Алаверды Дракону

Дмитрий Дагаев

Оглавление

Предисловие

Дагаев Дмитрий Викторович dvdagaev@mail.ru с 1987 года в атомной энергетике занимается разработкой и внедрением систем управления, искусственного интеллекта и моделирования на основе графических форм представления информации. В разное время были разработаны модели систем управления для тренажеров на основе языков стандарта МЭК 61131-3 (Functional Block Diagrams, Sequential Function Charts), экспертные системы обучения (симптомно-ориентированные инструкции) и диагностики (течи теплоносителя 1 контура), СКАДА-платформа СУОК и реализованные на ней общестанционные системы ИВС э/б 1 РоАЭС и 1 КлнАЭС.

Введение

«-У меня есть отличный коньяк
- У меня тоже есть коньяк
- Зато у Вас наверняка нет салями
- У меня есть салями
- Значит, мы с Вами хлебаем из одной тарелки»
Генерал в вагоне – Штирлиц

Недавно ко мне попало описание языка ДРАКОН и когнитивного подхода В.Д.Пароджанова[1] и, в силу рода занятий, вызвало большой интерес. Надо заметить, что российская автоматизация тяготеет в следовании западным подходам, стандартам, программным продуктам. В то время как наши собственные разработки полностью неизвестны за рубежом, да и внутри страны тоже. В части автоматизации имеются стандарты МЭК 61131-3 (языки программирования) и МЭК 61499 (функциональные блоки для программирования) и практическое применение, скажем, ДРАКОН-технологий в автоматизации людьми, принимающими решение зачастую будет рассматриваться как подход нестандартный, неформатный, а подчас и архаичный. По мнению специалиста [2], рекламирующего эти стандарты в 2003, «при знакомстве с доступными источниками 10 летней давности создается впечатления, что применение ПЛК связано с примитивным и неуклюжим программированием при помощи специализированных пультов, что выглядит чрезвычайно архаично».

Автор считает, что нужно продвигать отечественные технологии, которые в данном случае лучше принятых стандартов и пытается помочь и проинформировать об аналогичных проектах в атомной энергетике. Автор считает, что ДРАКОН должен быть принципиально лучше, чем используемый в МЭК 61131-3 «Sequential Function Chart» (SFC). В качестве стиля изложения я взял Алаверды – концепция ответного тоста: от нашего атомного стола к вашему космическому. Если и будет содержаться критика, то, надеюсь, она будет конструктивной и доброжелательной.

1. Может ли быть маршрутный язык ДРАКОНа декларативным?

Прежде чем ответить на этот вопрос, я опишу систему поддержки оператора СИДОР, которая делалась по заказу концерна РосЭнергоАтом в 1997г. Симптомно-Ориентированные Инструкции основывались на формализованном представлении аварийных процедур, разработанных ранее для АЭС под патронажем и в стиле Вестингауз. Фрагмент двух из 29 шагов приведен на Рис 1.

4.ϕ - 0		[illegible]		
[illegible]		[illegible]	[illegible]	
[1.]		[illegible]		
	-	[illegible]		
	-	[illegible]		[illegible] 1-[illegible] 2-[illegible]
	-	[illegible]		[illegible]
	-	[illegible]		[illegible] 4.[illegible] .1 [illegible] 1.
[2.]		[illegible] 1-[illegible] 2-[illegible]		

Рис.1 А-0 – Фрагмент аварийной процедуры А-0

Данная процедура определяет алгоритм действий персонала сразу после срабатывания аварийной защиты реактора АЗ-1. Шаги выполняются последовательно по номеру шага. Средняя колонка содержит Действия/Ожидаемый Результат. Действия выполняются, Ожидаемый Результат проверяется. Если все успешно, выполняется сверху вниз левая колонка (Главный маршрут по шампуру!). Если результат не получен, то выполняется правая, плохая колонка.

Нарисуем схему на языке ДРАКОН для двух шагов инструкции.

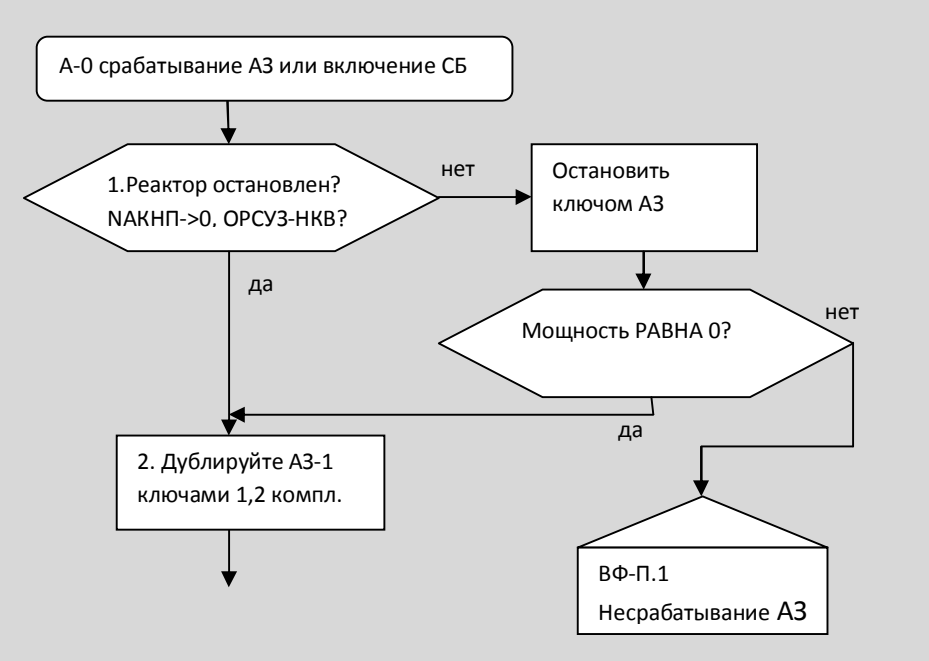


Рис.2 ДРАКОН-схема фрагмента аварийной процедуры А-0

Система СИДОР была разработана как экспертная система, сопряженная с тренажером АЭС, и оператор выполнял инструкции, следуя введенным в систему блок-схемам алгоритмов. Аналогично зарубежным аналогам COMPRO и EOPTS имелись функции экспертных систем: машина вывода и механизм объяснений.

Шаги по алгоритму ВЫПОЛНЯЛ ОПЕРАТОР а система СОПРОВОЖДАЛА и ОТСЛЕЖИВАЛА его действия. Сопровождение означает проверку выполнения им условий и действий. В частности, выполнено ли условие, отображалось системой подсветкой «Да», а при нажатии на вопрос «Почему?» для проверки уменьшения мощности выдавался текущий график мощности Рис. 3. Механизм объяснений содержит график, диапазон мощности в виде текста, число сработавших органов управления. Выполнение условия алгоритма отслеживается системой в подсветке, даже если ответ – «Нет», оператор выбирает вручную и может перейти по цепочке «Да». Отслеживание для условий (1.) означает алгоритм «остановись, подсвети по результату формулы и жди ввода оператора».

Как можно убедиться, алгоритм программы совсем иной, чем алгоритм действий оператора. При этом эргономически выверенный алгоритм – это сформулированные операторам инструкции, они представляют наибольший труд и ценность, а не какая-то вспомогательная их обучающая система, которую можно «накрыть полотенцем». Стало быть, маршрутный алгоритм в данном случае - действия оператора. А отслеживающая система написана на декларативном объектно-ориентированном языке с блок-схемой в качестве входных данных.

