам. Чистый исполнитель-машина – это абстракция (даже если искусственный исполнитель реализуется как САУ, человек действует хотя бы в начале работы, а иногда и вмешивается по ходу процесса); поэтому обычно имеем человеко-машинную систему.

Сказанное означает, что машинный визуал всегда вызывается на исполнение человеком-оператором; также и отображение состояния машины, воспринятое человеком (в виде прямой индикации и/или косвенных признаков «чутья машины» опытным оператором), вызывает его действия, направленные на дальнейшее решение визуализируемой задачи.

> Для целостного описания необходимо визуализировать и действия человека при машинном решении задачи; они также будут являться инструкцией оператору машины.

Целостная визуализация, как и традиционное программирование, возможна двумя методами: «сверху вниз» и «снизу вверх». Последнее подразумевает, что сначала описывается то, что решается рабочей машиной (косавтом), а затем от этого строится сценарий действий человека. При таком подходе существенные моменты течения процесса в целом м.б. упущены. В первом случае машинное описание выделяется из исходного человекоориентированного.

> Мы отдаём предпочтение визуализации «сверху вниз», что будет видно из дальнейшего.

Особенности реализации языка – ещё один ключевой фактор формирования ТФЗ; кроме того, они оказывают обратное влияние на ЯПЗ, в нашем случае – на стандарт семейства техноязыков. Это вкратце обсуждалось в п/п 2.1.2.2.

Подразумевается, что решение этих и других вопросов приведёт к определённым возможностям как языка ДРАКОН, так и созданных дракон-сред, из чего определится и конкретная технология визуализации в этих средах на этом языке. Однако формированию любого техпроцесса присущи и общие закономерности, и универсальные определяющие факторы; вкратце остановимся на этом.

Ключевым фактором при формировании техпроцесса является **принцип его структурирования**. Не мудрствуя лукаво, положим в основу проверенный принцип проектирования материальных систем (ОКР технических изделий): от эскизного проекта, относительно независи-

мого от технологии изготовления изделия – к рабочему, содержащему технологию для конкретных средств производства. Учтём разницу между материальным производством и информатическим: модель деятельности «изготавливается» сочинителем визуалов с использованием инфорсимы поддержки импер-языка (в нашем случае – ИСП-ДРАКОН). Конечный продукт – командная модель для заданного класса человеко-машинных исполнителей (уровня квалификации людей, семейства машин), эксплуатируемая путём выполнения конкретным исполнителем данного класса каждый раз по конкретному маршруту.

Для нашего процесса визуализации сказанное означает, что сначала можно выбрать ИСП-ДРАКОН, затем эскизно формализовать деятельность на техноязыке (обычно из семейства ДРАКОН-1), далее выбрать исполнителя (систему реализации модели) и под него делать рабочую формализацию (детальный визуал либо дракон-модель) уже на техноязыке спецификаций, а далее – на гибридном техноязыке из семейства ДРАКОН-2 для информашин-исполнителей и/или для формального выполнения/анализа деятельности. Точно так же мы формализуем саму ТФЗ вначале нестрого, для реализации только человеком-сочинителем, а затем строго, как максимально поддающуюся автоматизации (реализации формальным исполнителем-информашиной).

Определённую значимость при формировании ТФЗ имеет **представление о жизненном цикле** создаваемой информодели. Современная системная концепция[5] подразумевает т.н. полный ЖЦ (*пожицикл*) из трёх фаз: создание/совершенствование; применение; утилизация. В то же время содержание и/или инструментарий ряда техопераций совпадают для первой и третьей фаз, в связи с чем можно выделить их инвариант.

Визуализация деятельности – техпроцесс переработки данных, сводимый к конечному числу операций из определённого набора. Технология визуализации предусматривает как строго формальные операции построения дракон-схем, так и полуформальные (не дающие однозначных результатов, допускающие выбор альтернатив выполнения по нестрогим критериям). Прежде всего это обусловлено нестрогостью исходного языка текстов визуала, позволяющей представлять неоднозначности решения задач в описываемой предметной области.

Формализуя исходное описание, сочинитель должен иметь возможность отразить неоднозначности в более строгой модели как набор однозначностей, между которыми формальный исполнитель выбирает по формальным же критериям; сочинитель может пробовать разные варианты таких наборов. Возможны личные предпочтения сочинителя среди эквивалентных и синтаксически правильных конструкций; их система определяет стиль визуализации.

Рассматривая базовые структуры, мы столкнулись с рядом типов техопераций: добавление к телу примитива (ветки силуэта) шампур-блока (простого или сложного); добавление/удаление шампуров (в силуэт или переключатель); добавление отдельных нешампур-икон к отдельным шампур-иконам сбоку; перенос точек слияния вертикалей в сложном шампур-блоке (операции с лианой). Их можно описать формально (хотя мы пока этого не делаем).

В основе ТФЗ-ДРАКОН лежат укрупнённые техоперации подстановки и детализации первоначального визуала с неформальным текстом. Реализовать их можно по-разному, но с системной т. зр. всё равно придем к базису операций типа: прибавления элементов к структуре (в т.ч. начальной, из множества данных как аксиомы); вычитания элементов из структуры (ограничено недопустимостью вычета элементов использованной аксиомы); преобразования структуры (изменения конфигурации связей, ограниченного топологическими запретами и/или текста икон, ограниченного синтаксисом немаршрутных подъязыков ДРАКОНа).

Т.о. **метод реализации операций** также относится к ключевым факторам протекания визуализации.

[5] См. напр.: Спицнадель В.Н. Основы системного анализа. – СПб.: Изд. дом «Бизнес-пресса», 2000. – Гл.3.

Приведённые в п/р 2.2 конструкции следования, ветвления и цикла являются элементами визуально-структурного представления алгоритмов. Шампур-блоки, к которым они сводятся (на схемах обведены штрихпунктиром), можно использовать как части любых алгоритмических структур, или проще говоря, для «крупноблочного строительства» визуалов. Подобный метод визуализации назовём ***нестрогим***; он требует контроля автора за соблюдением правил техноязыка (прежде всего топологических запретов) Здесь добавление/удаление реализовано как вставка/вырезание фигур, а операции с лианой – это простое графическое редактирование формы ломаных (т.н. полилиний); размеры фигур корректируются буксировкой за маркеры.

Метод оправдан при ручном построении дракон-схем (напр., в офисном пакете). Он применим и в рамках стандарта техноязыка, если при построении схем ограничиться копированием/вставкой сложных шампур-блоков, ранее составленных по правилам языка.

При ***строгом*** методе визуализации ограничиваются операциями, данными в стандарте техноязыка, т.е. строят дракон-схемы только на базе заготовок путём: ввода шампур-икон (макроикон) в точки ввода (в ветвлении – с автовыключкой вертикалей); добавления/удаления шампуров и боковых икон; стандартных операций с лианой (грамматически правильного переноса концов с устранением неоправданных изломов). Также допускается копирование/вставка шампур-блоков из ранее созданных дракон-схем; это не нарушает строгости, если они были созданы тем же методом (а значит, синтаксически правильны).

При любом методе выполняют также неформальные преобразования структуры дракон-схем (эквивалентные и/или равносильные).

> Автоматизированная ИСП-ДРАКОН (дракон-среда) должна поддерживать метод Когни-Стиль, стандартные команды редактирования схем (вырезание/вставка фрагментов), ограниченные правилами шампур-метода, неформальные преобразования структуры дракон-схем (эквивалентные и равносильные), формальные преобразования шампур-метода (операции с лианой).

Ещё один фактор, управляющий визуализацией – **формат диосцены**. Формат следует выбирать из стандартных рядов (обычно ряда A<N>), исходя из удобства восприятия. Удобно накладывать на произвольные форматы *конструктивную сетку* с ячейками заданного формата; по горизонтали (в ряд) укладывается определённое число ширин, а по вертикали (в столбец) – высот ячейки.

> Примем, что базовая ячейка соответствует машинописному листу в книжной ориентации (A4), т.е. ширина 210 х высота 297 мм (или в альбомной A4L, т.е. 297 х 210 мм).

Когда необходимо, будем обозначать формат структурной формулой вида:

число-рядов-ячеек x *формат-ячейки* x *число-столбцов*.

> Для ДРАКОНа стандартны чертёжные форматы до A4x4 и A1, а в необходимых случаях – A0 и A4xN; формат A4 не оптимален, т.к. не учитывает свойств графического языка. В общем случае неудобны и произвольные форматы, вытянутые по вертикали.

На экране устройств отображения лист показывается в масштабе (чаще уменьшенном); однако при регистрации в форме твёрдой копии обычно требуется масштаб 1:1.

> Принимаем, что форматы печатного листа и рабочего произвольны и не совпадают.

Организация дракон-схем, оптимальная с учётом регистрации, м.б. следующей. Базовый формат ячейки, накладываемой на рабочий лист, определяется возможностями устройства регистрации; как и весь печатный лист, ячейка имеет поля, в которых также недопустимо размещение икон; оставшаяся площадь есть рабочая область. Схема, занимающая более одной ячейки, вписывается в сетку так, чтобы иконы находились внутри рабочих областей ячеек; вертикали и горизонтали могут переходить между ячейками и тогда разрываются (физически) на границе.

Твёрдая копия выводится поячеечно, а затем склеивается (за смежные поля ячеек), при этом части каждой разорванной линии физически совмещаются (а логически они и так были едины).

> Для поддержки логического единства дракон-редактор должен воплощать принцип

конструктивной сетки, работая с границами и полями и контролируя попадание икон в рабочие области ячеек. Оптимально задавать поля склейки отдельно, а остальные (внешние для печатного листа в целом) определять по заданным полям листа.

Здесь же задаются **параметрами компоновки дракон-схем** на рабочем листе. Базовым размером положим ширину основного для большинства икон элемента-прямоугольника (параллелограмма), либо большую ширину многоугольного элемента (шестиугольника, трапеции); ширину иных элементов иконы будем выбирать в пропорции к базовому, как и их горизонтальное смещение, если таковое есть. Рационально выбирать базовый размер из некоторого ряда.

Зададимся рядом значений базовой ширины (в мм): 20 (только для литеральных и абстрактных операторов); 30; 40; 50; 60; 80; 100; ряд можно продолжать, но для большинства случаев этого достаточно.

Для конкретной диосцены выбирается одно значение из ряда; для иконы *Конец* (и, как правило, для икон, присоединяемых сбоку), будем выбирать значение, ближайшее меньшее к нему; для иконы *Заголовок*, напротив, обычно будем выбирать ближайшее большее.

Высоты элементов определяем исключительно по размерам текста (см. п. 2.1.1).

Примем отступы текста от контуров не менее 1 мм.

Введём также критерии рационализации компоновки для ручного построения.

1. Допустим изменения ширины икон от базовой, чтобы убирать строки-«хвосты».

Примем отклонения ширины икон не более ±¼ от базовой для данной дракон-модели.

2. Обычно вертикаль проходит через середину ширины иконы (для несимметричных икон – через геометрический центр их слепыша), но мы будем допускать смещения икон относительно вертикали вправо или влево.

Допустим смещения центра икон относительно вертикали не более ¼ от заданной ширины.

3. На одной диосцене можно поместить несколько шампуров (линейных визуалов); при этом между ними д.б. промежутки, позволяющие человеку уверенно воспринимать эти шампуры как отдельные объекты.

Примем промежутки между иконами:

- по вертикали – от ½ усреднённой высоты соседних графоэлементов, но не менее 5 мм;
- по горизонтали – от ¼ базовой ширины икон в соседних шампурах (для разных визуалов – усреднённой), но не менее 10мм.

Т.о. возможны отступления от эргономичной компоновки дракон-схем в целях более экономичного использования площади диосцены, но лишь в некоторых пределах.

4.2. Неформальное построение дракон-схем

Дадим общий очерк визуализации, рассматривая его и как образец структуры первоначального описания решения любой сложной задачи. Для начала мы опишем техпроцесс визуализации укрупнённо, а содержание техопераций раскроем отдельно, дабы не загромождать описания.

4.2.1. Укрупнённое описание техпроцесса

4.2.1.1. Общие положения

Определяющими факторами протекания визуализации являются (в скобках указаны принятые значения фактора): сложившееся у разработчика представление о задаче (внутренний образ плюс описание, отчуждённое на IDEF0-подобном языке); метод визуализации (строгий или нестрогий); форма дракон-схемы (примитив или силуэт); степень содержательности текущего визуала дракон-модели (от конкретного до алгоритма-концепции); степень обобщения содержания икон (от укрупнённой в виде абстрактных операторов до детальной в точных терминах исполнителя). Они отчасти взаимозависимы и в свою очередь определяются: формой языка описания (графический с текстовым заполнением фигур); стилем визуализации (предпочтения среди эквивалентных синтаксически верных конструкций – наиболее эргономичные); применяемыми средствами поддержки (считаем, что используется по крайней мере офисный пакет) и заданным форматом диосцены (листа – произвольный, экрана и печати – A4|A4L).

С учётом принятых общего состава и последовательности этапов ТФЗ, применительно к ДРАКОНУ содержание и взаимосвязь этапов можно определить следующим образом.

4.2.1.2. Первоначальная (эскизная) формализация

На данном этапе составляется исходный визуал с максимально обобщённым текстом икон (как правило, на естественном языке); одновременно формируется описание сущностей, с которыми оперирует данный визуал, и схема отношений между ними. Т.о., даётся визуальная постановка задачи. Этому сопутствует формирование углублённого представления о выполнении тех или иных операторов.

Каков критерий укрупнения операторов? Можно положить, что исходное описание д.б. линейно, т.е. все операторы в теле визуала определяются как шампур-иконы. Тогда мы не сталкиваемся немедленно с пересечениями и имеем обозримое описание деятельности.

Можно вначале описывать задачу в текстовом виде, допустим, так, как мы описываем здесь визуализацию: отдельно модель её сущностей и отношений, содержание этапов решения через укрупнённые операции и наконец, детализацию тех операций, содержание которых неочевидно для сочинителя и/или исполнителя. Т.о. происходит качественная формализация.

Примером эскизной визуализации описания сущностей и отношений предметной области может служить схема выполнения вставки из п/п 2.1.3.3; здесь предметной областью является дракон-модель и её представление и выполнение.

Визуал может быть результатом перевода алгоритма с другого языка. Кроме того, изначально задача м.б. поставлена словесно (скажем, как основная задача УАЗО в п/п 2.1.3.5).

Описания сущностей и икон вместе с дракон-схемой (дракон-эскиз) составляют основу для последующей формализации задачи. При этом ищется математический метод решения.

Для нахождения решения мы можем:

- провести формальные логико-математические рассуждения;
- подобрать готовый метод среди аналогичных задач, решённых ранее;
- построить решение интуитивно, исходя из здравого смысла и опыта сходной деятельности.

Как мы помним, возможно нестрогое описание, но оно всё равно математическое, поскольку определяются количественные параметры и пространственные формы, а также связывающие их отношения (функции, операции), включая отношения порядка действий при решении.

> Нестрого-математическое раскрытие неопределённости задачи возможно, в частности, средствами нечёткой логики Л.А. Заде. Для знакомства с этими и другими средствами данного рода рекомендуется работа /7, Гл.3/. Конкретизация для информатического моделирования возможна на базе современных методов, о которых подробнее см. п/п 4.2.1.5.

В итоге имеем дракон-спецификацию процесса решения задачи. Немаршрутные подъязыки спецификаций используют термины предметной области. Далее мы переходим к начальной информатической модели процесса путём алгоритмизации математического метода с одновременным определением входящих величин как алгоритмических (типизацией).

Возможны три рода методов алгоритмизации (в т.ч. визуальной):

- формально-логическое построение по математической записи решения;
- перевод с алго-/прогязыка (обычно текстового) уже имеющегося описания, скажем, исходного текста программы на Паскале, Си или «школьном» псевдокоде;
- интуитивное формирование алгоритма по мысленному представлению и/или словесно-изобразительному описанию процесса (неформальной и необязательно визуальной постановке задачи).

Методы первого рода (частных целей, подъёма, отхода назад, ветвей и границ и пр.) составляют основу школьного курса алгоритмизации и разобраны во множестве пособий.

Методы второго рода опираются на стандарт гибридного техноязыка, построенный для языка источника. В /1, Гл.12/ приведены примеры такого построения для распространённых сегодня ТЯП; основываясь на них, можно построить стандарт любого другого гибридного языка. Опыт в этом деле лучше всего приобретать на практике, переводя нужные алгоритмы.

Методы третьего рода основаны на мысленной «сборке» алгоритма из готовых частей, экстернальных (отчуждённых и воспринимаемых сочинителем с ранее опубликованных документов и/или из текущих сообщений в устной и иных формах) и/или интернальных (воображаемых сочинителем), по принципу «подходит-не подходит». Явно неподходящие (по мнению сочинителя) алгоритмы (конструкции) отбрасываются, для других возможна адаптация к данной задаче (интуитивная, исходя из знаний и опыта сочинителя). Методы чаще применяются для невычислительных задач.

Типизация относится к формализации деклар-знаний, и здесь мы её раскрывать не будем. Для начала достаточно придерживаться правил предложенного языка атрибуции данных (см. п. 2.2.2 Приложения 2); дальнейшие методы хорошо разработаны в теории баз данных, а их основы (включая начала инфологического моделирования) изучаются в школьной и вузовской информатике; более глубокие знания нужны в основном ИТ-специалистам.

По мере формирования деклар-стандартов для ДРАКОНа содержание типизации (и вообще деклар-моделирования), очевидно, можно будет уточнить.

Переход к информатической модели мы определили как представление всех действий через присваивание; однако сущность этого перехода не так проста. В самом деле, если матмодель задачи (или часть модели) представлена в виде квадратного уравнения $ax^2+bx+c=0$ с известными коэффициентами, информодель мы вначале должны записать как: x:=решить($ax^2+bx+c=0$; a:=число1, b:=число2, c:=число3) и далее «растехноложить» операцию решить, записав вместо неё последовательность действий по нахождению x (известную из школьного курса математики); т.е. в этом случае ищется (а чаще – подбирается готовый) математический метод решения. Другой случай – матмодель не требует решения, но записана через действия, не входящие в репертуар исполнителя информодели и/или входящие в неё объекты не входят в багаж исполнителя (т.е. не м.б. подведены ни под один тип объекта). Пример – вычисление суммы ряда величин под знаком "∑", когда исполнитель выполняет только сложение пары чисел. Здесь переход несколько проще; нужно представить операции матмодели через доступный репертуар, а её объекты – через имеющийся багаж (обычно – сконструировав производные типы из багажных). Наконец, возможно, что матмодель (её часть) представлена через операции, имеющиеся в репертуаре, но записанные иначе (отличающиеся синтаксисом). Тогда дело совсем просто – достаточно переписать операции в требуемом формате.

В связи с вышесказанным введём определение:

Математическое действие – выражение (равенство, отношение), допускающее запись получения его результата через присваивание без дополнительного решения (преобразования) средствами математики.

Понятно, что «действенность» математического выражения зависит от имеющихся языка и его реализации; так, если мы составляем программу на прогязыке, то сколько-нибудь сложная (специальная) математическая функция обычно не входит в число операций языка, и её нахождение требует ряда действий с применением доступных операций, которые надо описать подпрограммой; если же мы работаем в математическом пакете, то такая функция чаще всего входит в язык пакета и выражение, включающее эту функцию, можно сразу записать как присваивание (как правило, в репертуар такого пакета входит и операция решить в различных вариантах, пригодных для нахождения значений функций различных видов).

В любом случае главная цель этапа – сформировать «хорошую структуру» решения задачи. Критерий здесь – ясность и логичность. Основными структурирующими средствами техноязыка являются силуэтное представление визуалов (где метаструктуру можно задать как шапку) и дракон-моделирование (выделение визуалов-вставок) путём подстановки.

Очевидной является декомпозиция процесса на подпроцессы; в итоге образуется набор визуалов, представленных дракон-схемами разного уровня: главная, вставки в главную, вставки во вставки и т.д. Мы называем такой многоуровневый набор ***дракон-моделью***.

Кроме иерархической подстановки, возможны три важных частных случая организации дракон-модели:

- тот или иной визуал может вызывать сам себя (рекурсия);
- один визуал может вызывать другой, а другой – снова первый (взаимная рекурсия);
- визуал может вызываться двумя и более параллельными процессами.

Рекурсия имеет место, когда внутри визуала есть икона *Вставка* с именем того же визуала. Дойдя до неё, исполнитель создаёт процесс-копию с новым состоянием; дойдя в копии до той же иконы, создаёт еще одну копию и т.д. В корректной модели эта цепочка где-то должна прерваться и начаться обратное выполнение каждой из созданных копий до завершения; для этого рекурсивный визуал д.б. нелинейным, а вызов самого себя должен находиться на одном из его маршрутов (побочных), выбор которого происходит лишь до определённого состояния.

При взаимной рекурсии (парной) в каждом визуале пары есть икона *Вставка* с именем другого визуала («Иван кивает на Петра, а Пётр – на Ивана»); таких икон м.б. и больше.

При параллельном вызове икона *Вставка* с именем одного визуала имеется в визуалах, одновременно выполняемых по иконе *Параллельный процесс*; если модель строится по объектно-ориентированному принципу, то это происходит с визуалом метода, сопоставленного ряду одновременно вызываемых экземпляров класса (разных классов).

> Корректная алгоритмизация рекурсии предполагает следующее. В алгоритме выбирается некая совокупность величин (обязательно включающая хотя бы одну переменную), и определённый набор их значений рассматривается как условие завершения алгоритма. При каждом вызове отдельные переменные из этой совокупности закономерно изменяются; конечное число рекурсивных вызовов должно приводить к завершающему набору значений.

Сформированная структура наполняется содержанием – шампур-иконами конкретных действий (информатических и/или физических). Определяются условия ветвлений и циклов.

Для визуализации известных фрагментов процесса сочинитель по возможности использует типовые конструкции, подходящие по логике, изменяя лишь имена действий и/или сущностей алгоритма (объектов, величин).

Для исходного визуала может оставаться неизвестным точный смысл тех или иных икон, что в ряде случаев не даёт возможности остановиться на конкретном виде атома для ввода в критическую точку. Чтобы формально завершить построение дракон-схемы, не нарушая синтаксических правил, можно вводить в такую точку атом *Комментарий*, отражая в его тексте неточный смысл будущего оператора (шампур-блока). Тем же образом можно вводить в дракон-схему комментарии к существующим операторам, не забывая, однако, что при этом объём схемы увеличится.

Рассмотрим состав и условия применения неформальных преобразований.

Возможен логический переход к силуэту для схемы любого уровня, изображённой в виде примитива, когда сложившееся у разработчика «примитивное» представление о задаче сменяется на «силуэтное», и он выделяет элементы её метаструктуры, т.е. части, логически (по смыслу) «напрашивающиеся» в отдельные ветки. Условия его неформальны, в отличие от топологического, и определяются оценкой сложности визуала.

> Создатель техноязыка в /1, с.91/ указывает возможный критерий перехода – разветвлённая структура примитива с числом икон 15 и более.

Ветки эргономичного силуэта должны отражать метаструктуру описываемого процесса.

> Сформулируем следующий *принцип раскрытия*: в законченной дракон-модели алгоритмы должны описываться силуэтами-концепциями на всех уровнях, кроме наинизшего,

где допустимы также ограниченно конкретные и детальные силуэты, а также примитивы; при этом шапки всех силуэтов должны отражать логическое подразделение деятельности (каждая ветка – функциональный блок).

В завершение или время от времени проводится **формальная эргономизация визуала** путём ряда преобразований, сохраняющих эквивалентность итогового алгоритма исходному.

С целью эргономизации, во-первых, главный маршрут должен идти по шампуру (главной вертикали) примитива или по шампурам веток силуэта. Остальные маршруты считаются *побочными* и их вертикали упорядочиваются слева направо от главной по некоторому критерию. Для этого проводится ряд равносильных преобразований – рокировок.

Рокировка – перестановка местами выходов развилки или икон *Вариант*, включая и лианы. В результате содержание маршрутов (напр., главного и побочного) меняются.

При наличии веточных циклов главный маршрут силуэта становится разветвлённым, и задача упорядочения усложняется.

Во-вторых, в эргономичном визуале число вертикалей д. б. минимально возможным. При наличии одинаковых фрагментов в дракон-схеме часто возможно их объединение.

Вертикальное объединение – соединение друг с другом одинаковых икон (шампур-блоков), расположенных на соседних вертикалях, выходы которых объединены. Входы вертикалей соединяются линией, продолжающей крайнюю левую вертикаль, после чего иконы других вертикалей и другие линии удаляются, как показано в /1, с.113/.

Горизонтальное объединение – соединение друг с другом одинаковых икон (шампур-блоков), расположенных в концах соседних вертикалях, выходы которых объединены. Производится аналогично вертикальному, но вертикали не удаляются целиком, а сокращаются на одинаковый фрагмент, как показано в /1, с.114/.

Возможны и обратные операции (вертикальное или горизонтальное разъединение).

Рокировки и объединения не строго формальны и могут приводить к пересечениям. В этом случае выбирается иной вариант преобразования, если он возможен.

Данные эргономические преобразования могут противоречить не только требованиям шампур-метода, но и друг другу. Вводится следующее правило разрешения таких конфликтов: *правило главного маршрута более приоритетно, чем правило минимизации вертикалей*.

В общем случае проводятся ***цепочки*** преобразований – последовательности рокировок, объединений, разъединений; они могут разделяться подстановками (и обратными вставками) и детализациями (и обратными свёртками) с целью облегчения анализа схемы и выбора варианта преобразования. С этой же целью можно переходить от смысловой записи визуала к литеральной и обратно.

В-третьих, схема не должна быть чересчур сложна для восприятия. Сложность определяется как структурой визуала, так и параметрами оформления: форматом текста, икон, просветами. Часто задан один определённый формат диосцены.

Возможен эргопереход к силуэту при отсутствии топологической или логической необходимости; укажем основания для этого.

Для наилучшего восприятия диосцена должна быть вытянута по горизонтали (как киноэкран); однако чаще всего примитив, напротив, вытянут по вертикали. В этом случае следует перейти к силуэту при отсутствии пересечений; разделив шампур на ветки, можно изменить пропорции дракон-схемы, согласовав их с оптимальным соотношением сторон диосцены.

При заданном формате диосцены смысловые части силуэта, логически выделенные как ветки, могут неоптимально разместиться в высоту и ширину. Тогда ту или иную часть можно разнести по разным веткам.

При выделении веток безо всякой логики нарушается принцип раскрытия; лучше следовать метаструктуре либо сменить формат диосцены. Если то, ни другое невозможно, следует применять иные преобразования.

Возможны эргоподстановки для упрощения восприятия дракон-схемы в заданном формате. Тем самым можно: повысить уровень конкретизации; заместить идентичные фрагменты в разных частях дракон-схемы, которые нельзя объединить; "сократить" высоту слишком большого шампура и т.д.

> Визуальные алгоритмы-вставки изображаются на отдельных рабочих листах; однако при наличии свободного места рядом с вышестоящей дракон-схемой можно совместить её со вставками на одном листе (для наглядности вставки группируются слева).

Техоперация эрговыделения веток решает задачу сокращения высоты шампура ветки иначе – разделяя её на две и более последовательных веток. Каждая часть исходной ветки д.б. шампур-блоком; кроме того, желательно делить ветку на логически законченные части.

Исходя из приоритета публикации данного положения, будем называть критерий применения эрговыделения веток "правилом Я.Романченко" (см. форум /8/).

В итоге получаем дракон-спецификацию, используемую далее.

4.2.1.3. Выбор методов и средств решения

Исходит из того, что как результат ТФЗ должна получиться командная модель решения задачи человеком (возможно – совместно оператором и программно-аппаратной информашиной).

Детализация как процесс формализации на ДРАКОНе, с одной стороны, касается уточнения маршрутных знаний на маршрутном подъязыке. С другой стороны, уточняется командное знание с повышением формальности командного подъязыка; то же касается декларативной составляющей. В пределе как стандарт текстовой части можно использовать синтаксис того или иного существующего языка программирования, что приводит к гибридному языку программирования ДРАКОН-Х.

Как средство визуализации выбирается дракон-система (среда) либо менее автоматизированная ИСП; во втором случае можно выбирать также между строгим и нестрогим методом визуализации (в дракон-системе возможен только строгий).

Здесь же выбирается метод дальнейшей формализации: опытовый («проб и ошибок»), логический или прототипирования.

Рекомендуется и определиться, будет ли дальнейшая работа вестись полностью силами носителя знаний либо с участием специалистов-аналитиков. К последним относятся инженеры по знаниям, менеджеры качества и т.п.

При автоформализации сочинителем является носитель знаний о задаче; чтобы выделять этот случай, назовём его автосочинителем. Однако он и в этом случае необязательно будет выполнять сочинённое описание в силу разных причин: сочинитель сам уже не решает задачу, а лишь отдаёт свой прошлый опыт как эксперт; он с самого начала не решал её, а изучал решение в роли аналитика. Сочинение «для себя» (только или не только) – также особый случай формализации, т.н. утилитарной; такого сочинителя можно именовать «бард».

Как нетрудно понять, термин заимствован из художественной культуры, где он обозначает автора-исполнителя литературных произведений; обозначить принадлежность к культуре информатической можно, построив имя по аналогии с другими информатическими терминами, скажем, «инфорбард», а более узко, к алгоритмической (импер-инфоркультуре) – «алгобард», «процесс-бард» и т.п. – варианты м.б. разными :).

В любом случае из всех исполнителей сочинителями м.б. только некоторые.

Выбрав конкретную информашину-исполнителя, имеем данные о предельной точности вычислений (определяется разрядностью арифметики процессора). Определив отсюда погрешность вычислений, можно провести проверку математического решения на корректность по правилам Математики-2, введённой Ю.П. Петровым. Если решение оказывается некоррект-

ным, выполняют дополнительные преобразования Математики-2, восстанавливающие корректность (если возможно).

Здесь имеются в виду следующие результаты, полученные отечественным ученым Ю.П. Петровым и его сотрудниками при исследовании задач и систем автоматического управления[6].

Прежде всего, до 1998 г. считалось, что ***корректные*** и ***некорректные*** по качеству математического решения задачи образуют два базовых класса, четко отделённых друг от друга. Однако, на протяжении ряда лет исследуя методы формального синтеза систем управления и причины неустойчивости синтезируемых систем, Петров обнаружил третий, ***пограничный*** класс задач, способных изменять корректность в ходе преобразования при их решении. В частности, существуют такие системы уравнений, решения которых не обладают свойством непрерывной зависимости от параметров. Корректность может изменяться, даже если все преобразование состоит в перестановке местами уравнений в системе относительно исходного их порядка.

Исходя из сказанного, вводится понятие об эквивалентности математических преобразований ***в расширенном смысле*** – т.е. как неизменения корректности решаемой задачи в ходе этих преобразований. *Для конкретных классов задач далеко не все преобразования, эквивалентные в традиционном (узком) смысле, эквивалентны и в расширенном; в связи с этим возможно, что формально правильное решение задачи будет фактически неверным.*

Показано, что популярные математические пакеты программ имеют методические ошибки, основанные на неправильном понимании эквивалентности. *В общем случае нельзя алгоритмизовать преобразования систем уравнений*, т.к. нужно учесть связи между вариациями параметров разных объектов, описываемых уравнениями (и, напротив, отсутствие таких связей); это может сделать только исследователь, понимающий физический смысл процессов, происходящих в объектах и системе в целом. В то же время такие пакеты, как MatLAB, автоматически совершают указанные преобразования в предположении (конечно, не программы, а ее разработчиков), что они всегда будут эквивалентны.

Свойства использованных преобразований следует рассматривать в комплексе: математическая модель – задача (в физическом смысле) – выбранный метод ее решения. Решение некорректных задач следует проверять контрольными примерами при вариациях исходных данных. На практике это не всегда возможно; поэтому для определённых задач применяют специальные методы проверки решений и приемы преобразований, сохраняющие эквивалентность в расширенном смысле.

В дальнейших своих работах Петров разработал общие приёмы установления корректности и частные их приложения к математической формализации задач различных классов. Всё это он назвал собирательным термином «Математика-2».

Отметим, что эти идеи были предвосхищены видным отечественным учёным-палеонтологом, мыслителем и литератором И.А. Ефремовым. В своём романе «Туманность Андромеды» (1957) он ввёл т.н. биполярную математику, которая должна отражать диалектический подход в количественно-формальном знании. В качестве составляющей этой математики Ефремов ввёл и определил т.н. репагулярное исчисление как формальное исследование переходов количественных характеристик предмета изучения в его новое качество. Из определения видно, что Математика-2 фактически может рассматриваться как разновидность такого исчисления применительно к переходам количества погрешностей решения задачи в качество корректности метода решения.

С точки зрения ключевых идей нашего курса, подобный результат можно рассматривать как подтверждение одного из главных положений «китайской» диалектики: «сущность является в форме». Недопустимое изменение формы приводит к тому, что математическая запись перестает отражать сущность описываемых явлений. Кроме того, мы сталкиваемся с системной триадой «модель-задача-метод», где каждый компонент существен для результата.

Следует отметить, что все вышесказанное – дело в первую очередь ученых-математиков, решающих задачи на своем уровне формализации; для практиков (включая специалистов-

[6] Изложение см.: Петров Ю.П., Петров Л.Ю. Неожиданное в математике и его связь с авариями и катастрофами последних лет. Изд.1/2/3. – СПб.: СПбГУ, 1999/2000/2002.

информатиков) главное – знать о наличии «подводных камней» на пути решения и требовать надлежащего качества математических результатов[7].

4.2.1.4. Детальная формализация

Здесь проводится уточнение дракон-спецификации (визуальной импер-информодели, где язык спецификаций лежит в основе немаршрутных подъязыков) до командной с переходом к формальному командному подъязыку и построением строгой инфологической модели отношений между алгоритмическими величинами; будем называть результирующее описание ***дракон-руководством***. Для простых задач такая модель часто очевидна и требует начальных средств типа схем ресурсов; в сложных случаях применяют деклар-МФЗ, подобные IDEF1X.

При автоматизации командную модель выполняет машина, но под управлением человека. В общем случае при наличии информашины-исполнителя имеем систему «человек-косавт»; для неё дракон-спецификация разделяется на операторскую и машинную части и в таком виде формализуется детально. Уточним, что вторую часть можно представлять как отдельные алгоритмы (вставки) в составе первой; однако возникает вопрос о представлении человеко-машинного взаимодействия.

> Чтобы формализовать непосредственно работу оператора за машиной, можно разделить вставку на визуал-инструкцию и параллельный визуал-программу; здесь взаимодействие происходит посредством пультовых команд человека и сообщений машины, изображаемых иконой И20 в нужных места инструкции (программы).

Итак, результатом данного этапа формализации является дракон-программа (для информашины) и/или дракон-инструкция (для человека); очевидно, они должны логически дополнять друг друга до целостной импер-информодели (дракон-руководства) и оперировать общим множеством объектов и отношений.

Суть данного этапа – превратить хорошо структурированное решение задачи в эффективные и гарантоспособные операторский и машинный компоненты. Гарантоспособный для человека – значит эргономичный, высвобождающий его усилия для творческого решения задачи, стимулирующий к совершенствованию процесса; в силу целостности сознания гарантоспособность человека взаимосвязана с эффективностью. Гарантоспособный для машины – выполнимый при конкретной её конфигурации: хорошо, если окошко «Программа выполнила недопустимую операцию и будет закрыта» появляется при редактировании маловажного текста, а если при управлении ракетой или реактором?.. О машинной (вычислительной) эффективности алгоритмов и программ сказано столько, что повторяться не будем; отметим лишь, что ручная оптимизация на когнитивно качественном языке, конечно же, упрощается, а машинная – дело транслятора, и значит, следующего этапа ТФЗ.

Точно также, как при эскизной формализации, используются методы алгоритмизации того или иного рода, но уже для отдельных операторов (шампур-икон и блоков) дракон-спецификации. В связи с формализацией немаршрутных подъязыков детальная формализация носит характер визуального программирования на техноязыках семейства ДРАКОН-2.

Практически при детальной алгоритмизации происходит **раскрытие содержания** составленного визуала (дракон-модели). При этом существенные сведения о выполнении операторов переносятся в дракон-схему из внешнего текста и/или детализируются укрупнённые абстрактные операторы. Здесь можно указать два возможных способа:

- икона (её внешнее текстовое описание) оформляется в шампур-блок (структурный, возможно, в дальнейшем – лианный), который замещает собой эту икону; этот способ пригоден,

[7] Обзор проблем для широкого круга читателей см.: Петров Ю.П. Новые главы теории управления и компьютерных вычислений. – СПб.: БХВ-Петербург, 2004. Конкретные результаты для ряда классов задач см.: Петров Ю.П. Обеспечение надежности и достоверности компьютерных вычислений. – СПб.: БХВ-Петербург, 2008.

когда первоначальная структура дракон-схемы удовлетворительно отражает задачу независимо от уровня детализации описания;

- на основе группы икон и их описаний составляется адресный или шампур-блок (структурный, лианный), замещающий собой сразу группу икон и в общем случае не эквивалентный этой группе по своей структуре; так следует поступать, если структура дракон-схемы или её части по мере формализации оказывается неоптимальной.

Определённую таким образом операцию раскрытия содержания назовем детализацией. Ей обратна свёртка – операция перехода к укрупнённому описанию фрагмента; однако можно свернуть ранее детализированный фрагмент и в несколько отдельных частей, и/или «захватить» смежные части, до этого (в предыдущем укрупнённом описании) не входившие именно в этот фрагмент.

Свёртку целесообразно проводить с сохранением раскрытого содержания; для этого можно поступить следующим образом: создаётся визуал, именуемый, скажем, <*СвёрткаN*>, где N – уникальный в рамках задачи индекс; в качестве тела визуала подставляется детальное содержание; на его место в тело укрупняемого визуала вводится икона Вставка с именем <*СвёрткаN*>. Другая возможность — использование икон Область, описанных в п/п 2.1.3.12.

Очевидно, что при исходно высокой концептуализации визуала (обусловленной, напр., заданным форматом диосцены) должны детализироваться алгоритмы-вставки; при этом в дракон-модели могут появляться новые уровни иерархии вызовов.

При формировании детального решения также проводятся подстановки. Возможно, они отменяют результаты ранее проведённых подстановок. Переподстановки приводят к изменению архитектуры дракон-модели, в т.ч. к её переходу в дракон-схему. Цель техоперации – прежде всего выделить повторно используемые фрагменты алгоритма (повторяемые в разных местах, возможно, с иными фактическими параметрами при инвариантном наборе формальных параметров).

Детализация и подстановка взаимно сочетаемы при условии, что алгоритм-вставка сохраняет эквивалентность.

Как и эскизную модель, готовую (полностью или частично) дракон-программу можно (а часто и нужно) структурно преобразовывать. **Смысловые преобразования** (меняющие работу алгоритма) связаны с пересадками и/или заземлениями лиан. Кроме того, совершаются и рассмотренные выше **эргономические преобразования**, сохраняющие смысл алгоритма, но улучшающие его структуру.

При детализации возможно нарушение запрета на пересечения и/или обрывы линий; в этом случае выполняется топологический переход к силуэту, когда шампур разделяется на ряд веток с целью вывести маршруты, образующие неустранимые пересечения, в петлю силуэта; тем самым исключается их обрыв внутри листа.

Процесс уточнения модели (раскрытия содержания) итеративен в силу следующего:

- Детализация любого из визуалов дракон-модели может влиять на представление о других визуалах, в. т.ч. иных уровней; возможна коррекция дракон-схем, вплоть до главной.
- Переход к иному языку способствует углублённому пониманию тех или иных элементов.

В силу этого происходит возврат к предыдущим шагам, в т.ч. на предшествующие этапы.

Эргономические преобразования также могут повлиять на представление автора схемы о построенных визуалах; для внесения изменений возвращаются к предыдущим этапам. Тем самым происходит итеративный процесс уточнения модели деятельности и её эргономизации.

Гибридный визуал в окончательной редакции одновременно служит основным документом на процесс (по Паронджанову – т.н. *исходным чертежом*).

4.2.1.5. О дальнейшем содержании ТФЗ

На основании исходного чертежа программы выполняются заключительные этапы ТФЗ: кодирование машинных алгоритмов, тестирование и оптимизация, ввод в эксплуатацию. Их содержание зависит от третьей компоненты методологии – реализации техноязыка как части информатической системы. Для наших целей их рассмотрение требуется лишь в той мере, в какой они влияют на существо ДРАКОН-МФЗ. Согласно общей концепции ЖЦ сложных систем, все они относятся к фазе создания системы; далее следуют вторая и третья фазы пожицикла.

Проверка и оптимизация. В настоящее время наметился решительный переход к промышленному производству инфопрогизделий. Он требует отношения к формализации знаний (и программированию как конечной её стадии) как к инженерному делу, требующему научно обоснованного, расчётно подтверждаемого для конкретных условий эксплуатации проектирования не только оборудования, но и программ. Теоретическое обеспечение такого подхода получило название доказательного программирования; это направление начало развиваться ещё в 1970-х годах, прежде всего Дейкстрой, Ершовым, Хоаром и др. Однако оно требовало и адекватного математического обеспечения программирования (а шире — формализации деятельности), которое стало появляться сравнительно недавно[8].

Воплощением МО являются модели деятельности как объекта проектирования, допускающие анализ корректности проектных решений, выполнения неких формальных свойств, отражающих существенные требования к решению задачи. Тем самым система реализации деятельности рассматривается как продукт, к которому применимы базовые процессы управления качеством:

- *верификация* — подтверждение на основе представления объективных свидетельств того, что установленные требования были выполнены;
- *валидация* — подтверждение на основе представления объективных свидетельств того, что требования, предназначенные, для конкретного использования или применения, были выполнены.

Определения даны согласно стандарту ГОСТ Р ИСО 9000-2000. Говоря проще, если при верификации проверяется, удовлетворяет ли продукт требованиям, как они сформулированы для разработчика, то при валидации — разработан ли тот продукт, что требуется заказчику (удовлетворяет ли он его действительные потребности, как они выглядят на момент завершения разработки); разницу между этими случаями наглядно иллюстрирует шутливая история разработки, нарисованная в этом пункте. (верификация проверяет, что «задано техзаданием», а валидация - «что надо было на самом деле» :)).

Т.о. валидация проверяет сами требования на соответствие ожиданиям заказчика (конечно, формализованным хотя бы качественно в виде нестрогого техзадания), а верификация — систему на соответствие этим требованиям (прошедшим валидацию).

Тестирование обычно рассматривается как часть валидации продукта. Процедура же проверки выполнения формальных требований на модели продукта относится к процессу верификации.

Почему адекватное матобеспечение появилось недавно? Доказать что-либо формально можно лишь на формальной же модели. Казалось бы, программная реализация деятельности полностью формализована; однако это обычно не так в аспекте семантики. Почему? Дело в том, что когда программная реализация подразумевает что-то неопределяемое по тексту программы (до исполнения), мы не можем говорить, что из этого текста непосредственно следует полное формальное описание её поведения (т.е. как программа будет исполняться). Это «что-то» включает такие обычные сегодня средства программирования, как приближённые вычисления над вещественными числами, использование указателей и вообще произвольных в смысле

[8] Показательно, что в системотехнике при структурировании автоматизированных систем по видам обеспечения изначально под «математическим обеспечением» понималось либо изложение методов реализации назначения АС математическим языком, либо... запись алгоритмов решения на языках высокого уровня.

п/п 2.1.1.2 структур данных, динамическое порождение потоков исполнения (в терминах техноязыка - «рабочих точек») и их взаимодействие.

При каких условиях верификация будет формальной? Можно вручную построить формальную модель системы с формализованной семантикой. Как правило, она будет абстракцией проверяемой системы, в которой сохранены лишь существенные её черты. По сути, это спецификация программы на языке формальной логики (в отличие от привычной формы спецификаций, обычно содержащих требования неточные, неполные и главное — не образующие целостной системы).

Каким же путём мы можем идти к формальной верификации? В настоящее время известны три направления:

- дедуктивная верификация;
- проверка эквивалентности;
- проверка модели.

Сущность двух первых изложена в различных работах[9]. Подробнее остановимся на третьем направлении, более известном под английским наименованием model checking[10].

Сущность метода — проверка того, что на данной формальной модели выполняется (принимает истинное значение) заданная логическая формула. При этом система моделируется системой переходов с конечным числом состояний. Используя т.н. темпоральные логики - неклассического («псевдофизического») рода с участием времени — выражают требуемые свойства поведения системы точно и недвусмысленно. Тогда верификация сводится к исчерпывающему анализу пространства состояний модели, не требующему логического вывода, и потому м.б. автоматизирована.

Результатом единичной верификации (однократного применения метода) м.б. либо подтверждение правильности, либо указание на неправильность модели. Второе реализуется как выдача *контрпримера* — траектории функционирования (переходов), на которой проверяемое свойство (конкретная формула-требование) не выполняется.

Практический смысл имеют спецификации, количественно отражающие временны*е* свойства систем (т.к. реальная система как минимум взаимодействует со внешней средой и обычно в ней совместно протекают ряд процессов), а часто — также и вероятностные (когда моделируются системы стохастические). Это потребовало некоторых новых математических средств.

> Искусственные системы (в т.ч. информатические), для которых внешняя среда выступает не просто как производитель исходных объектов (данных) и потребитель результатных, а как взаимодействующая сущность (которая в п/п 2.1.3.5 названа реальной средой исполнения), определены Карповым как ***реагирующие***[11] (а по Пароннджанову - «реального времени»). Такая среда (окружение) должна представляться как совместно протекающий процесс (система взаимодействующих процессов), взаимодействующая с процессом (системой процессов), протекающим в рассматриваемой системе; механизмы взаимодействия в техноязыке обсуждались нами в п/п 2.1.3.7.