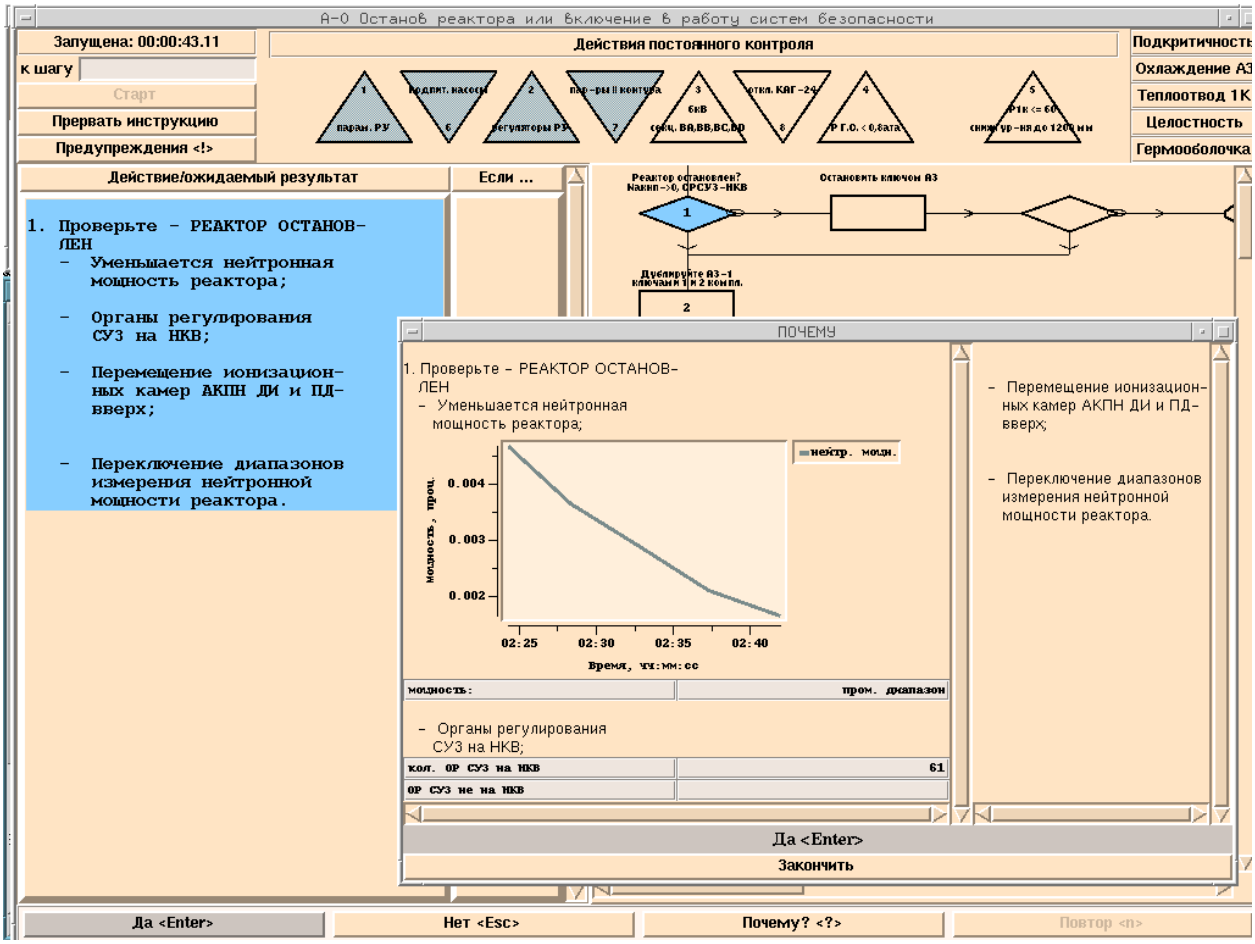


Рис.3 Система СИДОР, шаг 1 инструкции А-0.

Что же получается? Маршрутный алгоритм в системе отслеживания реализуется программой, декларативное описание блок-схемы которой загружается как данные! (Кстати, во избежание путаницы предлагаю не употреблять слово декларативный как антипод процедурному языку, как в [1], стр.283. Ибо декларативный используется еще в другом смысле: запись определений данных. Пускай будет процедурный/непроцедурный)

Если имеется блок-схема на входе, то что будет на выходе? Просто код на языке (С, Оберон)? Тогда подобная задача не решаема. Ибо для нее нужны ДАННЫЕ. В результате обработки схемы после редактора получается сложная модель объектов и связей.

Заметьте, если схема описывает алгоритм автоматического устройства (например, Буран), но на выходе нужен код на процедурном языке. А если процесс, то код на процедурном языке не нужен – требуется что-то другое.

А еще для схемы нужна и динамика. Разработчикам редакторов надо бы предусмотреть и отладчик. На Рис.4 отмечены цветом пройденные шаги. Шаг 4в не выполнен. Отладочная информация должна включаться в сгенеренный код, если это процедурный язык.

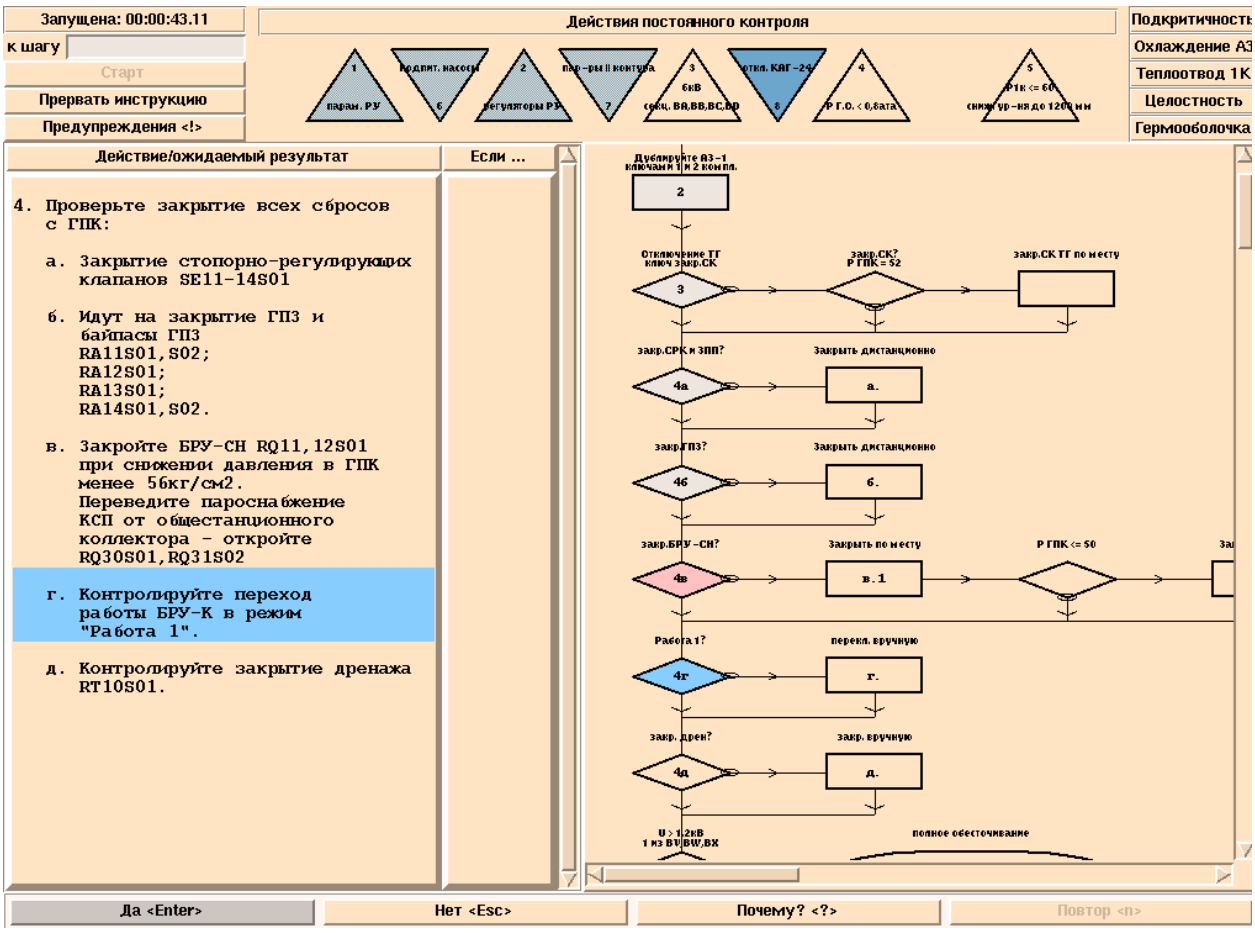


Рис.4 Система СИДОР, шаг 4г инструкции А-0

2. Может ли другой язык быть лучше ДРАКОНа в понимании алгоритмов и программ?

«Не поручают тигру ловить мышей»

Китайская пословица

В документе автора ДРАКОНА [3] рисунок 109 предлагает использовать ДРАКОН для программирования логических выражений. Можно, но вся когнитивность исчезает, в сравнении с стандартными языками представления логических алгоритмов управления. На рисунке справа представлена реализация логической функции на языке «Functional Block Diagram» (FBD) стандарта МЭК 61131-3, которая встроена в элемент языка ДРАКОН. Если кто-то еще сомневается, что диаграмма справа понятнее, Рис.7 иллюстрирует страницу альбома алгоритмов защит и блокировок (могу предоставить тысячи схем) для строящихся блоков АЭС, разработанных и утвержденных АтомЭнергоПроектом в рамках технологии Электронный Проект. И программисты подстроились под РАЗНЫЕ формы представления алгоритмов, стандарт 61131-3 включает в себя 5 (Пять!) разных языков. Про все 5 не скажу, но FBD точно подходит ДРАКОНу КАК КОМПЛЕМЕНТАРНЫЙ. У него другая форма визуального представления, к которой люди привыкли.

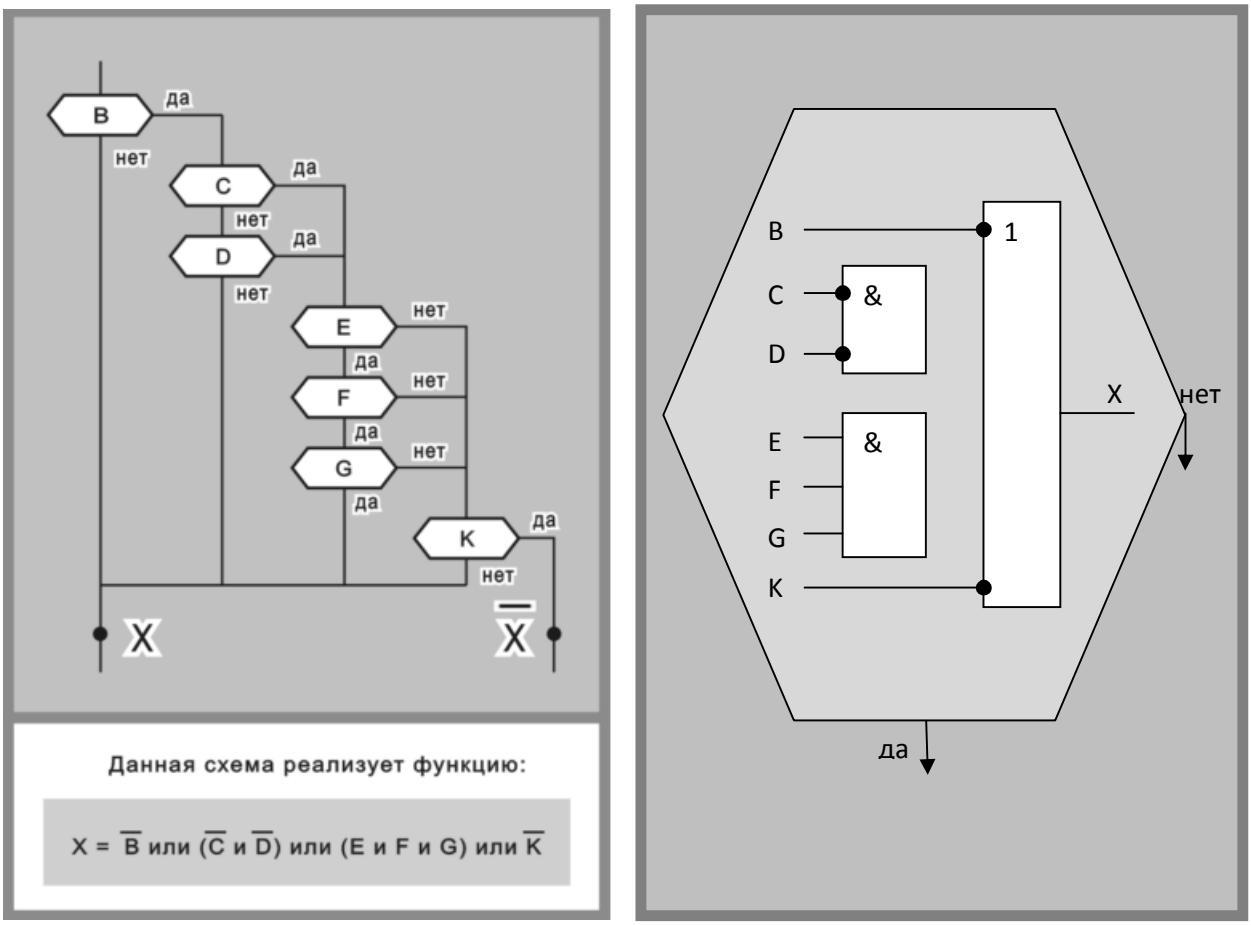


Рис.5 Представление логических алгоритмов

В 1993 г. автором был разработан пакет для моделирования автоматики АЭС на основе графического языка FBD (Рис. 6). Он состоит из входов (имена смотрят влево), выходов (имена смотрят вправо), функций (элементов с одним выходом) и блоков (элементов со многими выходами). Коричневым цветом сверху и снизу представлены описания. Я никоим образом не хочу рекламировать когнитивные решения (мне они сейчас не нравятся), реализованные в том проекте, я хочу подчеркнуть, что FBD неизбежен как язык автоматизации. Человечество добавило еще язык релейных схем в стандарт, вот он может как раз быть заменен на FBD. Блок Lmov11 справа – это схема внутри схемы, реализованная на текстовом языке. Хотя таким же образом может вставляться схема в схему. Это – одно из устоявшихся представлений схем.