Изначально построенные темпоральные логики были «чёткими», т.е. подразумевали выполнение свойств с вероятностью 0 или 1; время же в них имело «философско-грамматический» смысл периодизации на «прошлое/настоящее/будущее», задаваемое различными отношениями порядка между «точкой свершения» некоего события, «точкой говорения» о нём и «точкой референции», т.е. моментом, на который ссылается высказывание. Это в общем соответствует временам в естественных язы-

[9] Краткое определение дано в: Карпов Ю.Г. Model Checkng. Верификация параллельных и распределённых программных систем. - СПб.: БХВ-Петербург, 2010. - С. 27.

[10] По материалам работы : Карпов Ю.Г., 2010 — Гл.1.

[11] См.: Карпов Ю.Г., 2010. - п. 2.4.

ках[12]. Были выделены два класса таких логик: линейного времени (LTL) и ветвящегося времени (CTL). В первом классе будущее для каждого времени (состояния верифицируемого процесса перед очередным переходом) единственное; тем самым отражается одно возможное вычисление (маршрут алгопроцесса). Во втором классе допускается альтернативный выбор (развилка) в любом состоянии; тем самым отражается потенциальная множественность будущего протекания алгопроцесса. В любом случае это логики качественного анализа.

В дальнейшем были построены количественные логики вероятностного типа (Probabilistic) с возможностью «размытых» суждений о выполнении свойств, а также временного типа (Timed) с привязкой к реальному времени (числовым шкалам, примерно так, как по таймеру в ДРАКОНе-2). Они относятся к классу CTL.

> Важно понимать, что названные классы логик несводимы друг к другу; т.о. для выражения разных свойств нужно использовать логики из соответствующих классов.

Возможно проверять свойства следующих видов:

- достижимости — устанавливают, что некоторые специфические состояния системы м.б. достигнуты;
- безопасности — что некоторые события никогда не произойдут (возможно, при определённых условиях, ограничениях на функционирование системы);
- живости — что при любом функционировании системы некоторые события произойдут (определённое состояние будет достигнуто);
- справедливости — что некоторые события будут наступать неопределённо часто (возможно, на достаточно большом интервале).

Каждый вид свойств м.б. формально определён в той или иной темпоральной логике (но необязательно в каждой); кроме того, Карпов в Гл. 6 своей работы приводит формулировки конкретных свойств каждого вида.

Каково содержание процесса верификации по методу Model checking? Можно определеить его как следующую процедуру:

⇒ Построение формальной спецификации проверяемой системы.

⇒ Формулировка проверяемых требований.

⇒ Выполнение элементарной верификации.

⇒ Анализ результата.

Спецификация м.б. задана как описание структуры переходов, что требует высокой квалификации специалиста. В то же время сегодня стало возможным описывать процессы на императивных языках спецификаций высокого уровня (напоминающих языки параллельного программирования); инфорсима поддержки верификации автоматически строит по такой спецификации структуру переходов. Можно формировать спецификацию, так сказать, «снизу вверх» - анализируя и абстрагируя уже созданную систему, и «сверху вниз» - как укрупнённое описание будущей системы.

Требования изначально формулируются как формулы одной из темпоральных логик. Однако реально при верификации такая формула представляется также неким алгопроцессом. Его можно рассматривать как алгоритмизацию системы переходов, представляющей недопустимые траектории функционирования (т.е. отрицания формулы, выражающей желательное свойство).

Элементарная верификация по данному методу математически заключается в поиске на развёртке структуры переходов маршрутов, на которых не выполняется то или иное заданное свойство. Для разных логик используются разные методы такого поиска; автоматизация обеспечивается алгоритмическим представлением метода на конечных структурах данных (т.е. фактически информатизацией модели).

Анализ результата связан с определением по контрпримеру причин его существования. В связи с неадекватностью модели и реальной системы возможно, что контрпример не относится к по-

[12] Поэтому, кстати, наименование изначально определённой темпоральной логики по-английски — TL — расшифровывается как Tense, а не Time Logic, как можно было бы предположить.

следней, т.е. является артефактом абстрагирования системы. Если контрпример относится и к системе, то необходимо определить и устранить причины его появления; тем самым возвращаемся к фазе создания/совершенствования.

Разработан ряд ModelChecking-систем, автоматизирующих верификацию[13]. Они используют собственные входные языки представления, обычно уникальные для каждой системы; сегодня все они текстовые.

Отметим, что существует ограничение на сложность модели для автоматизированного анализа; однако уже можно проверять довольно сложные реальные системы.

Как же в целом можно оценить формальную верификацию вообще и метод проверки моделей в частности на современном этапе развития? Достоинства состоят в том, что:

- требования проверяются на всех возможных траекториях функционирования;
- можно проводить сразу, как только построена модель системы;
- возможна автоматизация процесса для данного метода.

К недостаткам относятся:

- главное — проверяется не реальная система, а её модель. Она м.б. неадекватной, и тогда проверка не выявит ошибок, имеющихся в реальной системе; в то же время модель может обладать свойствами, отсутствующими в системе;
- формально определить, что такое «полное отсутствие ошибок», невозможно, поэтому верификация не может гарантировать абсолютной правильности системы — она лишь выявляет поддающиеся установлению её неправильности (то же относится и к тестированию);
- язык спецификации м.б. недостаточным для формулирования всех желаемых требований к поведению системы;
- система автоматизированной верификации сама может содержать ошибки.

Всё это означает, что качество верификации напрямую зависит от квалификации персонала. Специалист по формальной верификации должен:

- определять степень адекватности модели относительно проверяемых свойств;
- формулировать именно те требования, которые важно проверить;
- устанавливать, какие поведения модели не свойственны реальной системе, а привнесены абстрагированием; их следует исключать при анализе, т.е. игнорировать предъявление данных траекторий как контрпримеров;
- вырабатывать необходимые изменения как в системе (для устранения ошибок), так и в модели (для повышения её качества в дальнейшем).

Ясно, что это требует глубокого понимания реальной системы и процессов её абстрагирования.

Метод проверки моделей позволяет сосредоточить внимание верификатора на специфицировании системы и анализе результатов. Поэтому по сравнению с неавтоматизируемыми методами он, безусловно, является шагом вперёд. Его недостатки скорее имеют место по отношению к некоей идеальной ситуации, когда любой специалист в сфере проверяемой деятельности мог бы без особых затрат на повышение квалификации начать работать по этому методу. Тем не менее такую ситуацию следует считать желательной по той же причине, что и автоформализацию профессиональных знаний — коль скоро доказательность должна стать обычным явлением, то «отдельных» верификаторов рано или поздно станет нехватать. Для рядового же «предметника» часто деятельность даже по данному методу окажется чересчур сложной.

Как соотносится с этими методами ДРАКОН-МФЗ? Прежде всего, очевидно её применение в процессе верификации для визуализации моделей и требований.

[13] Обзор см. в: Карпов Ю.Г., 2010. - пп. 5.6, 5.7, 12.15.

Кроме того, ДРАКОН и КогниСтиль позволят удобнее представить сами процессы верификации, включая алгоритмы . Тем самым можно рассчитывать на сокращение сложности восприятия, понижение «барьера вхождения» в данную деятельность.

Наконец, внедрение формальной верификации влияет на саму ДРАКОН-МФЗ, конкретизируя работу в фазе создания/совершенствования; очевидно, что эскизно визуализировать нужно сразу с расчётом на верификацию дракон-эскизов.

> Чтобы специфицировать поведение и требования для model checking, необходима достаточно высокая математическая культура. Очевидно, по мере распространения метода будет накапливаться опыт спецификации; часть его будет общедоступна (фактически начало этому накоплению положено в упомянутой работе Ю.Г. Карпова). В связи с этим представляет интерес когнитивно-эргономичное представление спецификаций; для этого необходим техноязык, гибридизированный со входным языком используемой ModelChecking-системы. Если такой язык используется в дракон-системе, то возможно взаимодействие систем по данным. При этом из визуальных алгоритмов работы проверяемой системы и проверяемых требований формируется входной текст для загрузки в ModelChecking-систему; в ходе верификации он может редактироваться; результирующий текст загружается в дракон-систему, обновляя содержание соответствующих визуалов.

Тестирование — проверка самой системы в различных ситуациях применения — имеет свои преимущества и недостатки. Следует считать его взаимодополняющим к верификации.

> В некоторых ModelChecking-системах имеется также возможность тестирования моделей (интерактивного выполнения, примерно как дракон-программ в дракон-диспетчере).

Сочетая тестирование и верификацию (формальную), опытный разработчик может не только устранять обнаруженные ошибки, но и более эффективно совершенствовать систему, чем по результатам одного лишь тестирования. Дело в том, что свойства достижимости, безопасности, живости и справедливости фактически описывают все основные требования к искусственным системам, обычно выражаемые в технических заданиях. Поэтому и формулирование этих свойств, и результаты проверки, тем более представленные наглядно, способствуют более глубокому пониманию работы системы, существенно помогают её совершенствованию.

4.2.2. Другие элементы полного жизненного цикла решения задачи

Здесь не рассматривается содержание выделенных фаз, а лишь выделяются некоторые вопросы, существенные по мнению автора именно для формализации деятельности на техноязыке.

Конечно, список поставленных вопросов неполон; практика визуализации и личное информатическое знание читателя, несомненно, добавят к ним другие и укажут свои пути решения.

4.2.2.1. Фаза эксплуатации решения

В данной фазе машинные программы выполняются на конкретных косавтах (под управлением) . Внимательный читатель из предыдущего, надо полагать, ухватил, что полную империнформодель задачи (руководство процесса) предлагается подразделять на человекоориентированную часть (инструкцию) и машиноориентированную (программу). В процессе пользования программой в соответствии с инструкцией человек вырабатывает навык деятельности, обычно отличающийся в чём-то от предложенного описания и может изложить своё представление в форме хотя бы дракон-эскиза; так инициируется реинжиниринг деятельности.

> Предполагается, что эргономичность и универсальность техноязыка облегчат работнику формирование и отчуждение личной т. зр. на тот или иной техпроцесс; это тем важнее, что даже при наличии формально действующей (сертифицированной) СМК ИСО 900Х фактически её ресурсы ограничены (обычно специфическая работа возлагается на отдел, а то и единственного специалиста по качеству, который физически может что-то сделать для реинжиниринга лишь с участием буквально каждого сотрудника оргсистемы).

Важно, что в руководствах (как и в спецификациях, и в эскизах) ДРАКОН применяется совместно с КогниСтиль. Поэтому дракон-редакторы должны поддерживать работу с элементами КогниСтиль, напр. в заданном для данного документа алфавите (см. п. 1.2.1 Приложения 1

4.2.2.2. Фаза утилизации решения

В этой фазе в отношении искусственного исполнителя (косавта) протекают процессы фондирования оборудования, адаптации кода и переноса данных. Первый и третий процессы вполне очевидны и относительно независимы от характера решения. Под кодом мы понимаем прежде всего исходные чертежи для обеих частей (символическую запись решения); машинный код программ нужно рассматривать как продукт трансляции – процесса, хотя и сложного, но чисто технического и более эффективного в рамках ДРАКОНа: сделали новые чертежи – получили по ним новый машинный код (возможно, задав в трансляторе иные параметры кодогенерации).

Какова же суть утилизации кода при таком подходе? Очевидно, мы выделяем в найденном и эксплуатируемом решении (точнее, в дракон-руководстве процесса) инвариантные структуры деятельности, пригодные для новых задач. По сути, это повторное использование символического кода, уже не ограниченное рамками одной задачи; появляются импер-шаблоны проектирования деятельности. В пределе сообществами сочинителей визуалов формируются банки шаблонов на псевдо-, специф- и гибридных техноязыках; надо полагать, возникнут и общедоступные банки такого рода, подобно свободному ПО и БД. Их можно считать настоящими базами импер-знаний (понятно, ничто не мешает и параллельному существованию баз деклар-знаний, но специфичных по предметным областям и представленных также на когнитивно-эргономичных языках). «Вкладчик» такого банка в качестве «процента» получает доступ к результатам других; кроме того, он может рассчитывать на оценку своих творений, критику (желательно корректную) и участвовать в обобщении содержимого и его конкретизации. Разумеется, будет существовать и коммерческий сектор: его удел прежде всего – визуализация практических задач (бизнес-процессов), исследование конкретной деятельности в визуальном представлении (для чего ещё предстоит наработать теорию анализа и оптимизации визуалов и дракон-моделей), обучение [авто-]формализации профессиональных знаний на базе ДРАКОНа.

Сегодня мы имеем огромный и непрерывно пополняемый банк информатических импер-знаний в текстовом представлении, как для конкретных задач и исполнителей, так и для обобщённых задач и абстрактных исполнителей; главными авторами последних, конечно, следует считать Н. Вирта с его "Структурами данных и алгоритмами" и Д. Кнута с его "Искусством программирования".

Фактически формирование банка визуальных импер-знаний, представленных на техноязыке, уже начато образовавшимися ДРАКОН-сообществами, напр., представленным на веб-форуме /8/. В качестве скромного вклада автора в банк дракон-знаний можно рассматривать иллюстрации данного документа; кроме того, планируется предложить некоторые достаточно простые и имеющие прежде всего учебную направленность примеры (см. Приложение 4).

Импер-шаблоны м.б. как смысловыми, так и литеральными (и абстрактными); последние, по-видимому, будут носить прежде всего учебный и поисковый характер.

4.2.3. О содержании укрупнённых техопераций ТФЗ

Предполагается, что навыки выполнения укрупнённых операций (названия которых в техпроцессе выделены таким образом) получены пользователем ранее (напр., по /1/).

> Под ***укрупнённой операцией*** в автоматизированной среде мы понимаем одну команду приложения автоматизации из числа средств решения задачи (в комплексе с мысленными условиями ручного выбора оператором тех или иных параметров этой команды), либо ряд команд одного или нескольких приложений, которые вместе с условиями ручного выбора порядка (маршрута) выполнения этих команд образуют алгоритм операции.

При работе в офисном пакете детализация всех операций рисования в конечном счёте приводит к командам элеграф-редактора. В их освоении для выбранного пакета определённую помощь, возможно, окажут описания в шаблоне графчасти (см. п/р 3.5 инфордока в данном архиве).

4.3. Основы формальной визуализации

Формальная технология визуализации представляет собой т.н. исчисление икон, т.е. процедуру вывода любой дракон-схемы из заданной исходной формы (выбранной заготовки) применением точных правил вывода из конечного набора с использованием конечных алфавита и словаря; известным примером текстового исчисления служит алгебра логики.

Шампур-метод есть разновидность структурного программирования, полностью формализованная в отличие от известных методов (см. /1, Гл.16/) и более наглядная за счёт эргономизированного графического представления алгоритмов.

Формализация основана на следующем. Кроме икон, в дракон-схеме выделяются и особые вершины – точки ввода (валентные).

> *Точка ввода* – это место разрешённого добавления нового оператора; при этом она считается *нейтральной*, если в нее возможен, но не обязателен ввод оператора и *критической*, если ввод оператора обязателен.

Критические точки ввода определены для всех макроикон, содержащих развилки и циклы. Дракон-схема не закончена, пока в ней есть хотя бы одна критическая точка ввода. В законченной схеме может содержаться любое число нейтральных точек ввода; при исполнении визуала они игнорируются (не влекут за собой действий).

Некоторые иконы (макроиконы) служат основой образования особых формальных объектов ДРАКОНа – атомов; будем называть их атомарными.

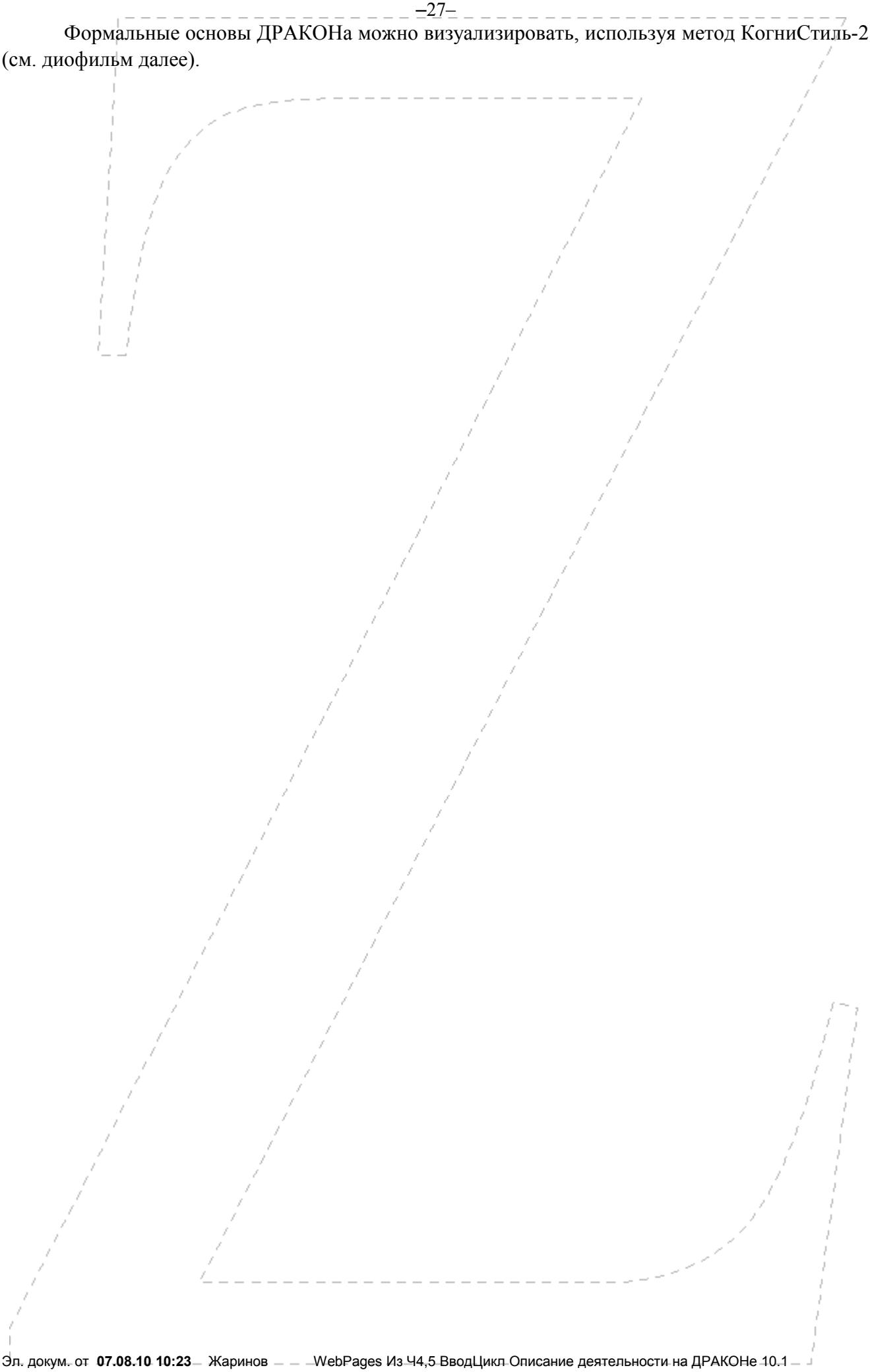
> *Атом* – это визуальный оператор, дополненный двумя нейтральными точками ввода – до входа и после выхода.
>
> *Простой атом* – объект, в основе которого лежит атомарная икона. Простой атом на основе иконы *Комментарий* образует собой особое подмножество *комментариев*; остальные простые атомы образуют другое подмножество – *функциональных* атомов.
>
> *Составной атом* – объект на основе атомарной макроиконы, которая дополнена одним и более атомами на месте имеющихся критических точек ввода. Эти атомы также делятся на подмножества; если критические точки не заполнены, то составной атом считается *пустым*, иначе *непустым*. Как правило, непустой составной атом выводится из соответствующего пустого добавлением простого атома в одну или более критических точек.
>
> *Матрёшка* – разновидность составного атома, в которой пустые и непустые атомы вложены друг в друга; также как атом, бывает пустой или непустой.

Отдельные иконы не являются шампур-блоками; будем называть их неатомарными. Они также м.б. подразделены следующим образом: икона *Период*, имеющая вход и выход на одной горизонтали, образует собой особое подмножество *проходных блоков*; остальные иконы, имеющие один выход сбоку, образуют другое подмножество – т.н. *боковиков*. Все они участвуют в образовании атомов, минуя точки ввода. Этим случаям в исчислении икон также даны формальные определения (случаи присоединения боковика *Формальные параметры* к атомарной иконе, боковика *Синхронизатор* к ряду атомарных макроикон и один непустой атом с проходным блоком, не выводимый из пустого путем добавления простого атома и потому включенный в число аксиом техноязыка).

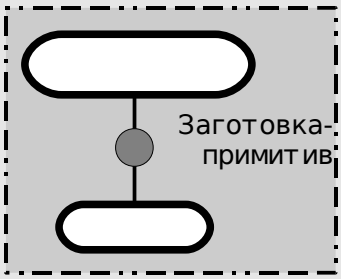
Для введённых таким образом объектов можно определить строго формальные операции соединения и преобразования (правила вывода), обеспечивающие получение правильной дракон-схемы, что и сделано в /2, гл.15/ в виде части набора тезисов. Множества правил для вывода теорем-примитивов (из заготовки-примитива) и теорем-силуэтов (из заготовки-силуэта) частично различны. Общие правила вывода: ввод атома, добавление варианта, пересадка лианы, боковое присоединение; дополнительно для примитивов определено удаление конца примитива, а для силуэтов – добавление ветки, заземление лианы, удаление последней ветки, дополнительный вход.

Отметим, что при построении дракон-схем по правилам шампур-метода достаточно пользоваться ограниченным подалфавитом, содержащим 18 икон и макроикон из общего числа 45; остальные получаются путем вывода из элементов подалфавита, как и дракон-схемы.

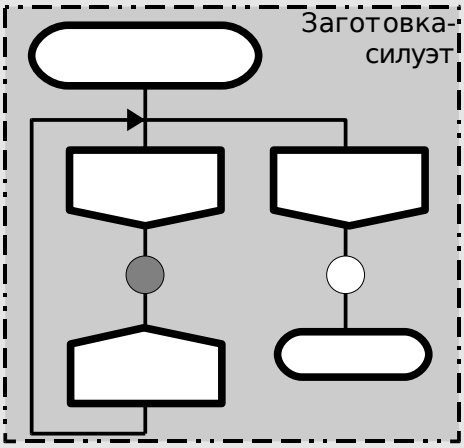
Формальные основы ДРАКОНа можно визуализировать, используя метод КогниСтиль-2 (см. диофильм далее).