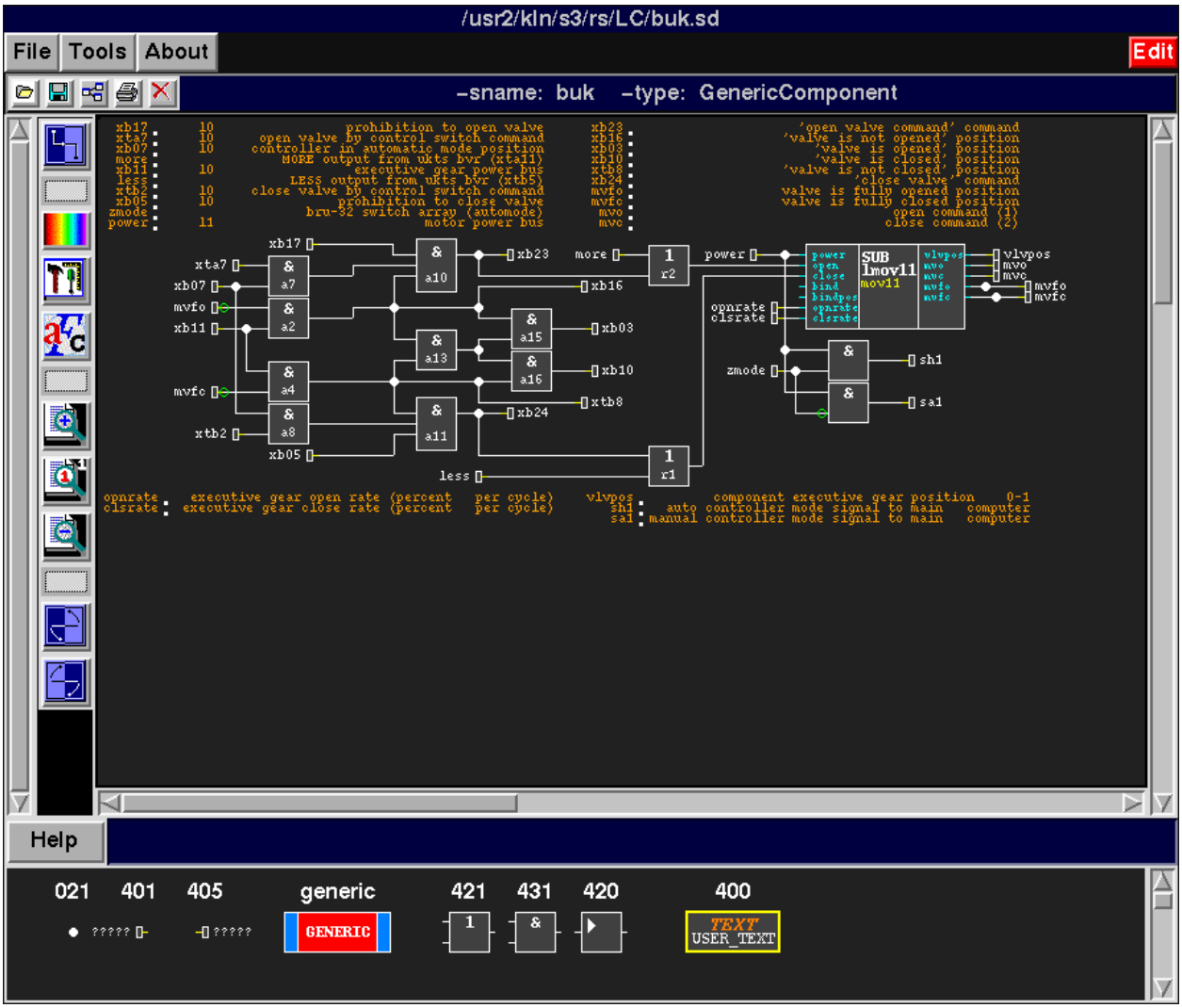


Рис.6 Схема блока управления клапаном для тренажера

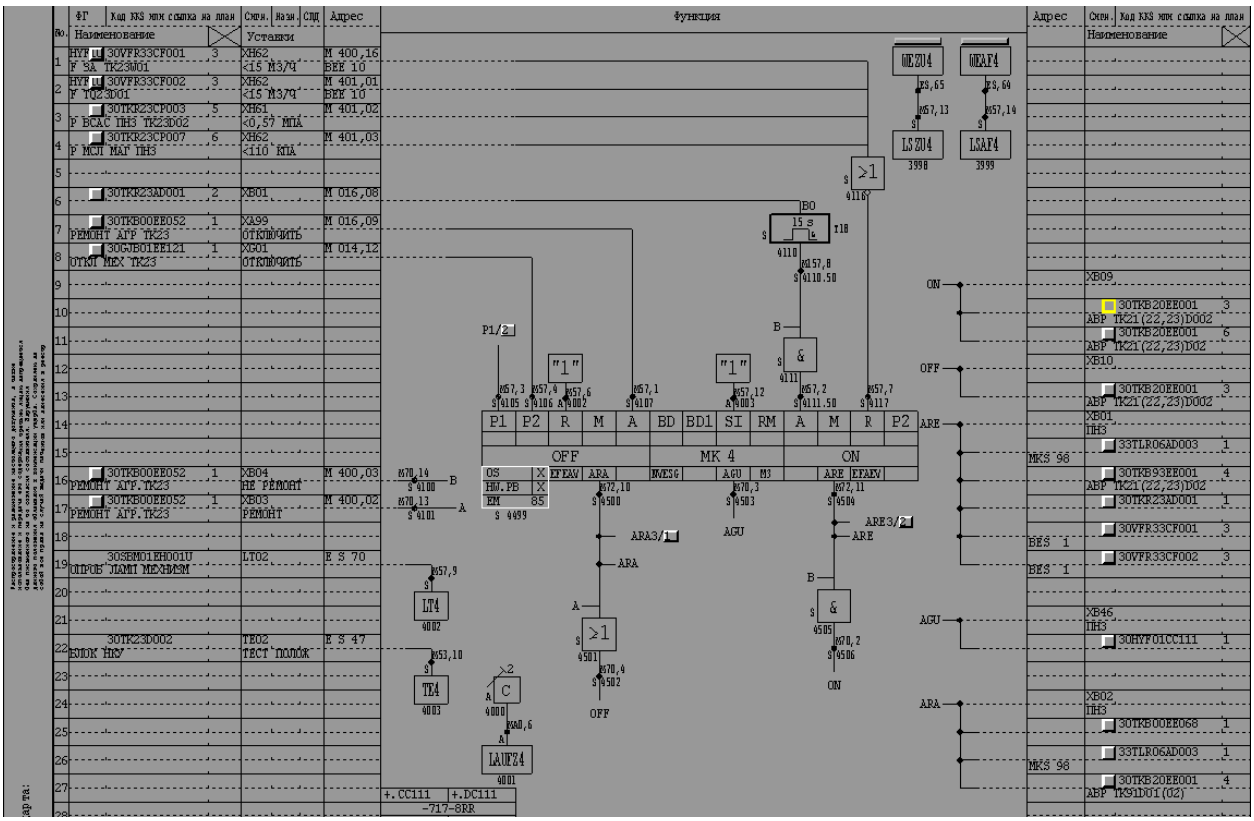


Рис.7 Схема обвязки насоса 30TRK23AD002

В визуальном языке программирования ДРАКОН-2 были реализации логических функций [1], но все приведенные рисунки названы эргономически неудачными. Привожу их на Рис.8 и попытаюсь объяснить.

Рис.8 Схемы управления роботом

Как утверждает автор, «для большинства специалистов, не знающих прикладную задачу, логическое выражение на Рис.77 не дает никакой подсказки о семантике логических переменных и фактически представляет собой загадочный ребус». Правильно. В качестве вывода предлагается графическим пакетом автоматический перевод из рамочного выражения (Рис. 78) в визуальное (можете нарисовать). Не правильно. На Рис.75/76 уже предложено правильное решение, но оно на ДВУХ ЯЗЫКАХ: алгоритм слева на ДРАКОНе и логическое выражение справа на текстовом языке, который в данном примере подходит ДРАКОНу как КОМПЛЕМЕНТАРНЫЙ. Я тут не оригинален, в стандарте МЭК 61131-3 есть еще два текстовых языка, сопрягаемых с алгоритмическими: паскалеподобный Structured Text (ST) и подобный макроассемблеру Instruction List (IL). Кстати, эргономически правильно не перегружать схемы, демонстрируемые операторам-технологам. Просто в имевшихся программных средствах не было возможности правильного объединения различных языков.