Визуализация формальных основ техноязыка



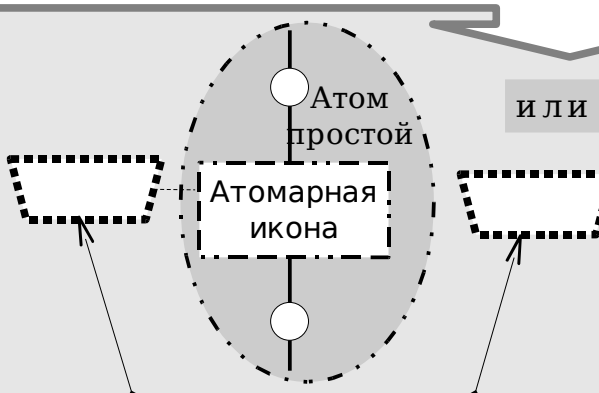
Заготовки - это исходные формы дракон-схем, т.е. допустимых предложений на техноязыке



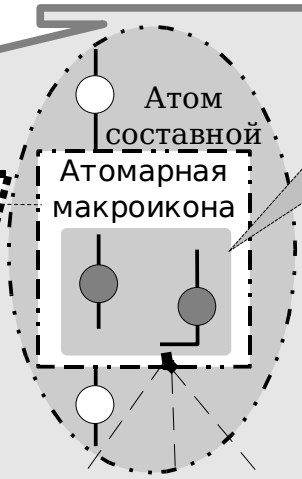
Валентная точка — место разрешённого ввода атома в схему (с разрывом линии). Бывает:
нейтральная — ввод атома в которую возможен, но не обязателен
критическая — ввод атома в которую **обязателен** для завершения схемы

Кроме валентных, вводятся точки формирования соединения/ разветвления линий (см. след. кадры)

Атом - это фигура (шампур-блок), заключающая атомарную икону или макроикону между двумя нейтральными валентными точками. Атом используется в операции Ввод атома как основа (исходная подсхема) для построения любой допустимой в техноязыке дракон-схемы. Имеет варианты ***внутреннего строения:***



или

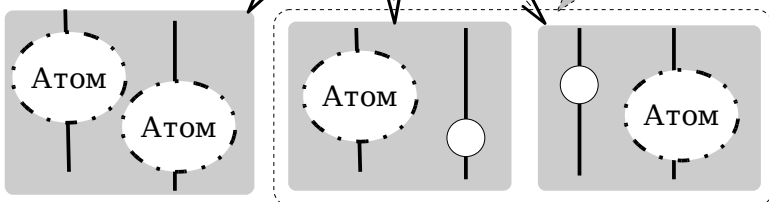


Макроикона изначально имеет две валентные точки для ввода других атомов

При вводе одного атома другая валентная точка считается <u>нейтрализованной</u>, т.е. ввод в неё уже необязателен

Икона *Синхронизатор* (выделена пунктиром) не принадлежит атому, а присоединяется сбоку к нему после ввода в дракон-схему

Если хотя бы один из введённых атомов составной, в дракон-схеме образуется ***матрёшка*** (как-то: сложное ветвление, цикл в цикле, цикл в развилке или наоборот)



Возможные результаты ввода атомов в макроикону

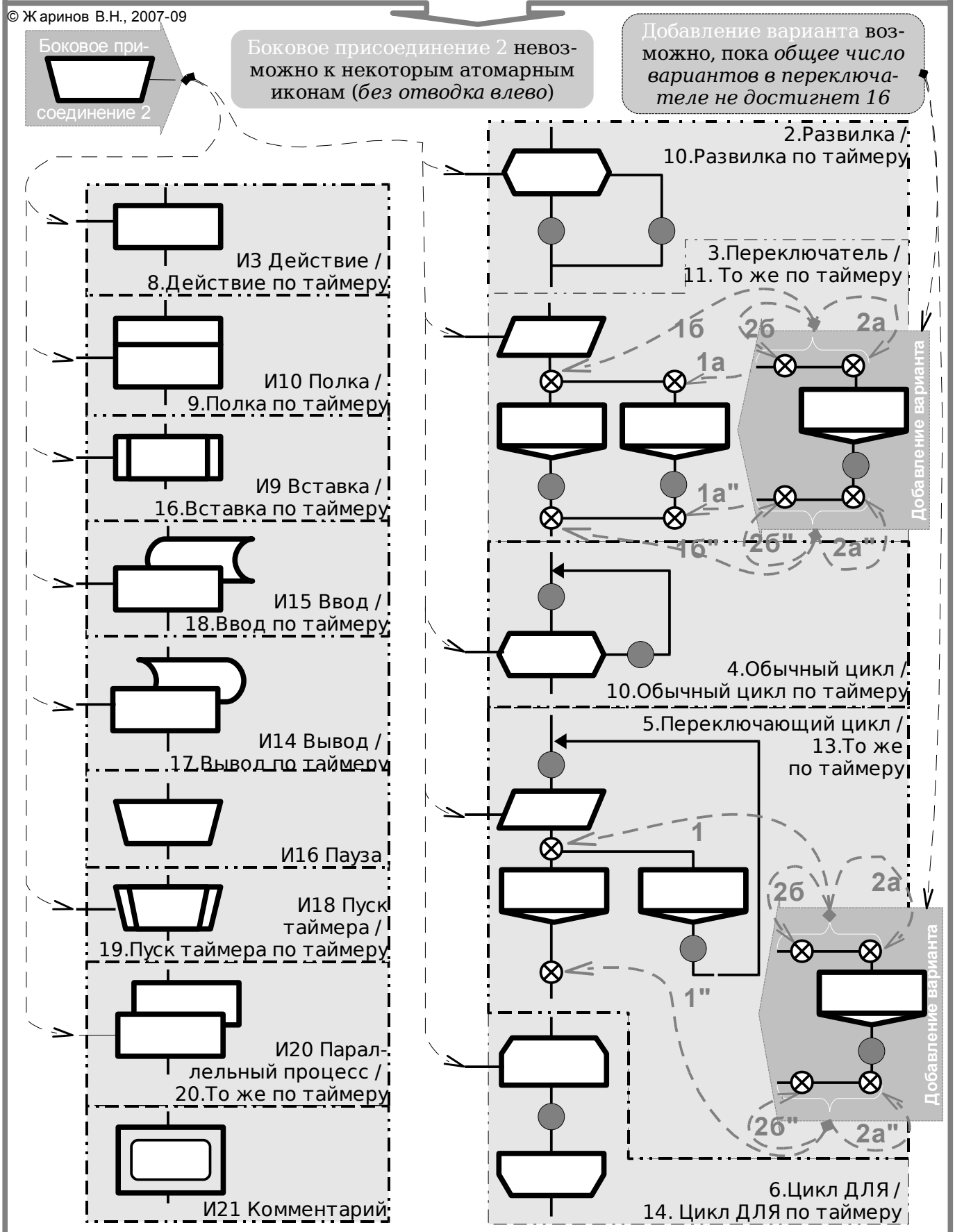
Матрёшкой называется фигура (подсхема), где внутрь составного атома введен другой составной атом (атомы), в т.ч. многократно (по нисходящей)

Составной атом (а равно и матрешка) является *пустым*, пока в нем имеются критические точки и становится *непустым*, когда все эти точки нейтрализованы. Дракон-схема считается ***завершённой*** тогда, когда в ней нет критических точек

Атомарные иконы и макроиконы и операции над ними см. след. кадры

Визуализация формальных основ техноязыка (в 3-х кадрах)

Атомарные иконы — это буквы, а *атомарные макроиконы* - слова техноязыка

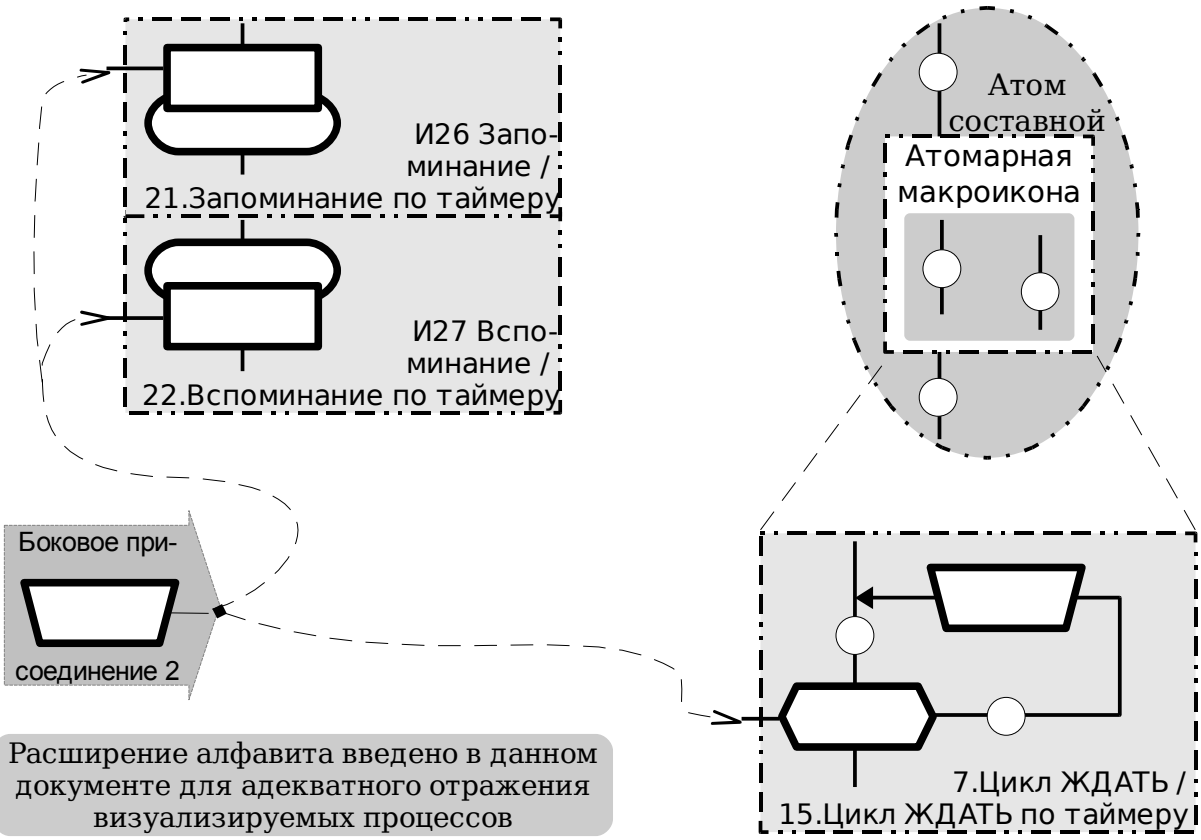


Это иконы *простых* атомов базового алфавита техноязыка. Они образуют простые шампур-блоки ДРАКОН-1 (с присоединением — ДРАКОН-2)

Это макроиконы *изначально пустых составных* атомов базового алфавита. Их множество (см. также <u>след. кадр</u>) сводит возможные ветвления и циклы к шампур-блокам, включаемым в атомы

Показанные ниже иконы образуют расширение базового алфавита (см. тот же столбец на пред. кадре). Они участвуют в образовании простых атомов ДРАКОН-1 (а с присоединением — ДРАКОН-2) так же, как другие атомарные иконы

Показанные ниже макроиконы образуют *изначально непустой составной атом* — особый подвид составных атомов, не требующий обязательного ввода икон внутрь. За исключением этого, он участвует в операции Ввод атома как другие (изначально пустые) составные атомы



Любое произвольное ветвление, допустимое в визуале (в т.ч. внутри цикла) имеет в основе матрёшку, полученную вводом одних составных атомов внутрь других; для завершения построения в этой матрешке проводятся ***пересадки лиан***. Полученная структура наполняется содержанием путем вставок простых атомов в её вертикали и боковых присоединений

Говоря проще, по правилам шампур-метода из всех икон лишь несколько используются самостоятельно в составе конструктивных элементов дракон-схем (простых атомов). Остальные сначала объединяют в «строительные заготовки», т.е. макроиконы, изображённые в Приложении 1. Из макроикон также лишь несколько используются для построения атомов (т.н. составных); остальные получаются боковым присоединением икон-модификаторов реального времени (при необходимости его алгоритмизации). Структуры маршрутов визуала это присоединение не изменяет, как и добавление икон-атомов внутрь макроикон-атомов.

При визуализации операций техноязыка использован метод КогниСтиль-2. Поясним его принцип на примере операции Добавление варианта (в переключатель).

Стрелки (направляющие) соединяют точки исходного объекта действия (прибавляемой заготовки-лианы) с точками результатного (расширяемого переключателя). Возможны разные случаи операции, что показано индексами линий. Цифровая часть индекса показывает различные очереди добавления (1 – начальная, в двухвариантный переключатель, 2 – после первой очереди, в уже расширявшийся); буквенная – различные возможности добавления в пределах очереди (поскольку у нас исчисление, то порядок их использования языком не устанавливается, выбор на усмотрение сочинителя конкретной схемы).

Отметим, что точки добавления – это места формирования разветвительных вершин (в верхней горизонтали) и соединительных (в нижней); те и другие напомним, показываются по правилам техноязыка простым слиянием линий. В то же время эти вершины сродни нейтральным точкам ввода, показывая допустимые в будущем места добавления; поэтому две вершины включены уже в макроикону (для случая 1а).

Одновременно в операции участвуют пары стрелок, имеющие одинаковый индекс, лишь для нижней добавлен штрих. Скобками показано, что в добавлении участвует не точка, а участок исходного объекта, для которого на конечном объекте делается разрыв (аналогично вводу атома в точку). Кроме того, имеем стрелки, возвращающиеся к исходному объекту (прибавление объекта к самому себе, т.е. своего рода визуальная рекурсия); так сокращённо показана кратность операции с продолжением расширения переключателя в ту или иную сторону от уже добавленного варианта (разумеется, кратность д.б. как-то ограничена; в нашем случае есть правило, что число вариантов не может превышать 16, данное на схеме в пояснении).

Можно сократить число вариантов, оставив только по одной возможности в каждой очереди, но тогда мы должны в уже заполненном переключателе делать рокировки, т.к. порядок вариантов с учётом новых может нас не устроить.

Переключающий цикл имеет несколько иную топологию, и здесь исходные точки добавления неравнозначны; верхняя уже есть разветвительная вершина, тогда как нижняя ещё не есть соединительная, а просто точка разрыва/вставки.

Если потребовать равнозначности исходных точек, то получится, что прибавление происходит как бы «углом», т.е. результатные точки неодинаково удалены от вертикали (лианы). Поэтому придётся иначе задать и конфигурацию заготовки-лианы, по-иному выбрать точки добавления.

4.4. Основы визуального программирования

Под визуальным программированием понимаем получение текста в системе команд предполагаемого исполнителя (целевой) на основании ***комплекта исходных чертежей*** - совокупности граф-схем, в т.ч. на различных языках, описывающей некий процесс в терминах формального исполнителя. При этом структурная часть процесса описывается строением граф-схем, а часть, касающаяся содержания элементов структуры - значениями атрибутов элементов граф-схем.

Возможны два уровня решения задачи:

- непосредственное отображение в коды команд из дракон-системы;
- получение промежуточного исходного текста, который кодируется внешней программой-транслятором.

Во втором случае мы можем принять различные языки исходного текста. Здесь также возможны два уровня:

- язык высокого уровня;
- язык Ассемблера для целевой системы команд.

Во втором случае каждому визуальному оператору соответствует одна команда исполнителя. В первом случае каждый визуальный оператор в итоге м.б. отображён более чем одной командой; правило отображения для каждого ЯВУ реализовано в трансляторе с этого языка.

В качестве формального языка текста граф-схем процессов (исходных чертежей программ) д.б. принят по крайней мере один промышленный прогязык реального времени, обладающий максимально высоким научно-техническим уровнем и имеющий реализацию в виде компилятора исходных текстов с этого языка.

В соответствии с мнением отечественных ИТ-специалистов, представленных на веб-конференции OberonCore.Ru, исходным м.б. выбран прогязык Активный Оберон.

Фактически к представлению модели как исходных чертежей добавляется представление её как исходного текста. Понятно, что между языком исходного текста и семейством языков исходных чертежей д.б. определено взаимно-однозначное соответствие.

Следует также обеспечить единство источника — из нескольких представлений только одно рассматривается как содержание модели, а все остальные получаются «переводом» на язык нужного представления в среде поддержки. И это можно реализовать по-разному, выбрав за базовый:

- язык исходного текста;
- языки описания визуальных моделей.

Точнее говорить о метаязыке, имея в виду классификацию отчуждённого знания на общее и частное и далее частных знаний — по видам.

> Конечно, в любом случае машинный документ, содержащий полиязыковую модель, должен содержать при каждой схеме указание на тип её языка (напр. в виде префикса заголовка). Оно используется средой сочинения для выбора языкового модуля работы с этой схемой.

Имеет смысл выбрать базовым текстовое представление; тогда визуальные модели получаются трансляцией текста. Однако метаязык этого текста будет, очевидно, шире прогязыка дракон-программы; в него будет входить и язык[и] обобщённой формализации задачи, и язык схематизации содержания документа. Поэтому нужна техоперация трансляции с метаязыка на язык исходного текста; в результате её должен получаться отдельный документ в формате внешнего транслятора с этого языка.

Среди уровней прогязыков логично принять за базовый язык высокого уровня. Тогда низкоуровневое представление (на Ассемблере или в кодах) получается также трансляцией (в отдельный документ в формате использования — напр. программатора, загрузчика кода).

Понятно, что режимы трансляции и набор поддерживаемых языков зависят от конкретной реализации ИСП с функциями визуального программирования (которую мы называем РДП-системой).

Т.о. реализуется двусторонняя трансляция формальных моделей чертежи↔текст. И это можно реализовать разными способами:

- оперативным — немедленно по редактировании одного представления обновляется другое;
- отложенным — обновление выполняется в определённых случаях, напр. при сохранении документа и/или по команде сочинителя.

В первом случае специальной команды обновления, конечно, не требуется.

В принципе это то же, что режимы on-line и off-line взаимодействия систем или интерактивный и пакетный режимы доступа (исполнения заданий оператора).

Что это даёт для техпроцесса сочинения? Во-первых, сочинитель не связан каким-то одним представлением — может работать как с графикой, так и с текстом. Технология формализации получается естественной - правка схем отражается (автоматически) на тексте и наоборот. При поддержке нескольких языков по одним и тем же чертежам можно получить текст на каждом из них (удобно также транслировать разные дракон-программы на разные языки).

Имея возможность иерархической детализации согласно п/п 2.1.3.12, мы можем реализовать «сборочную визуализацию» и алгоритмических, и иных моделей. Для этого нужно только организовать и вести каталоги фрагментов моделей, оформленных как автономные сущности; логично считать их областями, не связанными ни с какой схемой.

В каталоге мы можем поставить каталогизированной области на некотором языке в соответствие икону на более высокоуровневом языке (напр. области-схеме на дракон-ассемблере - оператор ДО-2).

> Исходя из того, что согласно п/п Смысл и текст икон п. 2.1.2 определён шаблон синтаксиса текста на любом языке), мы можем абстрагировать имена объектов в тексте и иконы, и детализирующей её области. Вместо имён в текстах появляются поля объектов — переменные-"иксы", формируемые условно; вхождению одного и того же имени в исходной схеме соответствует вхождение соответствующего условного имени по каталогу. При использовании в конкретной схеме сочинитель должен указать имя из числа употребляемых в этой схеме для каждого поля для обратной конкретизации.

При этих условиях можно, выбрав язык для детализации и подгрузив нужные каталоги, при выборе очередной иконы детализируемой схемы замещать её ранее построенной структурой, а не строить детализацию каждый раз заново. Понятно, насколько это удобнее сочинителю.

> Естественно, соответствие устанавливается в абстрагированной форме — сначала текст детализируемой иконы приводится к шаблону, а затем проверяется совпадение (по местам вхождений и типам) с иконой из каталога (текст котрой также абстрагирован по определению).
>
> Возможно, что одной иконе в каталоге соответствует более одной области; тогда сочинитель должен выбрать нужную.
>
> Если используется нешампур-область, то можно допустить и «подгонку по месту» связей выбранной структуры путём операций с лианой.

Естественно, каталоги можно вести при любом варианте ТФЗ — хоть вручную — но наибольший эффект даст каталогизация областей в РДП-системе (среде). При этом автоматизируется и проверка соответствия, для чего РДП-редактором используется шаблон синтаксиса; сочинителю же нужно по большей части просто отвечать на вопрос о "визуальном автозаполнении" области очередной иконы из подходящей по определению области каталога.

Лист-каталог, по-видимому, д.б. одноязыковым; он сохраняется и открывается отдельно. Сочинитель может создавать столько каталогов, сколько ему нужно.

Визуальное ассемблирование по «областной» технологии можно провести следующим образом. Каждый оператор дракон-программы детализируется областью, причём:

- конструкции записываются в лиоформе, как определено в п. 2.2.2 для ветвлений и в п. 2.2.3 для циклов; началом главной вертикали служит икона Заголовок, любой другой вертикали – икона Имя ветки;
- икона Развилка представляется как команда ветвления (см. п. 2.2.2);
- в иконе Имя ветки записывается метка (адрес памяти программ, с которого начинается озаглавленная ею вертикаль), в любой другой иконе – полный текст одной команды (в т.ч. в развилке – команды условного перехода УП, Адрес ветки – команды БП).
- икона Конец трактуется как команда останова машины, а при использовании в конце вставки – как команда возврата из подпрограммы;
- любая шампур-икона визуализируется как шампур-блок в минимальном базисе (веточные иконы представляют безусловные переходы, икона Вставка – команду вызова подпрограммы, икона Действие – любые другие команды); здесь сочинитель определяет выкладку вертикалей в памяти информашины, вводя БП по своему усмотрению.

Т.о. иконы рассматриваются как «контейнеры» для команд ассемблера.

Можно расширить базис, использовав для некоторых машинных команд подходящие по смыслу иконы. Прежде всего это касается ввода и вывода; также можно использовать икону Полка для визуализации команд пересылки (синтаксис очевиден, если иметь в виду смысл полки как присваивания).

Такую схему можно оттранслировать в текст на ассемблере по относительно простым правилам (которые д.б. определены в редакторе схем). Очевидно, что редактор должен контролировать, чтобы адреса БП указывали на существующие метки, а на каждую метку приходился хотя бы один БП (указывая сочинителю на нарушения). Также удобно снабжать каждую икону её начальным адресом (вычисляемым относительно головной иконы вертикали). Ну и разумеется следует автоматически контролировать правильность написания команд (как для любого поддерживаемого языка «костюма»); возможно, стоит предлагать автозаполнение.

В принципе для данного метода визуализации достаточно, чтобы редактор просто допускал рисование схемы-лиоформы (т.е. вставку икон БП в любых местах визуала) и мог формировать исходный текст из всех текстов икон схемы, «разбирая» их по вертикалям (точнее – по цепочкам между меткой и командой БП); этот текст можно будет передать в транслятор с Ассемблера. Правда, ручной контроль адресов БП достаточно сложен, но облегчается эргономичным представлением структуры маршрутов в виде дракон-схемы.

Можно ввести поддержку ассемблера в редактор; тогда нешампур-иконам, как обычно при гибридном программировании, сопоставляются маршрутные части команд. Однако в случае языка такого рода это не совсем удобно из-за иного, чем в ЯВУ, способа построения операторов; напр. мнемотекст (а равно и машинный код операции) команды УП фактически должен складываться из общей части, представленной как икона, и спецчасти, представленной знаком условия УП в тексте этой иконы – нагляднее будет записать команду полностью в тексте иконы.

5. РЕАЛИЗАЦИЯ И ПРИМЕНЕНИЕ ТЕХНОЯЗЫКА

5.1. Обзор реализации ДРАКОНа

5.1.1. Вводные замечания

ДРАКОН, как и любая МФЗ, реализуется как некая информатическая система поддержки «говорения» на техноязыке. Ранее уже говорилось, что эта система м.б. в той или иной степени автоматизированной.

При обсуждении стандарта техноязыка в Разд. 2 мы делали различные предположения о реализации отдельных его операторов (т.е. в фазе эксплуатации визуального алгоритма). Ещё более активно эти вопросы обсуждаются в ДРАКОН-сообществе; мы здесь не будем их касаться, а обсудим реализацию фазы создания/совершенствования визуалов (во многом сказанное будет справедливо и для фазы их утилизации). При этом ограничимся рассмотрением создания системы реализации фазы как процесса системной интеграции, т.е. выбора готовых решений и увязки их в целостную ИСП; будем называть её, как и в Разд.4, дракон-системой. Очевидно, нам придётся столкнуться с наличием целостных ИСП-ДРАКОН, но поддерживающих ТФЗ-ДРАКОН не в полном обьёме; будем называть такие изделия дракон-средами.

Ограничим теперь круг рассматриваемых изделий. С т. зр. области применения нас интересуют прежде всего дракон-системы (среды) не узкоспециализированные, но широкого применения. В плане исполнения рассмотрим инфопрограммные изделия (не требующие специализированной аппаратуры), работающие на платформе ПК ИБМ с массовыми семействами ОС, такими как Windows, LINUX, использующие графический интерфейс оператора (ГИО), поддерживаемый ОС. Также ограничимся некоммерческими изделиями (или изданиями), которые можно применять на платформе, включающей также некоммерческие ОС. И конечно, прежде всего глядя на эксплуатационные качества с т. зр. оператора-пользователя, мы будем иметь в виду, что ему порой придётся исполнять и некоторые функции операторов других категорий (сервиса и администрирования).

Дракон-система (среда) – это инфопрогизделие прежде всего тиражного вида (приложение или пакет). Разработчиком его м.б. чаще всего ИТ-специалист, который может относиться к изделию и как к утилитарному ("для себя", а точнее – для человека с квалификацией программиста или хотя бы системного аналитика); как следствие, оно не обязательно будет удобно для неспециалиста, желающего использовать дракон-среду при формализации профессиональных знаний вне ИТ-сферы.

Пользователь же при выборе среды может руководствоваться различными критериями. Существуют различные профессиональные системы показателей оценки, но мы здесь введём краткую; после каждой группы критериев даются их качественные определения.

Среди критериев можно выделить **общие** для любого приложения, такие как:

- эргономическая эффективность интерфейса, т.е. диалогов и рабочих форм;
- когнитивная эффективность как степень сокращения умственных усилий на освоение и эксплуатацию изделия за счёт его символически-информационных свойств;
- техническая эффективность как способность качественно функционировать в заданном окружении при нормальных условиях эксплуатации;
- функциональная гарантоспособность как выполнимость любой функции (команды) изделия при конкретных конфигурации и параметрах окружения, условиях внешней среды, считаемых нормальными;
- целевая гарантоспособность как отсутствие трудно обнаруживаемых и устраняемых ***уязвимостей*** изделия (применительно к инфопрогам – программного кода и массивов данных) от несанкционированного доступа, ошибочного или стихийного внешнего воздействия (в т.ч. от неожиданных изменений внешних условий), внутренних отказов и сбоев, т.н. *неде-*

кларированных (не соответствующих явно указанному назначению изделия) *возможностей* (*НДВ*) его использования, создающих угрозу обрабатываемым данным и иным средствам автоматизации (оборудованию, другим приложениям, системе в целом), а также человеку;

- поддержка как приложения, так и пользователей;
- степень открытости для независимого анализа и модернизации.

Типовые диалоги (для часто выполняемых техопераций) д.б. логично и просто организуемыми через ГИО, а рабочие формы (окна документа и диалога ГИО, форматы объектов) – удобными и естественными для предметной области носителя знаний.

В случае широкого применения естественность относительна и часто может пониматься просто как следование общим правилам построения форм; плюсом будет предоставление наряду с этим пользователю удобных средств настройки символических форматов интерфейсных объектов.

Для инфопрогизделий в основе когнитивной эффективности лежат наличие и продуманность справки, как обособленной (дополняющей, а часто заменяющей традиционное руководство по эксплуатации), так и оперативной (контекстной). Качественная справка должна не просто отвечать на вопросы, возникающие у пользователя по ходу работы (это одна из основных функций для её контекстной компоненты), но способствовать быстрому формированию комплексных навыков эффективного применения изделий (примерно это ныне называют модным словечком «компетенции») и дальнейшему устойчивому развитию этих навыков по мере обновления характера и сложности задач, смены предметной области. Для этого имеет смысл исходить из «правила братьев Стругацких»: инструкция – это для тех, кто ещё не умеет; иначе говоря, в рациональном описании должны отражаться законченные фрагменты работы с изделием и так, чтобы человек понял их внутреннюю логику и закономерности взаимосвязи в зависимости от характера работы и мог затем организовать её по своему усмотрению в изменяющейся обстановке применения. Один из путей для этого – встраивание сценариев типичных операций в виде т.н. мастеров – часто применяется, но ограничивает пользователя, иногда чересчур укрупняя законченные техоперации и сужая их множество, тем самым усложняя «сборку» из них произвольного техпроцесса. Другой путь – рассмотрение в справке сквозных примеров работы с изделием от начала до конца; но кроме этого, надо ещё рационально составлять описание собственно возможностей изделия; простого перечисления команд ныне уже недостаточно.

Возможный путь повышения когнитивного качества инфопрогизделий указывают нормативы СМК ИСО900Х. В них определены как вид НТД рабочие инструкции (РИ) по конкретным трудовым процессам. Подобную инструкцию можно составить для процесса применения изделия (инфопрога); по сути, в терминах /7/ она служит руководящим методическим материалом (РММ). Как правило, для подготовки необходима переработка справки инфопрога и эксперименты с ним для выявления умолчаний, косвенных определений, неявных связей материала и т.п. Для нашего случая можно рассматривать выполнение техопераций формализации знаний.

Для массовых изделий вместо выполнимости часто задаются, наоборот, требованиями разработчика к окружению. Тем самым отчасти гарантоспособность подменяется эффективностью.

Уязвимости и НДВ инфопрогизделий широкого применения, как правило, связаны прежде всего с недостатками технологии программирования и несовершенством спецификаций (техзадания), а также с порочностью заложенной концепции построения программного кода. Подобные пороки отмечаются прежде всего у объектно-ориентированного подхода[14] и обычно усугубляются его массовыми реализациями (коммерческими и учебно-демонстрационными средами программирования) так, что исправить их за счёт определённого написания (оптимизации) исходного текста чересчур сложно, а порой и невозможно. Изделия спецназначения, конечно, проходят специальную проверку по установленным правилами силами специалистов (прежде всего – организаций, уполномоченных на деятельность в сфере информатической безопасно-

[14] Напр., в /6, с.162-163/.

сти[15]), но поскольку их функционирование неотделимо от поддерживающей ОС, а для неё как изделия часто характерны те же недостатки, то должный уровень гарантоспособности в этих случаях достигается установкой также специально разработанных ОС (или приспособленных изданий массовых ОС), которые чаще всего широкому потребителю недоступны.

Тестирование массовых, тем более некоммерческих инфопрогов фактически возлагается на пользователей. Именно это мы должны полагать наиболее вероятным и для дракон-систем.

Уровень эффективности инфопрога может считаться приемлемым, если он выполняется без перенастройки (доработки) на массовых конфигурациях платформ и м.б. связан по данным (входным и выходным файлам и/или массивам) с другими изделиями, обычно необходимыми по действующей технологии применения (в нашем случае – по той или иной разновидности ТФЗ); подразумевается, что запуск в работу осуществляется оператором, а окончание работы – как вручную, так и автоматически (по завершении детерминированной последовательности действий, т.н. сценария применения). При оптимальном уровне инфопрог должен допускать пользовательскую настройку конфигурации (способов и параметров выполнения функций, включая форматы данных) под окружение в достаточных пределах и расширение (замену) функций по заданию пользователя, иметь возможность определения сценариев применения как полноценных алгоритмов, связи по командам с другими изделиями (т.е. быть вызываем ими и вызывать их сам автоматически в рамках неких сценариев). Адаптивный уровень подразумевает, что инфопрог распознаёт ситуацию применения в динамике и, как правило, способен либо сам настраивать параметры имеющейся конфигурации, либо находить возможности её расширения (обновления).

Взаимодействие инфопрогов по данным реализуется различным образом. Одно направление – расширение круга форматов, как входных (открываемых файлов) и выходных (сохраняемых файлов). Другое – возможность оперативного переноса данных (в первую очередь через карманы в ОП, т.н. буферы обмена) за счёт унификации определений форматов. Оптимизация часто реализуется путём введения возможности работы со внешними модулями (т.н. плагинами) переменного состава.

Очевидно, адаптивный уровень вряд ли достижим в ближайшее время в широко применяемых изделиях поддержки формализации знаний; примером достижения этого уровня среди массовых изделий, конечно, служат некоторые защитные средства (антивирусы, брандмауэры и т.п.).

Поддержка требует решения как текущих проблем эксплуатации, так и задач развития изделия. При этом выделяются (как создателями, так и пользователями) более и менее настоятельные вопросы, первоочередные и перспективные задачи развития. Необходима также действенная обратная связь (через развитые инфокоммуникации); желательно и наличие среди участников процесса квалифицированных пользователей, способных ставить вопросы ближе к терминам ИТ-сферы, учитывая когнитивно-эргономический аспект.

В нашем случае эта задача облегчается тем, что сама цель дракон-систем включает развитие этого аспекта и поддержана необходимым уровнем знаний создателя техноязыка. Поддержка некоммерческого изделия определяется, конечно же, доброй волей разработчика, поэтому предложения по развитию вряд ли можно ставить ультимативно. Тем не менее и в этом случае ранжированные проблемы и задачи уместны, т.к. служат ориентиром для других разработчиков, в т.ч. потенциальных.

При открытости исходного кода и форматов данных приложений облегчается поддержка и развитие изделия; фактически возможно образование широкого сообщества разработчиков и постановщиков задач (спецификаторов) изделия. Также лучше обнаруживаются и устраняются уязвимости исходного кода и/или структур данных.

Среди **частных** критериев **эффективности** инфопрога можно выделить:

- сценарные языки (управления заданиями на выполнение) и показатели их реализации;
- наличие модулей расширения для работы с нужными типами содержания, не поддержанными внутри, и качество их работы в сравнении с аналогичными внешними приложениями;

[15] У нас именуемой «защита информации».

- удобство интерфейса со внешними приложениями для отсутствующих функций (обычно через меню конфигурации должен настраиваться автоматический вызов для заданных типов файлов);

Для ИСП формализации знаний можно выделить такие **специфические** критерии, как:

- поддерживаемые стандарты ЯПЗ и корректность их реализации;
- удобство отображения моделей как документов;
- возможности анализа моделей.

Вопрос о ЯПЗ – это прежде всего вопрос, представляем мы только форму (символический вид) модели, или также её содержание. Если предполагается первое, то достаточен язык форматирования графики, текста и таблиц; если второе, то требуется определить состав и структуру языков по назначениям, а затем и требования к каждому языку. Об этом подробнее см. п/р 5.3.

В отношении анализа предполагается, что дракон-система (среда) по крайней мере проверяет сочиняемые схемы на правильность.

Возможны более сложные виды анализа частных моделей; однако встаёт вопрос о теоретической основе такого анализа. Если для деклар-знания имеется аппарат реляционной алгебры, из которого следует система т.н. нормальных форм структуры данных как иерархия приближения структуры отношений между элементами данных к информатически реализуемой (содержащей только отношения вида 1:1 и/или 1:N без аномалий), то применительно к импер-знанию, возможно, нужно ещё определить цели и методы анализа структур деятельности (маршрутов визуалов и дракон-моделей). Свои цели и критерии анализа возможны и для элементной компоненты частных моделей.

Возможны разные концепции отображения моделей в форме как видеограмм (на экране и иных сигнальных носителях), так и машинограмм (в твёрдых копиях и на иных знаковых носителях). На практике следует руководствоваться собственным опытом и мнением квалифицированных пользователей.

Для надлежащей оценки следует вначале сформулировать назначение системы (в данном случае дракон-среды), структурированное так, как это принято в системотехнике (т.е. на главное и вспомогательное, целевое и функциональное) и определить показатели назначения, а затем степень их достижения. Подобные оценки часто дают сами разработчики.

Можно предварительно определить типовую структуру ИСП (подразделение на отдельные приложения).

Для полной автоматизации создателем техноязыка предложена концепция построения ИСП-ДРАКОН, в основе которой лежат программы дракон-редактор (среда создания) и дракон-диспетчер (среда исполнения). Можно принять её за базовую в отсутствие других обоснованных типовых концепций. Вкратце она состоит в следующем.

Дракон-редактор функционирует на основе правил исчисления икон, введённых в /1, гл. 14-15/. Практически это значит, что все дракон-схемы выводятся из двух аксиом (заготовок примитива и силуэта) на основе алфавита (набора икон) и синтаксиса (заданного как набор тезисов) применением однозначных правил вывода необходимых дракон-схем. Строгое следование правилам означает, что дракон-схема будет синтаксически правильной в маршрутной части. Тогда верификация визуалов сводится к проверке правильности содержания икон (соблюдения правил используемых командного и декларативного языков), т.е. объём работы по доказательству правильности алгоритма сокращается.

Максимальное число веток силуэта в стандартном дракон-редакторе – 16. Этого достаточно, чтобы изобразить силуэт на одном листе, не слишком «тесня» иконы; разумеется, речь идет не о машинописном листе, а о произвольном формате.

Дракон-диспетчер обеспечивает выполнение визуальных алгоритмов в условном или реальном (при использовании версии ДРАКОН-2) времени. Возможно интерпретирующее исполнение визуала (дракон-модели) с отображением хода исполнения пользователю показом дви-

жения рабочей точки на дракон-схеме; это наглядно и полезно для целей обучения языку и изучения работы построенного алгоритма.

Для автоматизации гибридного программирования предлагается дракон-конвертор – программа, преобразующая дракон-программу на языке ДРАКОН-X в исходный текст для системы программирования на языке X.

Поскольку даже проникшись содержанием данного или других документов, читатель вряд ли сразу побежит приобретать коммерческую дракон-систему :)), да и далеко не всегда это нужно, подробнее остановимся на других способах реализации.

5.2.2. Средства общего назначения

Простейший путь автоматизации создания визуалов – подготовка дракон-схем в среде машинного документооборота. При этом сориентируемся на открытые средства, предоставляемые по бесплатной лицензии.

В такой постановке задача оформления дракон-схем становится частным случаем подготовки эледоков. Понятно, что технология подготовки д.б. индустриальной, интенсифицированной, а значит, нужны типовые формы документов и элементов их содержания, оптимальные для массового пользователя технологические приёмы, системно изложенные для конкретного приложения (пакета). Это тем существеннее, что контроль синтаксиса дракон-схем при неполной автоматизации возлагается на человека, и ему нужно «подстелить дорожку» в этом отношении.

Наиболее очевидным является применение для визуализации офисных пакетов. При этом удобнее применять специализированные редакторы блок-схем, которые допускают пользовательские форматы блоков и автоматически поддерживают соединительные линии между точками, указанными пользователем, с устранением неоправданных изломов. Работа в таком редакторе существенно комфортнее для пользователя, чем в обычном редакторе рисунков.

Широко распространённый пакет *OpenOffice.org* имеет графический редактор *Draw*, предоставляющий указанные возможности. Подготовленные в нём рисунки можно вставлять в документы, подготовленные в инфордок-процессоре (напр. *Writer* из того же пакета); в результате получается комплексный (текстографический) инфордок.

В целом как изделие данный пакет можно оценить высоко: он открыт, распространяется свободно, реализован для различных семейств и моделей ОС, в отличие от закрытых пакетов использует открытую спецификацию форматов документов, устойчив в работе, его документы достаточно редко повреждаются, а повреждённые – достаточно часто автоматически восстанавливаются. Интерфейс пакета эргономичен.

Для поддержки индустриальной визуализации в среде данного пакета используются:

- каталог графоэлементов техноязыка и КогниСтиль;
- шаблон файла рисунков *Draw*, содержащий заготовки дракон-схем;
- шаблон файла комплексного эледока *Writer* для графчасти, включающий описание реквизитов и элементов технологии подготовки иллюстраций и вёрстки их совместно с текстом.

Технологию оформления дракон-схемы мы визуализировали в виде примера использования ДРАКОНа именно для *OpenOffice.org*. Пример и шаблоны доступны в п. 1.1.1 Приложения 4.

Упомянутый пакет имеет свой веб-узел, где доступны его актуальные версии (см. Полезные ресурсы в Разд.6).

Несколько большие возможности обеспечивают системы моделирования бизнес-процессов, такие как *AllFusion Process Modeller* (бывший *BPwin*). В этом классе имеется свободно распространяемый пакет *DesignIDEF*, доступный в интернет. Он предоставляет возможность преобразования форм графоэлементов вручную; поэтому можно в принципе создать изображе-

ния икон техноязыка (как единые объекты). Пользовательские описания стандартных графоэлементов можно сохранять как их текстовые атрибуты, выводимые на печать в виде отдельных текстовых страниц. Справка содержит описания типовых процедур (укрупнённых операций моделирования), вынесенные в раздел Procedures; тем самым повышается её когнитивное качество в соответствии с критериями п. 5.1.1.

> **Совет:** прежде чем загружать свободно распространяемые инфопроги через интернет, поищите их в более локальных источниках. Возможно, нужные версии доступны в территориальной сети одного из Ваших региональных телеком-операторов, в продаже на носителях по разумной цене, наконец, есть у Ваших знакомых или коллег.

Для целей первоначальной формализации знаний можно рисовать схемы вручную или пользоваться любым графическим редактором, что мы и продемонстрировали. Однако строить дракон-схемы, разумеется, лучше всего в дракон-редакторе. Уже созданы автономные редакторы; появились и трансляторы с гибридных прогязыков.

5.2.3. Обзор специализированных дракон-сред

Пока таковых насчитываются единицы. Здесь даётся предварительный обзор дракон-сред и систем.

Первой следует считать дракон-систему ГРАФИТ-ФЛОКС. Она разработана для применения техноязыка при создании законченных косавтов по полному циклу НИОКР.Строго говоря, в этой системе реализовано семейство языков ДРАКОН (графовый импер-язык) и ФЛОКС (табличный деклар-язык), применяемое по технологии ГРАФИТ. Обзор содержится в этой теме.

В начале 1990-х годов Л. Эйсымонтом был разработан автономный дракон-редактор. На сегодня он не считается удобным, но м.б. освоен для общего представления и сравнения с другими изделиями.

Сегодня основной дракон-средой широкого применения является разработка Г. Тышова (будем называть её Ты-средой). Её оценка приведена в авторском описании (см. Полезные ресурсы). Общий вывод — при всех достоинствах выбранного подхода к реализации необходимо дальнейшее совершенствование среды прежде всего в когнитивно-эргономическом отношении (удобства работы и когнитивного качества создаваемых описаний). Среда постоянно совершенствуется разработчиком; пока при этом также изменяется формат файлов её документов без обратной совместимости, что осложняет разработку приложений, сопрягаемых с ней по данным. Обратная связь с разработчиком доступна в этой теме.

Также Я.Романченко создан автономный дракон-конвертор ДРОН для языка Активный Оберон (диалект языка Оберон – преемника Паскаля Н. Вирта). ДРОН использует дракон-модели, созданные в среде Тышова. Доступна версия, работающая лишь с одним из прежних форматов файлов Ты-среды; поэтому ДРОН привязан к одной из старых её версий (см. Полезные ресурсы в Разд.6). Обратная связь с разработчиком доступна в этой теме.

Каковы же общие рекомендации по выбору средств на сегодняшний момент? Если цель — создать целостную формальную модель, пригодную для программной реализации, то нужно пользоваться Ты-средой. Если же цель — создать модель в первую очередь для человека, следует либо использовать результаты моделирования в этой среде как основу диосцен, дорабатываемую в редакторах документов или изображений, напр. из состава пакета OpenOffice.org, либо сразу визуализировать в таких редакторах (если не предполагается дракон-программирование).

Отметим также перспективные проекты и целесообразные направления развития ИСП-ДРАКОН.

Разработчик систем управления реального времени Д.В. Барановский ищет возможности применить техноязык для сквозной формализации таких задач вплоть до гибридного программиро-

вания. На данный момент в этой теме он провёл сравнительный анализ языка и дракон-сред с другими методологиями, указал на некоторые проблемы реализации и предложил решения по смешанной текстово-графической ИСП. Отметим, что пока как язык реализации предложен не ДРАКОН, а язык минимальных блок-схем, что не позволяет говорить о полноценном решении.
В г. Орле ведётся разработка дракон-редактора, воплощающего идеи И.А. Ермакова по совершенствованию шампур-метода. Некоторые сведения доступны в этой теме.
Возможности визуализации бизнес-процессов на драконоподобном импер-языке введены в популярную среду автоматизации 1С:, о чём подробнее в этой теме (при разработке языка использовались консультации В.Д. Паронджанова).
В целом развитие специализированных сред будет зависеть и от развития языковых средств формализации знаний. Подробнее авторский взгляд на этот вопрос раскрыт в п/р 5.3. Языки представления знаний определяются в Приложении 2 к основному тексту ресурса.

> Тем самым реализация ТФЗ должна стать комплексной ИСП визуализации знаний; следуя Барановскому, удобно назвать её системой РДП – разработки и документирования процессов (и программ как предельно формализованной части этих процессов). ИСП без выхода на программирование (автоматическое) также будем называть РДП-средой.

Потребность в совершенствовании Ты-среды привела ещё одного разработчика — В.А. Тарасенко — к рефакторингу её приложения. Одновременно Тарасенко сформулировал требования как к среде, так и к стандарту техноязыка; обсуждение их с участием автора этих строк привело к выработке уточнённых требований. Результаты доступны в этой теме. Фактически это наиболее целостные требования к практически пригодной ИСП визуализации знаний. Поскольку необходима их отработка на конкретных ситуациях, в этой теме рассматриваются некоторые примеры.

5.2. Культура, информатика и техноязык

5.2.1. Информатика и формализация

Довольно часто в документе говорится об «информатическом». Смысл этого понятия, надо полагать, вполне ясен из описания формализации в п. 1.4.2.