В качестве примера приведу видеокадр разработанной автором и введенной в промышленную эксплуатацию системы ИВС/СППБ 1блока Клн АЭС. Деревья СППБ показывают достижения блоком барьеров безопасности; алгоритмы их обоснованы и утверждены, изменить автор ПО ничего не имеет права. Выполняются они слева направо, от условий – к выводам. Динамическая информация подсвечивается на видеокадре: есть ли отклонения и каковы значения параметров. Это – информационная система, операторам предоставляется поддержка для принятия решений. Условия содержат текстовое описание на технологическом языке, например, «Суммарный расх ..». Вычисления этих условий осуществляются обработчиком формул softlogic SCADA-платформы СУОК и складываются в расчетную переменную. Есть командный язык и онлайновый байтовый компилятор, обрабатывающий логические выражения, возрастания, производные, функции и проч.

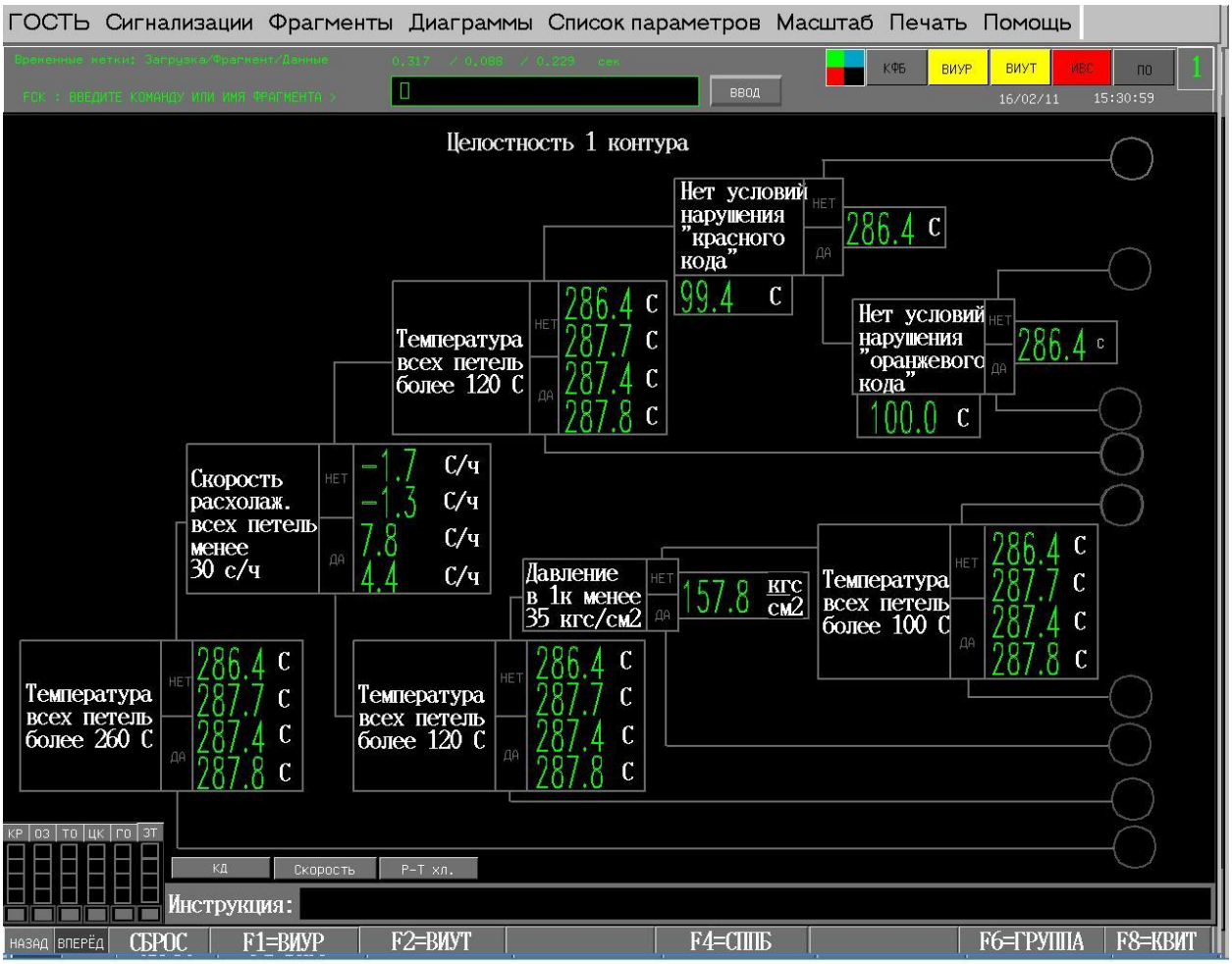


Рис.9 Видеокадр СППБ

С точки зрения эргономики нужно учесть следующее. Оперативный персонал интересует технология, они никогда не будут залезать в логические формулы. Другое дело – обслуживающий персонал, другие люди, другая форма доступа, другая квалификация (программисты). Для них – формулы. На Рис.10 приведен фрагмент системы в режиме редактирования базы данных. Формула вводится в п. 24 и доступна только программистам.

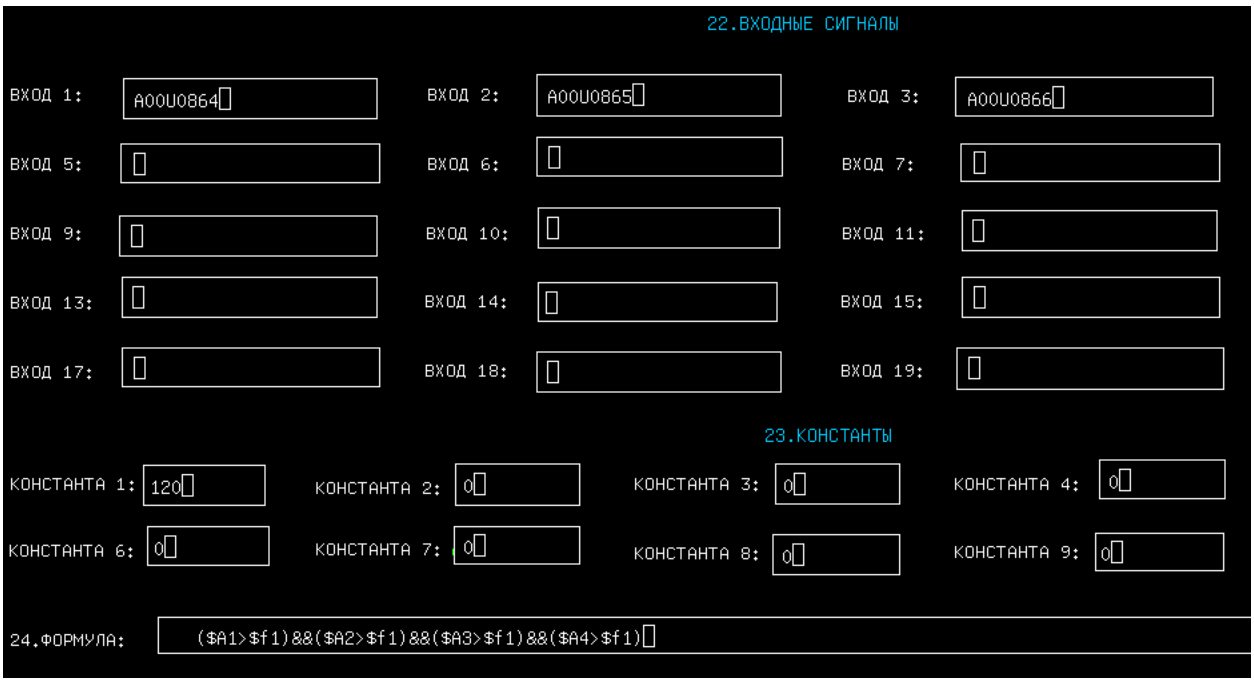


Рис.10 Редактирование формул базы данных реального времени

Да, формула представляет собой логическое выражение и написана на текстовом языке. Я еще раз подчеркиваю, что такое разделение естественно и определяется в каждом конкретном случае. Так же, как и в текстовом языке программирования, часть выражений структурируется в виде отдельной переменной, либо функции. Важно исходить из смысла задачи и не перегружать как схему, так и программу.

Еще одним примером неправильного использования ДРАКОНа является учебная экспертная система по химии за 10 класс [1].

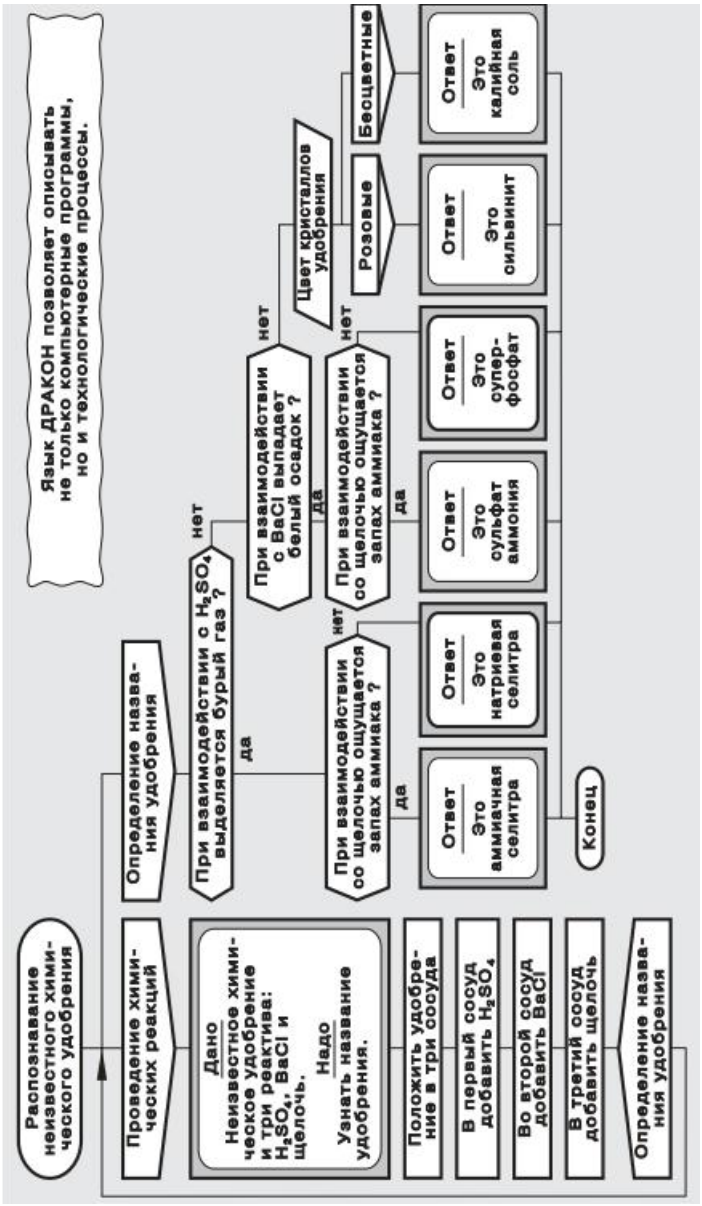


Рис.11 Предлагаемая технология химической диагностики

Задача диагностики определяет состав вещества в колбе на основе знаний предметной области (химии). Это может быть аммиачная селитра, натриевая соль, сульфат аммония, суперфосфат, сильвинит или калийная соль. Легко убедиться, что смесь веществ данный алгоритм не распознает. Ибо настроен на однозначное распознавание (да/нет), при котором не может быть 2 диагноза на 1 сценарий. С другой стороны, экспертная система должна позволять динамически добавлять правила, при этом может оказаться, что будет 2 правила на 1 диагноз. При этом правила могут и противоречить друг другу. И напоследок, механизм логического вывода реальных систем уже нужно строить на вероятностной логике, где, помимо ответа «Да/Нет» устанавливается достоверность [4].

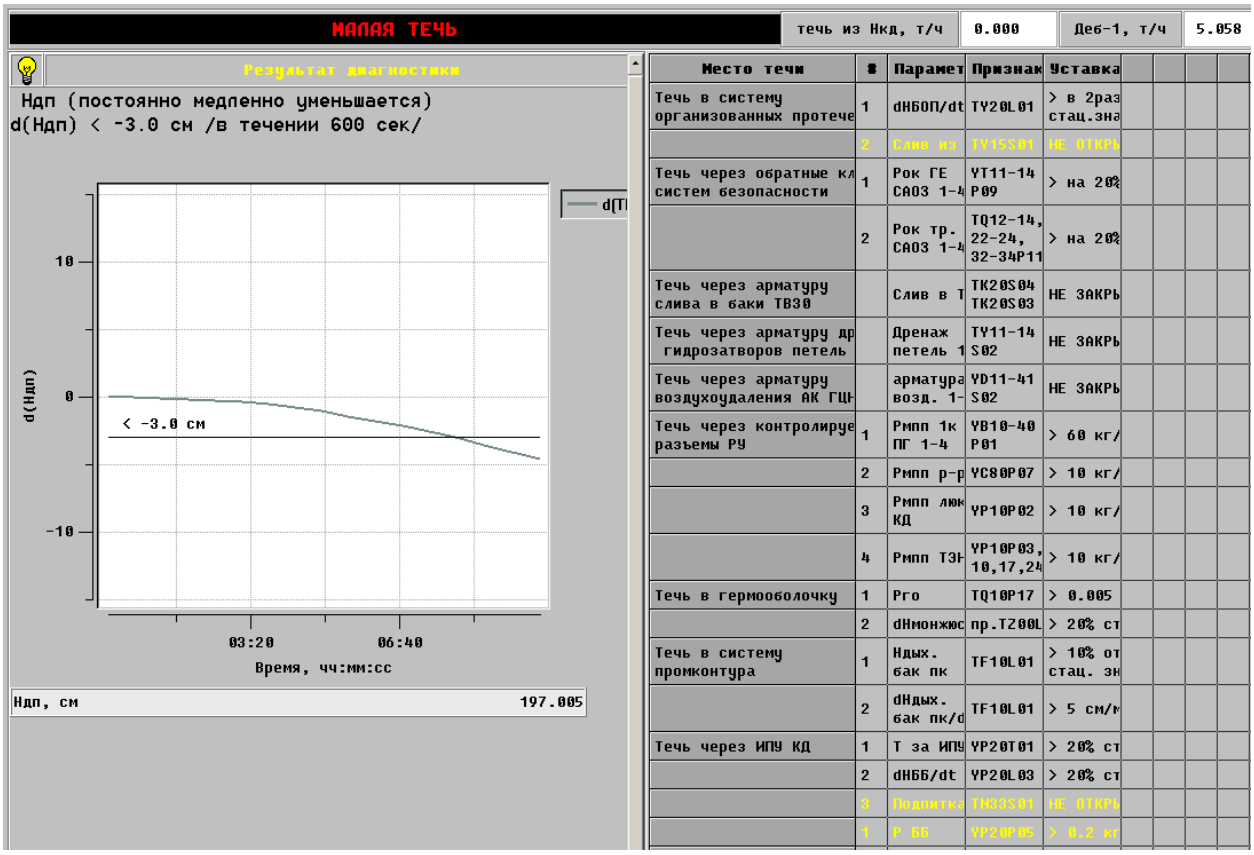


Рис.12 Диагностика течей теплоносителя

На Рис.12 приведен фрагмент системы Диагностики течей теплоносителя 1 контура, разработанной автором в 1998г. Слева на экране диагностируется характер течи, справа – место течи. Я специально привел сценарий, при котором место течи не определено однозначно, параметры, указывающие на течь (отмеченные желтым), расположены в «системе организованных протечек» и в «ИПУ КД». База знаний данной системы построена на объектах, каждый из которых представляет собой правило на непроцедурном языке, гипертекст для объяснения, краткие текстовые пояснения.