Также следует точнее определить содержание информатики как научной дисциплины. Сегодня уже очевидно, что неоправданно сводить её предмет к компьютерному делу (часто используется термин «компьютерные вычисления», являющийся калькой с англ. computer science), хотя у нас на это указывалось ещё около 20 лет назад. Здесь мы дадим определение, следуя подходу В.К. Белошапки, Ф. Перегудова, В.А. Пароджанова, А.Я. Фридланда – не рассматривать информатику как «знание о компьютерах» и соответственно не считать, что она возникла вместе с машинами Тьюринга и фон Неймана. Вместе с тем, не следует чрезмерно расширять её предмет; так, если рассматривать информатику как науку об информации в целом в смысле определения Фридланда (приведённого в п. 1.1.2 Приложения 1), то получится, что «информатика – это наука наук, и остальные науки, такие как физика, химия, биология и др., являются её детализацией. Или же информатика – это наука о мышлении.»/4, с.192/. Сам Фридланд далее определил информатику как науку о формализации любых задач, но в которой основную роль играют информашины (см. /4, с.194/). В то же время обоснование этого определения он начал утверждением, что информатика занимается формальной обработкой данных. Если вспомнить его же определение, что данные – это составляющая информации, отчуждённая от аппарата мышления, существующая как воспринимаемые и порождаемые этим аппаратом данные (знаки, сигналы), то можно дать иное определение:

> *Информатика* – наука о данных (в т.ч. как о представлении знаний, отчуждённых от их носителей), формах и условиях их существования, методах и средствах переработки, процессах формирования, сохранения и использования, а также о явлениях, имеющих ме-

сто в связи с этим.

В соответствии с этим информатика определенным образом связана не только с машинной переработкой данных; также к её предмету относится и реализация информатических процессов без участия информашин.

Информатика не подменяет собой математику и не входит в её состав; «если математика – это язык наук, то информатика -это инструмент наук» /4, с.193/ (и математики в том числе); однако инструмент, конечно, влияет на предметную область его использования. В то же время это и не наука о языке в широком смысле (филология, лингвистика); однако она очень тесно связана с нею в том смысле, что использует любые языки и порождает формальные (языки представления знаний). Более того, информатическое образование д.б. языковым в своей основе; сейчас это имеет место только для ИТ-специалистов. Пожалуй, именно так мы сможем прийти к моделированию как второй грамотности, к чему призывает, в частности, Паронджанов; он же в /1, Гл.1...3/ показал связи информатики с науками о человеке в когнитивно-эргономическом аспекте. Есть и иная связь этих дисциплин – в аспекте собственно формализации человеческой деятельности. В рамках теоретической информатики этим занимается направление искусственного интеллекта, но в самой общей постановке; более конкретным является направление [авто-]формализации знаний.

Информатику в широком смысле, следуя за Белошапкой[16], можно понимать как науку о формальном анализе произвольной предметной области, а её основной метод – как поиск инвариантов этой области; подразумевается, что предметная область в итоге описывается конечными моделями, т.е. аппарат информатики – это «математика без бесконечности» (точнее, методы сведения бесконечного к конечному, раскрытия неопределённостей). Отношение информатики к другим дисциплинам было системно показано Перегудовым[17].

В ходе анализа также возможны преобразования содержания дисциплины: реструктуризация, «исправление имён», коррекция языка и формы изложения; все они имеют цель согласовать процессы интерпретации знания в разных его отраслях, облегчить междисциплинарные взаимодействия и в конечном счете – повысить эффективность информационных процессов.

Итак, к предмету информатики, т.о., относится и анализ и синтез систем символического описания любых предметных областей, таких как стандарты ЕСКД, УСОРД и пр. Техническая отрасль информатики в числе прочего формализует и задачи построения средств переработки определённого назначения (широкого или узкого); однако собственно создание этих средств (информатической техники), очевидно, не относится целиком к её предмету; это область многих дисциплин; а вот организацию этих средств в систему, реализующую определённые информатические процессы, мы можем отнести к прикладной информатике.

Информатика занимается процессами ПрД и интеллектуальными процессами, её касающимися (т.е. там, где либо порождаются данные из знаний, либо наоборот); при этом она имеет дело с абстрактными величинами и данными. Содержательной интерпретацией этих данных занимаются другие науки (физических – физика, биологических – биология и т.д.); информатика занимается интерпретацией собственно информатических процессов.

По отношению к любой предметной области (в т.ч. и другой науке) информатика – это инструмент, позволяющий провести общий формальный анализ, выделить поддающиеся формализации задачи и довести их до реализации в виде искусственных информатических систем. Сегодня эти системы функционируют с помощью (или под управлением) искусственного устройства-исполнителя (информатической машины), но так было не всегда. Когда-то считали на счётах, а хранили данные на бумаге (камне, узловатой верёвке и т.д.); тем не менее и тогда существовала информатика, правда, скорее эмпирическая. Примером может служить работа Л. Пачоли о бухгалтерском учёте «Трактат о счетах и записях» (1494); в ней он по сути, стандартизовал информатическую технологию отражения движения ценностей в организованной деятельности. В теории и практике безопасности, кстати, давно уже выделяют решения (механизмы реализации) технические – использующие для достижения целей машины и организацион-

[16] Белошапка В. Мир как информационная структура. // Информатика и образование. – 1988. – №5.

[17] Перегудов Ф. Системная деятельность и образование. // Информатика и образование. – 1990. – №1.

ные – использующие только возможности человека и предоставленных ему нормативных документов, т.е. системы «человек-знание» (точнее, «человек-данные»).

Формальный анализ знания приводит к понятию м е т а т е о р и и (букв. «теории о теориях») как общей формы научного знания; здесь информатика смыкается с философией. Можно попробовать и определить информатику с философской т. зр., допустим, так:

> *Информатика* – это наука об отражении действительности, существующем вне сознания (в т.ч. отчуждённом от аппарата мышления интеллектуальной системы), формах и условиях существования такого отражения, методах и средствах (механизмах реализации) его целенаправленного порождения в виде искусственных систем (в частности – на основе машин для переработки данных), процессах его переноса, организации и интерпретации, а также об имеющих место при этом явлениях.

Здесь предмет можно определить и иначе, рассматривая только целенаправленно порождённое отражение (а не любое отчуждённое); но автор сомневается в необходимости такой конкретизации.

В философии же определяется содержание понятий 'отражение', 'действительность', сущность связи между ними, а также раскрывается сущность и соотношение материального и идеального, что используется другими науками (в информатике – в виде понятий сущности-объекта и сущности-модели, физического и информатического процессов и т.п.). В настоящее время дискуссионен вопрос, считать ли информацию категорией действительности; в то же время философское рассмотрение понятия информации уместно и плодотворно, т.к. даёт подходы к формальному его определению и даже к построению моделей информации. Сегодня известны соотвествующие результаты Герасименко и Н.В. Макаровой.

Разумеется, сказанное не претендует на истину в последней инстанции.

5.2.2. Техноязык и культура

Вначале кратко о состоянии применения техноязыка.

Создана система программирования на основе базовой концепции дракон-системы (см. /1, гл. 14/). Система фактически является САПР типа I-CASE. Язык ДРАКОН в свое время был рекомендован для изучения в вузах[18]. Автор языка также подготовил пособие для начального образования, вышедшее очередным изданием /2/.

Имеются сведения о практическом использовании методологии ДРАКОН профессиональными разработчиками ИС для постановки и алгоритмизации задач (примеры см. /1, гл. 13/, а также интернет-ссылки, в т.ч. в Разд.6). Методология ДРАКОН может использоваться как основа современной технологии проектирования сложных АИС, таких, как разработанная С.Д. Паронджановым.

Построенная ДРАКОН-система I-CASE послужила основой CASE-технологии РПО “ГРАФИТ-ФЛОКС”, которая применялась, например, в ряде космических проектов (см. /1, с.27). Немаршрутные подъязыки в этой системе специфические, «заточенные» под задачи управления космическими аппаратами.

Разработчики новых ИСП одновременно решают и вопросы о стандарте языка, о новом представлении известных задач с его помощью; это можно видеть из материалов, представленных в п. 5.1.3.

Каковы реальные перспективы техноязыка как элемента культуры? Здесь нужно вспомнить о том, как культура структурирована. Так, российский мыслитель и художник XX века Н.К. Рерих (на базе познания индийских и др. восточных доктрин мировоззрения) определял её как триединство веры, науки/техники и искусства в любых человеческих проявлениях. Структуру по Рериху можно наложить на «большую тройку» нового системного подхода: "рацио"(понятно, что наука)-"эмоцио"(искусство)-"интуицио"(вера).

Как на этой основе определить информатическую субкультуру? Понятно, сколь много здесь науки и техники; о специфике их достаточно сказал Ф. Перегудов в вышеупомянутой статье. Информатизация связана, как мы говорили, со сведением бесконечного и вариативного к конечному и однозначному как следствием «материализации» математических представле-

[18] Примерная программа дисциплины "Информатика". Издание официальное. – М.: Госкомвуз, 1996.

ний о конкретной задаче, о чём хорошо сказано И. Ермаковым в докладе, вложенном в это сообщение конференции OberonCore.ru. Здесь требуется и искусство — недаром свод конкретных информатических знаний, труд формирования которого в текстовой форме взял на себя Д. Кнут, так и называется «Искусство программирования». А где же вера?

Не новички в информатике могут вспомнить представление времён, «когда компьютеры были большими», о программировании как «жречестве» - но сегодня среди ИТ-профессионалов распространена прямо противоположная т. зр., что можно увидеть, допустим, в проекте Информатика-21 (в виде исходной посылки, выделенной на стартовой странице его веб-ресурса жирным шрифтом). Бытовал и культ «пиджинфор-инглиш» в информатике (когда убеждали, как «нетрудно выучить три десятка английских слов», используемых как ключевые в прогязыках, чтобы «быть международно понятым») - но у нас это окончательно развеял как раз один из выдающихся информатиков мира А.П. Ершов, предложив русскую алгонотацию для общеобразовательного курса информатики (а ещё раньше употреблялся язык Аналитик для первых «персоналок» семейства МИР). Техноязык же вообще заменяет ключевые маршрутные слова графикой.

Ну а есть ли какие-то положения, которые можно считать непреходящими «догматами веры» в информатике? Для понимания этого опять обратимся к основам информатики в изложении Фридланда. Конкретно, в /4, п. 10.6/ обсуждается алгоритмическая разрешимость задач — т.е. возможность поручить их решение формальному исполнителю (у Фридланда - «устройству», у Ершова - «материальному средству»). В связи с этим приводятся примеры алгоритмической неразрешимости задач, показывающие необходимость доказывать разрешимость для конкретных задач или их классов. Далее указывается:

«На практике, в связи с невозможностью полностью формальных постановок задач, всегда есть возможность так варьировать начальные условия, так менять постановку задачи, что алгоритм всегда находится, но вопрос об адекватности этих моделей, доведённых до численного решения, реальным проблемам всегда зависит от уровня компетенции разработчика и заказчика, так называемых конечных пользователей.»

Отсюда вытекает следующее. Не имея возможности доказать в общем, что для любой поставленной задачи можно найти алгоритм её решения, мы должны верить, что если задача практически важна — то наши интеллектуальные усилия приведут к построению алгоритма хотя бы в изменённой постановке — так, что в исполняющей решение оргтехсистеме «человек-устройство» неалгоритмизуемая часть (оставляемая за человеком) будет соответствовать возможностям его интеллекта, тогда как в чисто организационной системе (где решение исполняет только человек/коллектив) нагрузка на человека могла быть чрезмерной. Вот и первый «догмат». Из той же цитаты вытекает и ещё один — достаточно произвольно меняя постановку задачи, разработчик решения иногда лишь интуитивно определяет адекватность её реальной проблеме — научное обоснование может возникнуть лишь впоследствии, в т.ч. по итогам решения.

Но «вера без дел мертва есть» - и чтобы найти нужные решения, порой требуется мобилизация всех возможностей человека. И здесь важны средства, улучшающие работу ума. Обсудим взвешенно, какие свойства техноязыка позволяют воспринимать его в этом качестве.

Раз уж мы придерживаемся «классического» подхода к информатике как дисциплине, рассмотрим техноязык (и вообще языки представления знаний) в сопоставлении с естественными человеческими языками и выделим существенные черты развития ЯПЗ.

В языкознании человеческие языки делят на несколько родов (строёв) по организации; ею обусловлены различные возможности передачи смысла между носителями языка[19]. Другое измерение классификации – передают ли письменные знаки звуки устной речи или понятия.

В настоящее время сохраняется принцип абсолютизации текста. Можно видеть это, напр. в цитате из следующего сообщения: «Преимущество текстов в плане технологичности, масштабируе-

[19] Вкратце это описано в Девятов, Мартиросян, с. 345-347.

мости и универсальности перед графическими системами коммуникации -- как я уже когда-то тут говорил -- давно доказано практикой человечества.».

У Фридланда мы видим противоположный подход: «Действительность поступает в аппарат мышления с помощью чувств (чувственное восприятие). Большая часть сообщений в настоящее время поступает в виде текстов. Следует обратить внимание, что физиологически человек (глаза) не приспособлен к чтению - длительному рассматриванию мелких объектов (букв, слов). Текст - это искусственное построение. Тысячелетия человек занимался охотой и смотрел в основном в даль. Развитие телевидения и компьютеризация приводят к тому, что человек уходит от неестественного процесса чтения к естественному - образному видению, правда, смена событий должна быть медленной, а не мелькающей, как в клипах (хорошо это или нет - это другой вопрос, здесь не рассматриваемый). Следствием этого является уменьшение числа людей, читающих книги, и увеличение числа смотрящих на экран телевизора или компьютера.».[20]

Однако некритически следуя второму подходу, мы можем прийти к противоположной крайности - «принципу абсолютизации картинки».

> Техноязык, сочетая графику с текстом, исключает крайности абсолютизации той или другой формы представления данных (в т.ч. отчуждённых знаний). Более того, любую граф-схему следует рассматривать как текст — но шампур-метод для графов, которые мы в п. 2.1.2 назвали маршрутными, задаёт строгий двумерный порядок чтения, облегчающий восприятие смысла, отчуждённого сочинителем.

Разрабатывая ДРАКОН, создатель техноязыка одновременно создавал методологию формализации знаний. Из неё он вывел также нормативную теорию интенсификации интеллектуальной деятельности, кратко изложенную в /1, гл. 20/. Общий вывод теории — построение искусственного языка влияет на качество мышления субъекта, который им пользуется.

Может возникнуть вопрос: не слишком ли это абстрактно? Уже нет — взять хотя бы работы по теории изобретательства. Один из последователей отечественной ТРИЗ М.А. Орлов в своей работе рассматривает и механизмы мышления, включая влияние форм представления задач, объектов на эффективность решения. По сути, это подход к интенсификации интеллекта под иным углом зрения.

> Итак, если мы хотим решать серьёзные задачи — нужно формализовать их на когнитивно-эргономичных языках — это само по себе приблизит нас к результату.

Как может бытовать техноязык в культуре? Довольно сложно м.б. представить это для человека европейской, «литерной» культуры — об этом косвенно говорят и приведённые цитаты. Стоит обратиться к опыту цивилизаций «драконоподобной» культуры — а конкретно идеографического естественного языка. Так, в китайском языке иероглиф обозначает не звук, а понятие, как и в техноязыке. Смысл м.б. многозначен — можно это увидеть хотя бы по нижеприведённой цитате из словаря:

дà 1) большой; великий; высокий (*ростом*); крупный, огромный; вырасти; 2) старший (*по возрасту, положению*); быть старше; 3) сильно, очень, весьма; громко; особенно; преувеличенно; 4) увеличивать, расширять; преувеличивать; напускать на себя важность; 5) *сокр.* высшее учебное заведение, университет...

из китайско-русского словаря

Многозначность китайского иероглифа

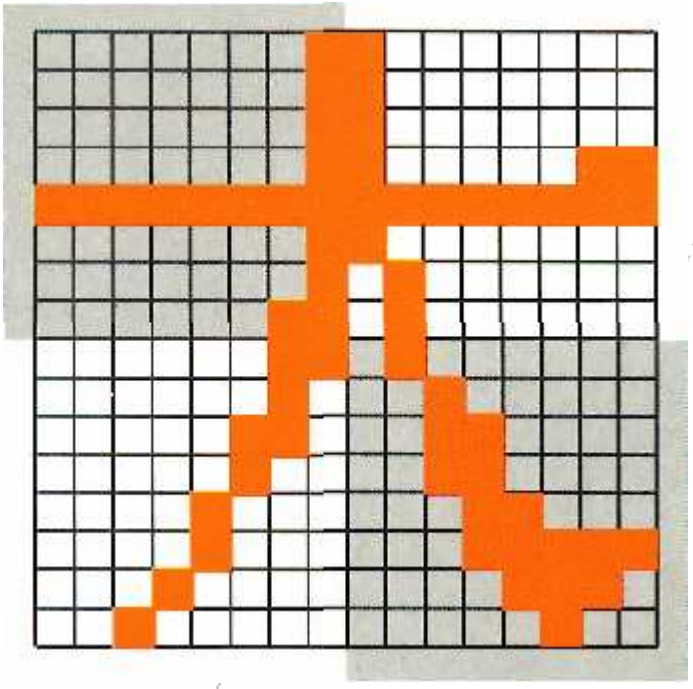
По сути, подобным образом трактуются и иконы техноязыка в п. 2.1.3.

В устном языке возможно сходство произношения разных иероглифов, что влечёт зависимость смысла от чтения — об этом хорошо сказано Девятовым, им же указано на путь

[20] Фридланд А.Я., 2005. - С. 47.

преодоления возникающей неопределённости применительно к естественному (китайскому) языку — введение нормативного письменного языка «вэньянь», общепринятого для носителей языка[21]. В формальном языке такие «вольности» неуместны — но всё же одна икона (макроикона) при «гибридизации» техноязыка с разными прогязыками может менять свой смысл.

В настоящее время можно сказать, что основное внимание уделяется именно отдельным иконам как алфавитным знакам. Решаются вопросы, которые можно опять-таки на примере рассмотренного иероглифа символически представить следующим рисунком:



Пример информатизованного представления китайского иероглифа

Т.е. выбирается представление знаков, оформление их текстовых и графических атрибутов. К сожалению, даже вопросы представления знаков техноязыка пока не решены разработчиками ИСП до практически требуемого уровня, что можно увидеть при работе с конкретными приложениями.

В то же время нужно заниматься и «правилами чтения» знаков в тексте различной степени формальности. Неизбежно при этом появятся «диалекты», согласование которых будет существенной частью бытования техноязыка. Пока на конечном уровне формализации – «исходных чертежей», т.е. визуальных командных моделей – разница смыслов либо игнорируется (в первую очередь за счёт моноязычия – гибридизации с одним прогязыком), либо идут по пути «вэньянь» – вводят единую «норму» для всех гибридных языков. При этом упускается из виду, что язык не естественный (когда правила чтения формируются в сознании носителя как результат общения), а формальный, и при унификации он неизбежно обрастает новыми отчуждёнными от носителей правилами построения текста, согласующими конструкциями, а значит – становится более громоздким. Но что более существенно – возникает неоднозначность прогтекста как результата трансляции исходных чертежей, а точнее – подмена прогязыка X «вэньянь-прогязыком». Она требует разрешения, часто вручную специалистом высокой квалификации – прогтекст-то требуется на языке X... :)

По мнению автора этих строк, следует раздельно гибридизировать техноязык с каждым выбранным прогязыком, не задумываясь о возможности образования «визуальных диалектов». При этом аналог «вэньянь» возникнет естественным образом как результат согласования диалектов – и будет, конечно, касаться не языка исходных чертежей программ, а более высокого уровня формализации (т.к. потребует интеллектуальных процессов для «диалектизации»).

[21] Девятов, Мартиросьян, с. 302-303.

Также нужно глубже вникать в природу информашины как формального исполнителя визуалов и учитывать в гибридных языках (включая маршрутную часть) особенности исполнения алгоритмов. До некоторой степени это возможно без учёта особенностей архитектур – но и проникновение этих особенностей в язык следует воспринимать естественно. В конце концов ЯПЗ предназначены как для практики, так и для исследований и образования – и такое проникновение только пойдёт им на пользу как универсальным инструментам.

В общем, мы приходим к тому, о чём говорилось вначале – в примере А.А. Мамедова, иллюстрировавшем второй уровень формализации знаний в п. 1.4.1 – что-то лучше передать на одном языке (диалекте), что-то на другом.

В то же время концепция «вэньянь» вполне приложима и на уровне командных моделей. Только на дело нужно посмотреть иначе —ввести прогязык не обобщающий, а сопрягающий, как «интерлингву» между разными Х-языками, унифицирующую промежуточное представление программ на них— и в программировании этот подход хорошо разработан[22]. По сути, таких языков реализовано уже несколько – и для дракон-моделей нужно создать подходящий. Понятно, что они предназначены программистам (а возможно, и только для машинного использования). По этому пути пошёл Барановский, предложив язык текстового формата алгоритмов и программ (ТФАП) – но над проблемой надо ещё работать.

Сегодня мы можем говорить лишь о «единичном производстве» визуалов как результатов труда независимо от реализации (среды поддержки). Очевидно, как и всякие искусственные системы, визуалы по мере вхождения техноязыка (на деле, как мы видим – семейства языков) в практику начнут производиться (сочиняться) сначала серийно, а затем и массово. Как это повлияет на ДРАКОН-МФЗ? Можно привести пример такого материального объекта, как... истребители ЯК в годы Великой Отечественной войны.

Конструктор этих машин в своих мемуарах приводит следующий отзыв о них[23]: «Вот ещё одна оценка ЯК-3, данная французами в майском номере журнала «Айрплен» за 1956 год... «ЯК-3, на мой взгляд <автора статьи, цитируемой Яковлевым>, является идеальной машиной для массового производства на предприятиях с неквалифицированной рабочей силой. Конструкция неправдоподобно груба, с качеством сварки, соответствующим уровню деревенского кузнеца, а в то же время внешняя отделка прекрасна и такая же, как у гоночных машин. Это машина, в которой отказались от ненужного украшательства, надёжная и построена с расчётом на жизнь в несколько часов, какой живёт истребитель в военное время. Она полностью отвечала требованиям к таким машинам, и при этом в ней не было ничего лишнего».»

Интересные сведения на эту тему можно извлечь и из книги другого конструктора оружия военной поры – В.Г. Грабина; они показались автору столь практически значимыми, что были опубликованы для обсуждения в этом сообщении и в этом сообщении. Последовавшее за тем развитие предмета в этой теме показало актуальность для ИТ-сферы несмотря на то, что описанные события происходили три четверти века тому назад... ;)

Что общего между оружием и моделями деятельности? Сразу очевидно, что массовое производство визуалов (в рамках той же СМК ИСО9000) точно также будет осуществляться людьми весьма различной и в общем невысокой информатической и математической культуры. Норма времени и сил на эту работу для них будет мала и безо всякой войны – в силу того, что это не их основная деятельность. Жизненный цикл такой продукции тоже м.б. невелик – в условиях реинжиниринга деятельность регулярно улучшается, а значит и модели обновляются. При этом «внешняя отделка» их м.б. понята как удовлетворение требований к когнитивной эргономичности – и ясно, что требования здесь не меньшие, чем для обеспечения аэродинамики :) А надёжность аналогично м.б. обеспечена простотой графовой конструкции и доказуемостью её правильности – какой бы «неправдоподобно грубой сваркой» (т.е. операциями вывода по шампур-методу) она не образовывалась. Наконец и организация производства

[22] См. напр.: Свердлов С.З., 2007. – С. 431-435.

[23] Яковлев А.С. Цель жизни: Записки авиаконструктора. – М.:Республика, 2000. – С. 296.