Вышесказанное иллюстрирует, что написание алгоритмов диагностики как на Рис.11 неправильно. Мало-мальское усложнение системы приведет к т.н. сдвигу парадигмы, когда ломается старая модель и нужно делать все по-новому!

3. Можно ли ДРАКОН совмещать с другими когнитивными языками?

«– Ну и что ж ты бы мог сделать?
– Я мог выдумать что-нибудь вроде счастья, а от душевного смысла улучшилась бы производительность.
– Счастье произойдет от материализма, товарищ Вощев, а не от смысла. Мы тебя отстоять не можем, ты человек несознательный, а мы не желаем очутиться в хвосте масс.»
А.Платонов. Котлован

Можно. Доказательство от противного, допустим – нельзя. Тогда какой смысл предлагать и развивать нестандартный ДРАКОН, когда есть SFC МЭК 61131-3, которому можно? Доказано.

Пока стандарта на язык даже не существует, и программистское сообщество думает как бы написать редактор, в атомной энергетике для новых блоков реализуется подход SmartPlant Foundation, когда создается полностью «электронный проект» блока АЭС и переносится в виде базы данных, видеокадров алгоритмов на понимающие эти форматы современные системы АСУ ТП на базе Siemens, Schneider, Areva. Есть западная технология, которую остается «заполнить пушечным мясом». Территория отечественных программистов сужается. Видимо, в космонавтике то же самое.

Поэтому считаю возможным дополнять и излагать. В частности, считаю, что ДРАКОН может стать краеугольным камнем для возникающих вокруг него языковых средств, которым будет уготовано долгое и мирное с ДРАКОНом сосуществование.

Нужен стандарт на маршрутный язык и, я надеюсь, космическая отрасль будет этим заниматься.

Как было показано на множестве примеров, даже системы среднего размера распадаются на два, иногда несколько уровней. База знаний и машина вывода, схемы и базовое ПО контроллеров, браузер и web-страницы в конце концов. Я согласен с упоминанием в [1] И.Вельбицкого, что структура – понятие многомерное. Я не согласен с выводами, что графический язык придет «на смену» текстовому. Объясню почему. Графические языки добавляют новые свойства, в которых они лучше, но при этом у текстовых языков остаются их «конкурентные преимущества»: единообразие всех данных, у кого объектно-ориентированность, у кого декларативность. К тому же, текстовые языки хорошо работают на уровне «обработки данных», тогда как графические – на уровне «интерфейса пользователя». Однако, перспективны и развиваются в первую очередь новые пользовательские уровни. Отмирают только те языки, которые не имеют конкурентных преимуществ, и, выражаясь языком линейной алгебры, будут линейно зависимы по отношению к группе новых. Принудительная «унификация» ни к чему хорошему не приведет, вспомним опыт сворачивания российских разработок БЭСМ-6 в угоду унификации с IBM-360 [5].

Сегодня мы можем обозначить такие языки. При этом нужны стандарты на интерфейсы СЕМЕЙСТВА КОМПЛЕМЕНТАРНЫХ ДРАКОНу языков (типы параметров, передача параметров – до уровня совместимости библиотек).

4. Что у ДРАКОНа должно быть на выходе?

Если визуальные алгоритмы представляют собой верхушку айсберга, такой «когнитивный уровень», то на выходе должен быть некоторые псевдо-язык, впоследствии преобразуемый либо в код на Обероне, либо на С, либо в декларативные данные в зависимости от задачи. Желательно иметь инвариантное ДРАКОН-схеме представление, чтобы редактор мог как загрузить файл с псевдо-языком, так и сохранить картинку в виде этого псевдо-языка.

1	Внятная схема
2	Графический редактор
3	Псевдо язык
4	Прикладной язык
5	Прикладная оболочка

Согласно стратегии «разделяй и властвуй» сложная программа разбивается на уровни. При этом, уровни 1-3 можно выделить и разрабатывать базовыми (на самом деле, под каждую задачу потребуется сделать отображение своих специфических схем). Уровень 4 представляет собой набор сменных модулей для каждой задачи, он и является искомым преобразователем.