должна измениться в русле реализованного Грабиным – должны делаться (причём во всевозрастающем объёме) и модели деятельности и реализующие их методико-программно-аппаратные косавты для конкретных оргсистем, а не только средавты для «рабочих мест вообще», назначение которых умозрительно определено разработчиком, а конкретная методическая и техническая поддержка фактически возлагается на пользователей, обменивающихся опытом по преодолению «фичей и багов»... :)

Как могут видоизмениться свойства техноязыка, чтобы визуалы могли выходить «из рук каждого деревенского кузнеца» в нужном количестве, качестве и главное – своевременно? Прежде всего массовые визуалы будут в некотором смысле также «лишены украшательства», т.е. их структуры будут упрощёнными. Видимо, в первую очередь за счёт таких особенностей пользования техноязыком, как:

- ограничения произвольности нелинейных структур (пересадок и заземлений лиан);
- использования типовых визуалов и фрагментов;
- преимущественно доказательного построения (в частности, на базе цикла Дейкстры для нелинейных конструкций);
- формализации текста на базе шаблонов синтаксиса, зависящих как от типа иконы, так и от языка схемы.

Конечно, сохранится и «единичное производство» визуальных моделей, с нетиповыми структурными решениями, индивидуально проверяемыми на корректность.

Далее, технология визуализации как проектирования деятельности станет промышленной, с последовательным уточнением проекта от эскизного к рабочему. В обеспечение этого возможны следующие направления развития:

- введение в язык вершин-заместителей для построения эскизов;
- использование областей-вариантов для проработки решений схемы в целом и/или по частям и выбора нужного;
- введение областной декомпозиции визуала со сменой уровня формальности его языка;
- ведение приложений икон в сокращённом составе – имеет смысл пользоваться только управленческими, которые будут служить пояснительными (а в каких-то ситуациях – и объяснительными ;)) записками сочинителя, обосновывающими его решения по визуализации данной иконы (а для высших уровней – и по декомпозиции).

Эскизы и варианты визуалов (и других классов схем) по сути реализуют давно известную концепцию FEO (моделей «для сведения»). Очевидно, они наиболее широко применимы для учебно-научных построений.

Состав приложений икон, реализованный в Ты-среде, также замещается названными языковыми средствами; приложения алгоритмические заменяются результатами областной декомпозиции и вариантами, а программные и трансляции – корректной реализацией трансляции дракон-программы в прогтекст.

Наконец, комплексное знание о задаче (частью которого является дракон-модель) нужно целостно и эргономично представить. Для этого создатель техноязыка и предложил схематизацию знаний. Следуя этому, нужно реализовать РДП-документ как схему, в которую вписывается основное содержание (и текст, и таблицы, и графика). Иначе говоря, объект содержания сопоставляется не напрямую диосцене (листу документа), а базовой схеме, которая служит непременной составляющей листа. При этом схематизированное содержание (те же дракон-модели) должно сохранять свои внутренние связи. Т.е. реализуется тот же принцип, что в динамической («электронной») таблице – вычисления заданы формулами, которые сочинитель располагает произвольно по ячейкам одного листа, разных листов одного и/или разных документов-«книг» – но координаты ячеек служат идентификаторами величин в формулах, и тем самым отражается структура вычислений «поверх» структуры документов.

В офисных пакетах внутренние связи обычно называют «зависимостями» и отображают временно по команде оператора. В нашем случае также следует ввести способ и форму отображения структур моделей в РДП-документе.

Важен и такой вопрос: а для чего все эти усилия, иначе говоря – каково предназначение методологии визуализации знаний? Представляется, что это формализация задач созидания реальных ценностей (и практических, и научных, и образовательных – если вспомнить классификацию информационных процессов по Фридланду). Соответственно нужно «затачивать» визуальную МФЗ – языки, технологию, реализацию – под процессы реального производства, включая проектирование косавтов именно для этих процессов, причём конкретных, привязанных к реальной оргсистеме (предприятию, учреждению). Справедливо замечено, что «реальных товаров для населения от новых модных и престижных версий програмных продуктов практически не прибавляется»[24] – и здесь имеется в виду как раз продукт «абстрактный», привязываемый силами пользователей. Причём речь должна идти, видимо, прежде всего о гражданском секторе – военное применение работающей «на гражданке» методологии можно будет обеспечить быстро, а эффективность т.н. «высокотехнологичных» систем оружия и управления войной, типа американской СОИ/НПРО, сомнительна без волевых и умелых бойцов и военачальников – автор согласен с тем, что «победа-то всегда в конечном счёте определяется человеческими качествами, а именно: волей командиров, воодушевляющей войско на подвиг и стойкость»[25]. А люди, умелые в визуализации, интенсифицировавшие свой интеллект, приходят в армию (в т.ч. военные вузы) и к созданию техники и технологии для неё не откуда-нибудь, а из школы и гражданского вуза – где нужно это должным образом преподавать...

Ну и напоследок, раз уж мы здесь затронули организационно-экономическую субкультуру – некоторые соображения об этой стороне внедрения визуализации. Модным ныне способом реализации нового дела вообще и коммерциализации научных разработок в частности являются инвестпроекты. Однако стоит внимательно присмотреться к сущности этого понятия. Так, Девятов пишет, сопоставляя опыт российского, западного и китайского хозяйствования: «Вера – иррациональна, а потому бесстрашна, что даёт громадные преимущества хозяину в предприимчивости. ...главное при подходе к делу не пресловутые *инвестиции*,... но *предоплата* за ресурс, оставляющая хозяина ресурса независимым «хозяином положения» в своём деле.»[26] и далее поясняет свою т. зр.: «Смысл имени «инвестиции» в том, что богатеет не тот, кому дают, а тот, кто сначала даёт, а потом потребляет...»[27]; «...если китайский капитал зайдёт в Россию в форме инвестиций, то произойдёт то же, что Россия имеет в отношениях с Западом. Инвестиции примут форму кредита, валютная выручка от поставки природных ресурсов будет покрывать лишь проценты по обслуживанию долга, а возврат инвестированных капиталов придётся делать за счёт ущемления своих суверенных прав на ресурсы или чего-нибудь ещё более непотребного.»[28] и наконец даёт практическую бизнес-рекомендацию: «...вычеркнуть слово «инвестиции» из переговорного процесса... и заменить его всюду на слово «предоплата».»[29]. Конечно, это примеры макроэкономического уровня – но очевидно, что те же явления воспроизводятся и на микроэкономическом. Так что тем, кто верит в практичность как визуальной методологии, так и собственноручно разработанных средств её поддержки, стоит подумать о сказанном (и почитать написанное около этих цитат – там ещё много интересного).

В этом свете важной представляется мысль в упомянутом докладе И. Ермакова о нетенденциозном выборе языковых средств разработки АИС и уверенном следовании этому выбору независимо от позиции заказчиков. И здесь, кстати, существенным подспорьем м.б. организация

[24] Девятов А., Мартиросьян М. – С. 293.
[25] Девятов А., Мартиросьян М. – С. 292.
[26] Девятов А., Мартиросьян М. – С. 242.
[27] Девятов А., Мартиросьян М. – С. 268.
[28] Девятов А., Мартиросьян М. – С. 270.
[29] Девятов А., Мартиросьян М. – С. 270.

работ по внедрению как «частной интеллектуальной инициативы» владельца интелресов визуальной формализации знаний. Типичный «спецпропагандистский» приём «любителей инвестировать в то, что плохо лежит» из ряда: «а за что это я должен вносить предоплату, если чиста канкретна мне нужного продукта ещё нет?» :) вполне преодолевается положительными результатами такой инициативы – признанием и распространением некоммерческого продукта.

Таковы практические соображения по возможному развитию техноязыка. Общий вывод – движущей силой включения его в культуру будут реальные задачи, в какой-то степени влияющие на стандарт языка.

5.3. ДРАКОН в структуре средств системно-информационного метаязыка

5.3.1. ДРАКОН и некоторые вопросы формализации

Как показано в /1, гл.17/, язык ДРАКОН с математической точки зрения является формальной системой, относящейся к классу исчислений (автор называет его «исчислением икон»). Элементы алфавита техноязыка являются аксиомами этого исчисления, а каждая дракон-схема – визуальной теоремой; правила визуального синтаксиса и правила вывода теорем сформулированы в шампур-методе. Данная система включает как часть алгебру логики, поэтому возможно графическое представление булевых функций дракон-схемами, что показано в /1, гл.9/.

Формальность ДРАКОНа позволяет однозначно определить порядок трансляции дракон-схем в программы для вычислительных машин (на АЯВУ или в машинных кодах), т.е. фактически реализовать ***визуальное программирование***. При этом устраняются неопределенности, характерные для традиционного структурного программирования на текстовых языках, а первоначальную модель (спецификацию задачи) может составить непрограммист. Такая спецификация может рассматриваться как постановка задачи; далее автор самостоятельно или с помощью специалиста уточняет дракон-схему, переходя к более формальному командному языку и детализируя алгоритм. Так осуществляется автоформализация знаний.

Хотя на первый взгляд ДРАКОН отвергает почти всю сложившуюся практику алгоритмизации и программирования, на самом деле он является продуктом творческого развития идей структурного программирования Э.Дейкстры и особенно теории схем программ А.П. Ершова, что показано в /1, гл.16-17/. Оригинальная составляющая заключена в рассмотренных ключевых идеях языка и средствах их реализации, которые автор языка назвал "шампур-методом". При этом объектом метода является абстрактная дракон-схема (в которой отсутствует текстовая часть), также называемая *шампур-схемой*. Язык метода (маршрутный, или *шампур-язык*) с теоретической точки зрения относится к т.н. языкам крупноблочных схем. Шампур-метод фактически определяет визуальный синтаксис этого языка.

На практике ДРАКОН выступает как родоначальник семейства техноязыков; его стандартом задан визуальный синтаксис маршрутного подъязыка. При дополнении шампур-языка текстовым синтаксисом икон получается конкретный язык семейства. В частности, использование синтаксиса того или иного существующего языка программирования приводит к т.н. *гибридному* языку программирования ДРАКОН-Х (здесь Х-имя использованного языка). Такой прием Паронджанов называет *гибридным программированием*; визуал с формальным текстом именуется *дракон-программой*.

Примеры перехода к программам на операторных АЯВУ автор языка привел в /1, гл.12/. Построение визуала на языке ДРАКОН-Х из макроструктурных (структурных, лианных, адресных) блоков фактически эквивалентно структурному программированию на использованном языке Х. При этом иконы "Адрес", употребляемые по правилам построения силуэта, задают безусловные переходы однозначно в отличие от аналогичных операторов в текстовых языках (goto и его заменителей).

Возможно доведение визуального программирования до уровня Ассемблера (т.е. получение машинных кодов непосредственно по дракон-схеме). Все это открывает путь к глубокому и эффективному управлению качеством алгоритмов и программ как компонентов обеспечения оргсистемы (предприятия, учреждения).

5.3.2. О других средствах описания деятельности

Сегодня деятельность оргсистем принято описывать на основе методологий моделирования SADT (Structured Analyses and Design Technique – техника структурного анализа и проектирования) и им подобных, как методология функционального моделирования IDEF0. Мы можем принять IDEF0 в качестве средства обобщённой формализации знаний.

Применением алгоритмов-вставок в ДРАКОНе фактически достигается декомпозиция (иерархическое структурирование) описываемой деятельности, аналогичное методологии IDEF0. В то же время такая методология требует в конце концов перехода к алгоритмическому языку для алгоритмизации задачи и программирования. Тем самым расширяется не только множество языков модели, но и вводится новая форма ее представления; меняется и точка зрения на проблему. Понимание такого описания требует дополнительных интеллектуальных усилий. Поэтому следует вводить прямое соответствие между функциями (процессами) в ФМ-языке и техноязыке. Естественным является соответствие «одна функция – один визуал»; тогда дракон-схема в терминах IDEF0 является спецификацией процесса выполнения функции, получая объекты видов Input и Control явно как формальные параметры, а отдавая объекты вида Output неявно, если только они не определены как компоненты результата визуала-функции. Можно ввести также соответствие «одна функция – одна ветка силуэта»; здесь получение входных объектов также будет неявным, и отображение логики процесса (связей ФМ-блоков на связи веток) требует определённых усилий.

В случае применения ДРАКОНа в процессе формализации просто детализируется (и по необходимости уточняется) исходный визуал с повышением формальности текста икон вплоть до формул и команд. Форма представления проблемы остается неизменной (комплект дракон-схем), что снижает интеллектуальные затраты участников формализации вне зависимости от профессиональной подготовки; достаточно лишь хорошо знать используемую версию ДРАКОНа.

При моделировании процессов, которые оперируют малым числом простых сущностей (данных, предметов и т.д.), объекты можно описывать в самой дракон-схеме или как простейшее приложение к ней; однако если сущностей много и они связаны сложными отношениями, следует моделировать декларативную составляющую отдельно. Для этого нужна методология, которая удовлетворяет критериям когнитивного качества в своей сфере формализации; возможным вариантом является IDEF1X.

В то же время IDEF0 пригодна как высокоуровневая методология для обобщенного моделирования и императивной, и декларативной составляющих. Среди SADT-методологий имеются и другие, подходящие на эту роль; некоторые из них описаны, например, в /5/; в последнее время среди ИТ-специалистов популярен также язык объектного моделирования UML. К сожалению, во всех случаях предлагается при последующей детализации пользоваться набором разных языков, что снижает когнитивное качество. Однако обобщенная модель в исходной форме вполне понятна и может послужить "прародительницей" и модели деятельности по методологии ДРАКОН (комплекта дракон-схем), и модели объектов (например, комплекта диаграмм IDEF1X).

Необходимость в описании оргсистем, простом и достаточно формальном, отчетливо выявилась при внедрении систем менеджмента качества (СМК) оргсистем. Международные стандарты МОС в отношении СМК (серия 900Х) в современной версии (закрепленной в России группой ГОСТ Р ИСО900Х-2001) прямо предписывают описывать процессы деятельности (бизнес-процессы) с целью их понимания персоналом и непрерывного совершенствования, т.е. автоформализовать профессиональные знания. Рассмотрим, какие средства предлагается для этого использовать и насколько они удовлетворяют критериям когнитивного качества.

Для обобщенных моделей МОС рекомендует функциональное моделирование по методологии IDEF0; при этом, как мы отмечали выше, требуются языковые средства для строгого описания содержания IDEF0-блока.

Ещё один класс – символические сценарные языки. Они описывают процесс применения различных устройств человеком детерминированно, как командную модель. Формировались в середине XX в. в связи с усложнением техники, повышенными требованиями к работе оператора. Сценарий задаётся как маршрут манипуляций органами управления, выбираемый в

зависимости от показаний органов индикации и иных доступных восприятию признаков работы устройства; чаще всего сценарии воспроизводятся непосредственно на пульте управления устройством. Используют специально разработанные алфавиты обозначения маршрутных переходов, состояний органов индикации, выдержек времени между манипуляциями и т.д.

> Важно понимать, что сценарии такого рода в пределе не что иное, как алгоритмы для человека («педантические» командные модели). Поэтому они сами по себе в какой бы то ни было совокупности не могут покрывать всё содержание деятельности; практически это выражается в том, что либо в сценарии, либо в охватывающем его описании (обычно — в руководстве оператора устройства) определяется развилка (или ряд на различных маршрутах) вида «в случае неуспеха сценария — действовать по обстановке»; тем самым в сознании оператора запускается интеллектуальный процесс поиска «незасценаренного» решения.

Другой класс – языки операторных (функциональных) схем, представляющие деятельность как процесс прохождения значений входных данных (дискретных или аналоговых) через систему трактов переработки в выходные данные. При этом элементы трактов заданы через математические соотношения «вход-выход», т.н. передаточные функции. Так удобно описывать «жёсткую логику» реализации деятельности (чисто аппаратным путём), поэтому такие языки часто используются при описании управляющего оборудования. Однако те или иные функции м.б. реализованы и программно; в этом случае алгоритмы их вычисления, конечно, оптимально записывать на техноязыке.

Разновидностью данного класса являются языки схем действий. Описание на таком ЯПЗ представляет собой, по сути, визуализацию процесса вычисления выражений. Поэтому он удобен при описании вычислительных задач и подзадач. В таких средствах автоматизации, как процессоры динамических (распространяющих, жарг. «машинных») таблиц, командная модель решения задачи представляется сразу в развёрнутом виде; каждый путь получения результата записан в ячейках динатаба от начала до конца. На такую запись адекватно отображается именно схема действий; поэтому на заключительных этапах информоделирования табличных вычислений она удобнее дракон-схемы (если не используется программирование для динатаб-процессора).

5.3.3. О других компонентах системного моделирования

Итак, для обобщённого описания системы, решающей произвольную задачу любой мыслимой предметной области, предлагается функциональное моделирование.

> Возникает вопрос: а почему бы не применить ДРАКОН уже для обобщённого описания решаемой задачи? Здесь мы рискуем последовать постулату Л.А. Заде: "имеющему в руках молоток везде видится гвоздь" :) иначе говоря, встать в зависимость от средства решения задачи. Конкретно, ДРАКОН является импер-языком и потому не показывает деклар-компоненту задачи. В принципе можно использовать систему из ДРАКОНа и когнитивно-эргономичного деклар-языка, но последний ещё предстоит создать; описание же, созданное такими средствами, всё равно окажется логически разделённым, что нарушает целостность исходной точки зрения на задачу – нужны умственные усилия для воссоединения компонент. В то же время IDEF0-подобный язык обеспечивает эту целостность.

Следует взглянуть и на общее содержание объёма модели; здесь следует привлечь некую философскую концепцию. Если мы стоим на позициях материалистической диалектики, то следует считать, что описание должно раскрывать аспекты пространства, времени и движения предмета изучения на заданном направлении. Такое раскрытие для импер-компоненты частного знания автор по возможности предпринял в п. 1.4.2; специалист в области философии и общей логики, возможно, найдёт в этом описании те или иные неточности.

На системные описания можно взглянуть и в иной плоскости – считать ли при описании более важными сущности («вещи») системы, или связи (логические и физические отношения) между сущностями. Методологии типа «сущность-связь» (англ. «Entity-Relationship», напр. визуальная МФЗ стандарта ERD) как раз совмещают эти точки зрения при обобщённом описании

и проясняют структуру логических отношений моделируемой предметной области для конкретного описания её через информатически строгую структуру данных.

> Информатическую строгость мы понимаем в смысле иерархии уровней моделирования и формализации, введённой в п. 1.4.2: всё существенное для целей моделирования представлено явным и конечным описанием. Как понимается информатика в данном документе, см. в этом пункте.

Ещё одна плоскость рассмотрения систем – подход к определению семантики (смысла) при формализации. Были выделены три подхода на математической стадии (при специфицировании решений задач для последующего программирования): денотационный, операционный, аксиоматический. Более подробно мы на них останавливаться не будем[30]; отметим лишь, что подходы по современным представлениям образуют системную триаду, поэтому закономерны взаимосвязи и взаимопереходы между ними.

Обобщённое знание может представляться с т.зр. сущностей-объектов (конструкторское описание любой системы, напр. как изделия по ЕСКД, где функционирование объясняется через принципиальные схемы объекта с указанием параметров процесса в тех или иных точках) и с т. зр. сущностей-процессов (напр. функциональное по IDEF0, где уже описание сущностей-узлов и их связь раскрыта через преобразования сущностей-ресурсов от входов блоков-процессов к их выходам); примером скорее аксиоматического подхода к описанию искусственных систем можно считать, по-видимому, теорию решения изобретательских задач (ТРИЗ) Г.С. Альтшуллера, где даются исходные структуры и элементы возможных искусственных систем (как т.н. веполи, т.е. комплексы «ВЕщество-ПОЛе») и правила «вывода», т.е. синтеза систем, решающих конкретную задачу[31]; аксиоматически задана структура системы и некоторые другие объекты.

> Конечно, возможности ТРИЗ шире; в принципе её можно рассматривать как первый практически реализованный подход к формализации знаний о механизмах произвольной предметной области (системах-решателях задач). Заданная в ТРИЗ сегодня структурная схема машины, по-видимому, должна относиться и к информашинам.
>
> Базовую процедуру вывода в ТРИЗ Альтшуллер и другие часто называют «алгоритмом изобретения», что являлось предметом критики (см. напр. /4, с.179/). Конечно, эта процедура скорее является «исчислением эффектов», т.е. относится к математической стадии формализации. Для частных случаев, ограниченных условий тем не менее возможно доведение её до информатически строгой, доказательством чему является возникновение т.н. «ТРИЗ-софта» – инфопрогизделий синтеза по правилам ТРИЗ; понятно, что ТРИЗ-софт является средством поддержки интеллектуальной деятельности грамотного специалиста в предметной области синтеза, но также при определённом его построении – и поддержки образования в этой области. При этом программа берёт на себя алгоритмически разрешимые компоненты решения изобретательской задачи, а человек делает всё остальное, применяя программу по неформальной технологии на базе собственного опыта.

Точно так же можно сказать, что определение ДРАКОНа явно тяготеет к аксиоматическому подходу.

Столь популярный ныне объектно-ориентированный поход, который пытаются приложить как «единственно верный» буквально ко всем предметным областям, а не только в информатике, как видим, на деле является лишь одной из возможных точек зрения. Критику его «идолизации» в информатике можно найти, напр., на форуме /8/.

До сих пор мы говорили об уровне обобщённой (предварительной) формализации знаний в целом, предшествующей уровню конкретной (детальной) формализации по частям. В качестве частей мы, следуя Паронджанову, предложили императивное и декларативное знание и

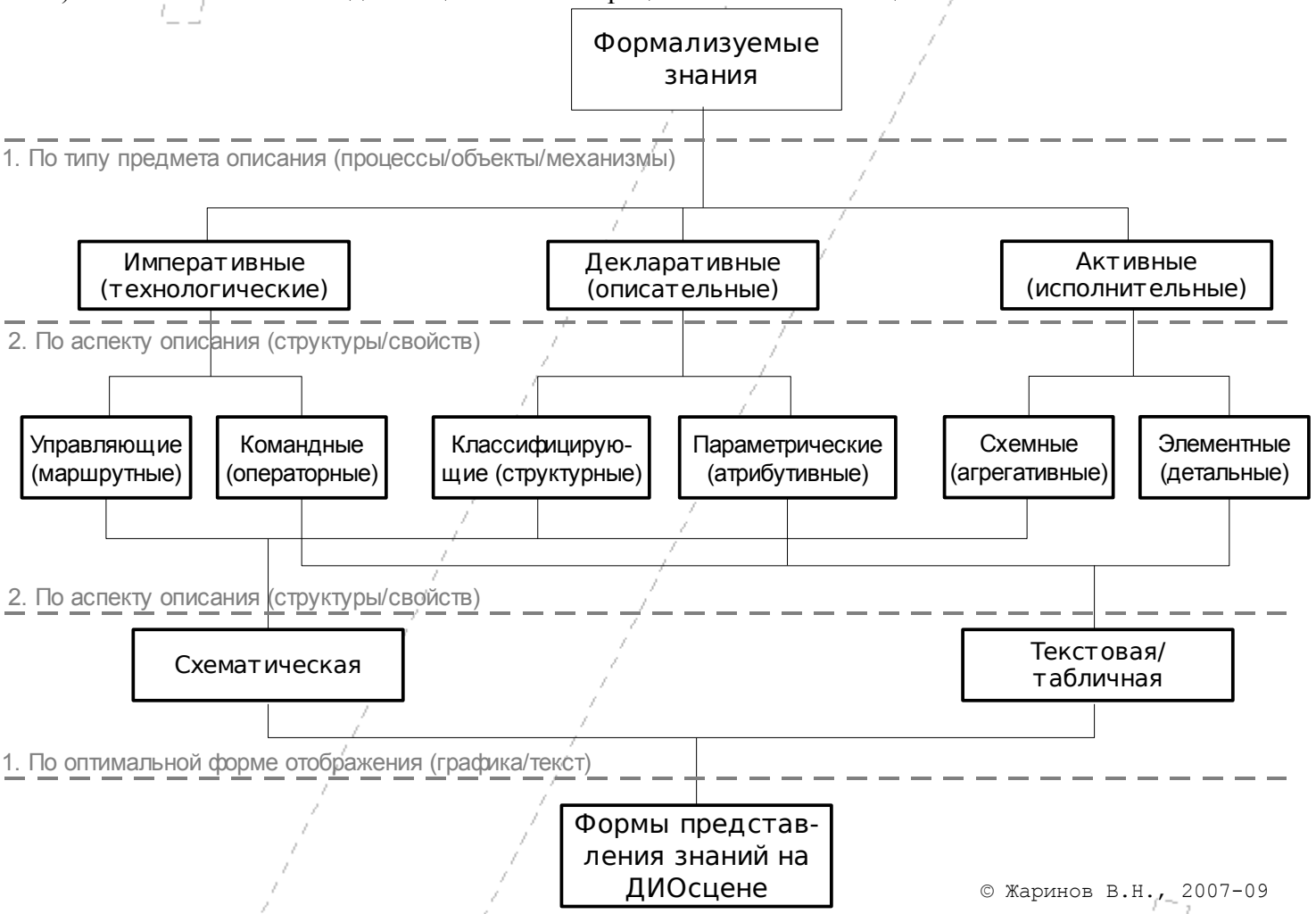
---

[30] Подробнее различие между подходами обсуждается, напр., в работе: Агафонов В.Н. Спецификация программ: понятийные средства и их организация. – Новосибирск: Наука, Сиб. отд., 1990. – С. 39.

[31] О современной ТРИЗ см. напр. в работе: Орлов М. Основы классической ТРИЗ. – М.:СОЛОН-Пресс, 2006.

соответственно импер- и деклар-моделирование. Раскроем «секрет полишинеля», который, возможно, уже очевиден для вдумчивого читателя. Конечно же, структура частных знаний, данная в начале цикла, неполна. К импер- и деклар-знаниям следует добавить знания об исполнителях (актив-знания). В результате классификация частных отчуждённых знаний изменится; в ней появится третья подструктура, аналогичная двум введённым ранее (см. схему).

Элементы актив-знания содержатся в различных местах цикла. Более широко, любая дисциплина, имеющая предметом к.-л. средства и системы, относится к актив-знанию. Как и другие рода отчуждённого знания, оно м.б. получено как денотационное, операционное и аксиоматическое. Традиционно первые два подхода совмещаются при описании реальных систем. Так, в описаниях промышленного оборудования всегда содержатся разделы об их устройстве (составе и свойствах составных частей) и принципе действия (но не решения конкретной задачи). Это и можно считать денотационной и операционной составляющими актив-знания.



Полная классификация частного отчуждённого знания

Отметим, что эти соображения не являются принципиально новыми. В своё время к тому же пришёл, скажем, один из патриархов информатики Н. Вирт, что он и отразил, в частности, в своей Тьюринговской лекции 1989 г.[32]

Формализация знаний об исполнителях осуществляется в различных отраслях знания. Так, в инженерной психологии в своё время предпринимались попытки описания работы операторов человеко-машинных систем так же детерминированно, как технических средств, напр. через функциональные схемы с передаточными функциями, однако выяснилось, что в большинстве случаев при этом теряется существенная информация о человеческой деятельности. Сегодня используются иные подходы, и в т.ч. процедурное описание человека как исполнителя; при этом часто начинают с весьма общего, глобального уровня, так сказать, «формализации смысла жизни». На техноязыке в

[32] Публикация на русском языке: Вирт Н. От разработки языков программирования к конструированию компьютеров. – Микропроцессорные средства и системы, №4/1989. – С. 42-48.

частности, этим занимается Ю. Феодоритов, чьи результаты можно найти в /8, тема Дракон-схемы сознания, самосознания и свободы воли/. С системной т.зр., в принципе можно поставить глобальную задачу целенаправленной системы, из которой любая конкретная практическая задача будет следовать как частная; на основе этой постановки можно определить обобщённую архитектуру исполнителя. Интересно, что для человека как естественной интеллектуальной системы-решателя произвольных задач обоснование закономерности формирования его облика также было дано И.А. Ефремовым в ряде его литературных произведений. Та же проблема разрабатывается в рамках общей теории безопасности, одна из западных разновидностей которой известна у нас под названием секьюритологии; здесь можно отметить работы В. Ярочкина, В.А. Герасименко и С.П. Расторгуева. Автору документа довелось в учебных целях разрабатывать постановку глобальной задачи функционирования оргсистем с позиций теории безопасности; при этом также оказалось необходимым выходить на глобальные цели человека как ведущего компонента таких систем. Полученные результаты также оказались весьма общими; можно лишь отметить, что возможно сформулировать назначение системы как обеспечение безопасности в широком смысле – не только как физического выживания («спасения шкуры»), но и поддержки выживания и развития других, т.е. так, как это трактуют общеизвестные морально-этические системы, – а также дать функциональную модель реализации ОБ.

С информатической т. зр. вопросы поведения человека как звена информационных систем разрабатываются в социальной информатике, в основном в целях изучения сферы массовой информации и пропаганды; здесь можно отметить результаты Г.Г. Почепцова и С.А. Муратова. Однако не следует ожидать вывода точных закономерностей совершения человеком действий в рамках оргсистем; разумнее сегодня исходить из позиций Фридланда и Симоновича, которые определяют человека как субъекта, взаимодействующего с искусственными системами определённо по формам (сенсорным и эффекторным каналам), но произвольно по содержанию. Детальную формализацию деятельности нужно проводить применительно к конкретным задачам и человеко-машинным системам.

Если сузить рассмотрение до исполнителей-машин, то следует отметить, кроме упомянутой ТРИЗ, вновь работы по структурированию информашин в рамках инфорбезопасности. Здесь основополагающими можно считать результаты Герасименко (/3, Гл. 7/) и В.В. Мельникова. Кроме того, целостный подход и к понятию информации, и к средствам её переработки в своё время предлагал В.В. Фурдуев, видный советский электроакустик.

Итак, при конкретной (детальной) формализации знаний нам нужен третий класс частных МФЗ. Посмотрим, что можно предложить для этой цели из имеющегося арсенала моделирования.

Примером реализации полного (в смысле охвата всех указанных классов) набора средств формализации знаний может служить язык UML. Нетрудно видеть, что это на самом деле семейство языков, решающих разные задачи. В частности, есть и средства описания механизмов решения задач, но недостаточно развитые; открыт и вопрос о целостности описания, получаемого с их помощью. В то же время ещё в бывшем СССР предпринимались усилия в этом направлении; в частности, для моделирования и проектирования оригинальных систем автоматизации в отрасли связи был разработан т.н. структурно-топологический метод описания распределённых и иерархических инфорсим и соответствующий ему СТО-язык[33]. Его можно использовать для визуализации структуры исполнителей, указывая реализуемые в узлах процессы и передаваемые между ними объекты, сопрягая при необходимости с картами местности и/или планами строений, техописаниями оборудования как атрибутами элементов СТО-схем. Однако нужно рассмотреть вопрос об информатической строгости данного языка и других его характеристиках как актив-ЯПЗ.

Концепция описания искусственных систем нормативно задана в семействе стандартов ЕСКД. Здесь мы видим как различные категории средств структурного описания по назначению (структурные, функциональные, принципиальные схемы), так и различные типы схем в зависимости от физического принципа действия структурных элементов (механический, электро-

[33] См. Кудрявцев Г.Г., Мамзелев И.А. Микропроцессоры и микроЭВМ в системах технического обслуживания средств связи. – М.:Радио и связь, 1989. – П.1.3.

магнитный, гидро-пневматический). Говорить о единстве языка описания структуры механизмов, т.о., не приходится.

> Формируемый последователями ТРИЗ каталог физических эффектов можно рассматривать как алфавит принципов действия элементов искусственных систем. Возможно, что на его основе можно задать графический язык структурного описания механизмов единообразно и независимо от принципа действия, подобно техноязыку.

Применительно к инфорсимам сегодня также действует отечественная система стандартов КСАС и стандарты Международного электротехнического комитета (МЭК). В частности, среди стандартов МЭК имеются нормирующие принцип структурного описания технических систем; пример их системного использования можно найти в современной работе по автоматизации производства[34], где автор на добрых двух десятках страниц опровергает заявления разработчика АИС управления объектами о классификации по стандартам МЭК на отказоустойчивость метаструктурного построения одной из линеек его продукции. С содержательной точки зрения, в рыночном обществе часто продукт представляется потребителям по принципу «коммерческой правды»; однако в данном случае имеет место расхождение заявлений и действительности, которое автор работы определяет как «технический миф». Нас же интересует формальный аспект: какими средствами и способами, инвариантными к данной конкретной ситуации, этот миф развенчивается.

В упомянутом отрывке автором работы фактически строятся информатически строгие определения отказоустойчивых метаструктур на базе стандартов, а также проводится формальный анализ соответствующих заявлений разработчика, для чего сведения о его продукции также приводятся к информатически строгому виду (с восстановлением умолчаний и прояснением неопределённостей). В результате появляется возможность определить истинный класс отказоустойчивости указанных структур не только математически, но и информатически, «на пальцах» показать по шагам алгоритмического процесса, что происходит при тех или иных условиях обстановки с анализируемой структурой и со стандартными структурами заявленного и фактического классов по МЭК; тем самым делается очевидным, для какого класса анализируемая структура служит примером, а для какого – контрпримером.

Для нашего предмета важно, что для представления здесь используется как визуально-структурный язык, так и ограниченный естественный язык, т.к. возможностей первого недостаточно.

В общем и целом, поле формализации актив-ЯПЗ ещё непаханное; к счастью, пока, по-видимому, можно ограничиваться возможностями СТО и других подобных языков.

Вернёмся к деклар-направлению конкретной формализации. Оно разрабатывается для ДРАКОНа в связи с реализацией дракон-сред широкого применения. При этом часто можно видеть, напр. на веб-форуме /8/, уклон в преимущественно текстовую реализацию (напр. реализацию описаний величин как комментариев в дракон-схемах), тогда как здесь действует тот же принцип, что и для импер-моделирования: декларативное описание в своей основе д.б. визуальным. Более точно, необходим комплекс из классиф-подъязыка (визуального) и атриб-подъязыка, используемых «в связке», как маршрутный и командный подъязыки ДРАКОНа.

Комплекс деклар-языков должен позволять:

- употреблять типы величин, встроенные в систему "информашина-транслятор", на которой эксплуатируются программные коды алгоритмов, визуализированных в дракон-среде;
- задавать (конструировать) визуальные описания производных типов через встроенные и/или иные производные типы в виде граф-схем (т.е. поддерживать визуальный классиф-подъязык, описывающий определение произвольного типа через иные типы аналогично тому, как импер-язык ДРАКОН описывает структуру алгоритма через иконы и/или другие алгоритмы);
- атрибутировать каждый тип, т.е. назначать ему конкретные значения параметризованных свойств на некотором атриб-подъязыке (пример неформального задания параметров – язык атрибуции данных из /3, Прил. 2/), причём пользователь должен иметь возможность зада-

[34] Фёдоров Ю.П. Справочник инженера по АСУТП. – М.: Инфра-Инженерия, 2008. – С. 34-50.

вать значения переменных параметров в диалоге реализации подобно тому, как это делается для типов данных в визуальных СУБД;

- указывать изменяемые характеристики отображения величин встроенных и производных типов на ресурсы реальной системы (для инфорсим – модель основной памяти, устройств ввода-вывода и иных программно-доступных элементов).

Разумеется, реализация не д.б. узкоспециализированной, для чего за основу берутся прогязыки, поддерживаемые средой в качестве стандартов немаршрутной части гибридного техноязыка (Оберон, 1С и т.д.). Если разработка ведётся на несовместимой системе, необходимы средства определения встроенных типов для целевой системы.

Примером может служить комплекс языков, определённый в Разд.2 Приложения2; однако это языки качественного уровня, не обладающие информатической строгостью. Символику можно использовать как основу алфавита визуального классиф-языка.

> Деклар-язык ФЛОКС мог бы служить языком конкретной формализации деклар-знания, но он достаточно узко специализирован и "приближен к машине"; тем не менее ФЛОКС можно расматривать как приближённое определение требований к такому языку, своего рода "печку", от которой можно "танцевать" при выработке стандарта деклар-языка, сопрягаемого с ДРАКОНом. В свою очередь примером машинно-независимого деклар-языка неопределённо широкого применения можно считать язык информационного моделирования методологии формализации деклар-знаний IDEF1X; в тоже время в нём прослеживается приближение к реализации IDEF1X-моделей как СУБД, т.е. некоторая зависимость от средств реализации в широком смысле имеется.

По сути, все эти языки представляются скорее денотационными. Есть и иной вариант, уже всплывавший при обсуждении дракон-сред в /8/: использовать операционно-ориентированный классиф-язык. Текстовым средством подобного рода являются БНФ-языки. По сути, их визуализацией являются языки т.н. синтаксических диаграмм. Если по аналогии с СТО-языком использовать условные вершины для задания определённых условий ветвления и цикла (вместо разветвительных вершин диаграмм), то в принципе получим аппарат визуализации структур объектов (в т.ч. типов величин).

Возможно, для реализации визуального классиф-языка такого рода необходимо математически обосновать условия укладки схем данных на плоскости без пересечений (разрывы устраняются, как и в техноязыке, иерархической декомпозицией через подстановки), подобно тому, как это сделал Пароджанов для схем алгоритмов.

Обоснованием состава частных языков с т. зр. автора документа и служит вышеприведённая полная классификация. Вместе с обобщёнными языками они должны образовывать полное множество (комплекс) средств формализации профессиональных знаний. Представлены также предложения автора документа о свойствах этих языков и их реализаций.

К имеющимся комплексам ЯПЗ и их реализациям следует подходить критически (см., напр., статья об UML в Википедии). В Справке по Ты-среде также указано следующее: «...UML содержит большое разнообразие по назначению графических диаграмм, графических объектов и элементов отображения их взаимодействия. Соответственно, UML сложно освоить, использовать для документирования и общения с не квалифицированным персоналом. UML не имеет интеграции со средой программирования.» Однако главные недостатки UML, по-видимому, всё-таки в другом, а именно в стихийном формировании этого семейства языков, вследствие чего:

- состав UML-языков не обоснован должным образом;
- языки не проработаны когнитивно-эргономически сами по себе и как части целого (не всегда есть чёткая взаимосвязь разных описаний);
- нет должной формальности ряда языков (потому и нет прямого выхода на программирование).

Критикуя, не следует «выплёскивать с водой и ребёнка»: взамен UML нужно предложить именно систему языков представления знаний, только целостную, минимально необходимого состава и имеющую выход на реализацию до уровня командных моделей (программ для информашин и/или формальных, «педантически» строгих инструкций для человека-исполнителя). В этой системе семейство гибридных техноязыков представляется достойным содержанием импер-компоненты, но нужны усилия по адекватному наполнению других компонент. Конечно, это вопросы к дракон-сообществу в целом.

А ЧТО ЖЕ ДАЛЬШЕ?

Данный сайт – лишь введение в мир ДРАКОНа. Какую ценность может представлять то, что Вы узнали здесь? Можно сразу указать некоторые возможности.

Во-первых, развитие алгоритмического мышления. Обратившись к классикам, можно вспомнить слова советского писателя А. Некрасова, сказанные устами его героя – капитана Врунгеля: «Навигация – это наука, которая учит нас избирать наиболее безопасные и выгодные морские пути, прокладывать их на картах и водить по ним корабли... Навигация – наука не точная. Для того, чтобы вполне овладеть ею, необходим личный опыт продолжительного практического плавания». То же можно сказать и об алгоритмизации, имея в виду под «морскими путями» структуры деятельности, и как частный их случай – алгоритмы функционирования автоматизированных систем. Однако неточный – не значит путаный, излишне неопределённый – а так получается, если формализовать деятельность на известных текстовых языках. Именно эту путаницу в алгоритмизации и снимает техноязык. Регулярно применяя его к описанию повседневных задач (пусть даже мысленно), Вы вырабатываете, говоря модным сейчас языком, «алгоритмическую компетенцию». Это повышает Вашу ценность на рынке труда; если более глубоко вникнуть в сферу когнитивно качественной формализации знаний, то можно даже стать системным аналитиком (постановщиком задач).

Во-вторых, повышение эффективности автоматизации. Сегодня стремятся автоматизировать самые разные процессы все шире и глубже; понятно, что при этом растет объем работ, но, кроме того, увеличивается «средняя цена» ошибки автоматизации – ведь искусственная система берет на себя больше и работает часто за рамками возможностей человека. Поэтому растет число случаев, когда уже нельзя оставлять ИТ-специалиста практически «один на один» с формализацией знаний заказчика; последний сам должен более точно, обстоятельно – и вместе с тем системно, целостно – поставить первому задачу на автоматизацию. Точно так же и программист, придерживаясь строгого и удобного метода, может выиграть в сроках, не потеряв в качестве.

> В ряде случаев этот результат приобретает ценность не только для качественной и информатической стадий, но и для математической в силу следующих обстоятельств. Сегодня установлено, что традиционные математические модели и методы решения задач (напр. автоматического управления) в ряде случаев неверны[35]. Нахождение верных решений требует более сложных методов, и значит, объективно требует дополнительного времени математика. А где взять это время? Только на других стадиях – у аналитика и разработчика. Да и сама модель может получиться сложнее для реализации; тогда и у разработчика появится дополнительный объем работ.

В-третьих, облегчение обучения. Сегодня образование становится «всё более непрерывным», не в последнюю очередь за счёт внедрения управления качеством по международным стандартам. Они основаны на процессном подходе, где импер-знания о деятельности составляют основу. При этом практикуется самообучение и взаимное обучение, активная работа с нормативно-техническими документами; все это требует максимального когнитивного качества представления знаний. Да и там, где не внедрены новые стандарты, это не будет лишним.

У внимательного читателя может возникнуть вопрос: почему описание языка не содержит примеров визуализации? Ну, во-первых, коль скоро техноязык предназначен для формализации любых процессов, то примерами могут (и должны) стать Ваши задачи. В их визуализации в отсутствие дракон-системы Вам могут помочь машинные документы, размещенные под рубрикой Полезное. А во-вторых, создателем ДРАКОНа подобран ряд примеров из разных предметных областей; их можно найти в /1, Гл.13/. В-третьих, некоторые частные примеры собраны в Каталог примеров.

[35] Обзор проблем для широкого круга читателей см.: Петров Ю.П. Новые главы теории управления и компьютерных вычислений. – СПб.: БХВ-Петербург, 2004. Конкретные результаты для ряда классов задач см.: Петров Ю.П. Обеспечение надежности и достоверности компьютерных вычислений. – СПб.: БХВ-Петербург, 2008.

В качестве примера мозговой проверки визуалов попробуйте найти ошибки в примерах визуализации.

Следуя духу тринитарного мышления (см. п. 1.4.2), автор стремился дать скорее целостное, чем полное описание формализации знаний на техноязыке; возможно, где-то оно вышло лишь приближённым.

С общесистемных позиций скорее следует говорить о ***новом системном мышлении***, в котором целостность описания и переход к тернарной структуризации являются одними из необходимых условий.

На техноязыке можно визуализировать и модные современные нанотехнологические процессы, и старые проверенные алгоритмы[36] медицинских манипуляций. В то же время при формализации, как и в любом деле, не стоит забывать сказанное одним из героев братьев Стругацких: «инструкция – это для тех, кто ещё не умеет». Конечно, это не значит, что можно нарушать рамки, установленные нормативными документами (особенно не рекомендуется это делать с ограничениями по безопасности :)) – просто типичная инструкция внутри этих рамок определяет один, в лучшем случае два-три нормативно одобренных пути достижения цели, а в любом живом деле их может быть гораздо больше, и тогда выбор (с соблюдением ограничений) остаётся за исполнителем. Так и с техноязыком – его синтаксис хоть и формален, но допускает определённую свободу выражения. Главная цель автора – не столько ввести нормативно одобренные пути выражения, сколько показать рамки этой свободы, внутри которых каждый активно пользующийся ДРАКОНом выработает свой стиль визуализации. Конечно, всё сказанное здесь – это личный взгляд автора на формализацию деятельности; рекомендуем ознакомиться с различными мнениями (благо в последние годы появляются источники по этой теме, хотя бы в виде сетевых ресурсов). Кроме того, для дальнейшего развития в этом направлении Вы можете найти единомышленников (для этого можно, в частности, провести поиск в интернет по терминам данного ресурса), пройти подготовку по визуализации (если она доступна там, где Вы живёте). Успехов Вам в когнитивно-эргономичной формализации знаний!

[36] Вообще-то техпроцессы, т.к. манипулируют здесь не с данными, а с реальными объектами.

6. ИСТОЧНИКИ

Документы для ссылок

1. Пароджанов В.Д. Как улучшить работу ума. Алгоритмы без программистов – это очень просто! – М.: Дело, 2001.

2. Пароджанов В.Д. Занимательная информатика. – М.: Дрофа, 2007.

3. Баранцев Р.Г. Становление тринитарного мышления. – М.-Ижевск: НИЦ «Регулярная и хаотическая динамика», 2005.

4. Фридланд А.Я. Информатика: процессы, системы, ресурсы. – М.: БИНОМ. Лаборатория базовых знаний, 2003.

5. Функциональное моделирование. Методология IDEF0: Стандарт/русская редакция. – М.: МетаТехнология, 1993.

6. С.В. Черемных, И.О. Семенов, В.С. Ручкин. Структурный анализ систем: IDEF-технологии. – М.: Финансы и статистика, 2001.

7. Герасименко В.А. Защита информации в автоматизированных системах обработки данных.– М.: Энергоатомиздат, 1994.

8. Визуальный язык ДРАКОН: веб-форум. – Конференция OberonCore.

Полезные ресурсы

mgopu.ru/PVU/2.2/Sprint-Inform – здесь можно изучить современную информатическую терминологию, введённую в /4/.

Конференция OberonCore.ru – на ней обсуждаются, кроме ДРАКОНа, также элементы информатической культуры и практический инструментарий программирования.

Усилители интеллекта – рассылка, посвящённая средствам облегчения умственного труда и в частности, техноязыку.