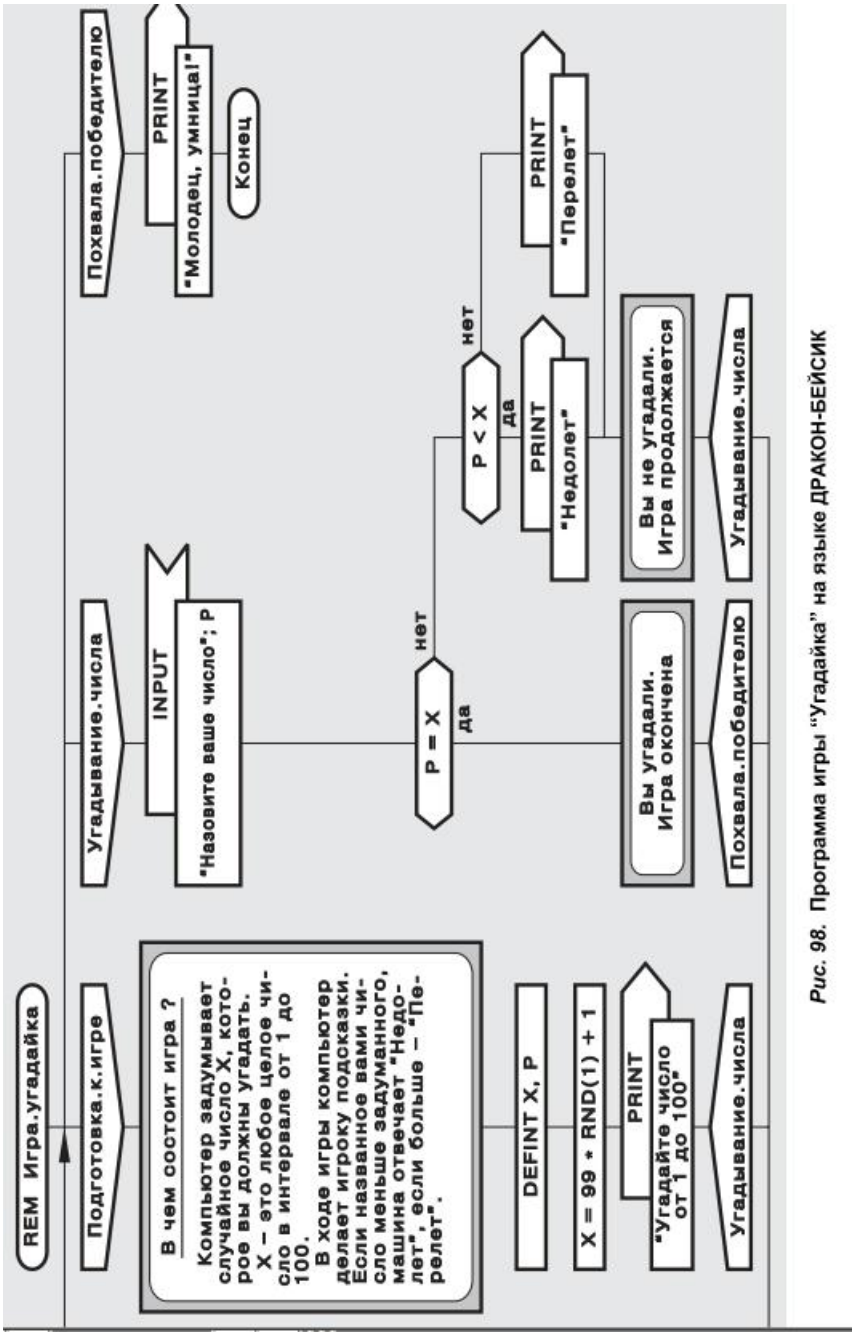


Рис. 98. Программа игры "Угадайка" на языке ДРАКОН-БЕЙСИК

Рис.13 Алгоритм программы Угадайка

Для алгоритма Рис 13 приведен пример псевдоязыка. Предполагается, что код на Рис.14 будет генериться после редактора. За основу взят tcl (это чисто мое предпочтение, Lisp или Perl вполне подойдет для этой роли; можно использовать XML, но будет несколько сложнее преобразование). Это – скрипт, выполняемый байтовым компилятором (или интерпретатором). Каждой иконе в схеме соответствует свой оператор вида: ВЕТКА, АДРЕС, ДЕЙСТВИЕ.

```
ЗАГОЛОВОК "Игра Угадайка" ugadaika

ВЕТКА "Подготовка к игре"
КОММЕНТАРИЙ {
В чем состоит игра?
}
ДЕЙСТВИЕ [INT X P] "Декларирование X,P в целых числах"
ДЕЙСТВИЕ "X = 99 * [RND 1] + 1" "X = 99 * RND(1) + 1"
ВЫВОД "Угадайте число от 1 до 100"
АДРЕС "Угадывание числа"

ВЕТКА "Угадывание числа" 2
ВВОД "Назовите ваше число" "INT P"
ВОПРОС [EQ P X] {
        КОММЕНТАРИЙ {Вы угадали.
Игра окончена
        }
        АДРЕС "Похвала победителю" 3
} {
        ВОПРОС "P < X" {
                ВЫВОД "Недолет"
        } {
                ВЫВОД "Перелет"
        }
        КОММЕНТАРИЙ {Вы не угадали.
Игра продолжается
        }
        АДРЕС "Угадывание числа" 2
}

ВЕТКА "Похвала победителю" 3
ВЫВОД "Молодец, умница"
КОНЕЦ
```

Рис.14 Программа Угадайка на псевдоязыке

Я включил два сменных модуля для генератора кода Basic и C (получилось 141 и 137 строк программного кода и 3 часа работы). Каждый оператор представляет процедуру на псевдоязыке. В результате получается нижеприведенный код на C и Basic.

Следует отметить, что выходные данные могут быть сгенерены в любом формате: декларативный язык, XML, скрипт для импорта в базу данных, что может понадобится, если мы рассматриваем алгоритм процесса, а не программы. Пример в XML вставлен для “P<X”.

```
<if expr=”P&lt;X”>
    <then> <print>Недолет</print> </then>
    <else> <print>Перелет</print> </else>
</if>
```

И во-вторых, с появлением спецификаций графического языка как стандарта можно будет как скорректировать синтаксис этого псевдо-языка, так и добавить графическую информацию (координаты икон и связей – это желательно). Я брал за основу «книжное описание» в [1],[8], поэтому выходной псевдо-язык тоже предложен в прототипной версии. Однако, реальная версия получится путем лишь небольшого расширения «базового комплекта».

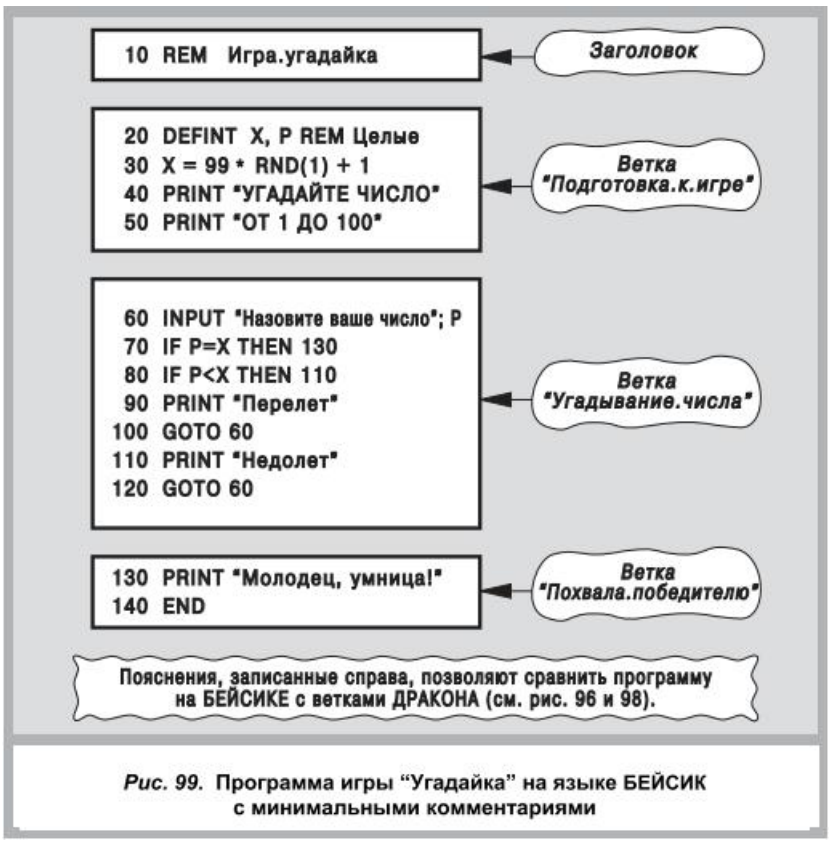


Рис. 99. Программа игры "Угадайка" на языке БЕЙСИК с минимальными комментариями

Рис.15 Программа Угадайка на Бейсике

```
 10 REM 'Игра Угадайка' ugadaika

 20 REM ВЕТКА Подготовка к игре
 ``
 `` В чем состоит игра?
 ``
 30 DEFINT X,P ``REM Декларирование X,P в целых числах
 40 X = 99 * RND(1) + 1 ``REM X = 99 * RND(1) + 1
 50 PRINT "Угадайте число от 1 до 100"
 60 REM GOTO Угадывание числа

  2 REM ВЕТКА Угадывание числа
 80 INPUT "Назовите ваше число"; P
 90 IF P = X THEN 100 ELSE 120
 `` Вы угадали.
 `` Игра окончена
 ``
100 GOTO 3
120 IF P < X THEN 130 ELSE 150
130 PRINT "Недолет"
140 GOTO 160
150 PRINT "Перелет"
 `` Вы не угадали.
 `` Игра продолжается
 ``
160 GOTO 2

  3 REM ВЕТКА Похвала победителю
180 PRINT "Молодец, умница"
190 END
```

```
void ugadaika ()
{
    // 'Игра Угадайка' ugadaika

L10:    ;       //ВЕТКА Подготовка к игре
    /*      *
     * В чем состоит игра?
     *
     */
    int X,P; // Декларирование X,P в целых числах
    X = 99 * rand(1) + 1; // X = 99 * RND(1) + 1
    printf("Угадайте число от 1 до 100");
    //goto ВЕТКА Угадывание числа

L2:    ;        //ВЕТКА Угадывание числа
    printf("Назовите ваше число");
    scanf("%d", &P);
    if (P == X) {
        /*          * Вы угадали.
         * Игра окончена
         *
         */
        goto L3; // Похвала победителю
    } else {
        if (P < X) {
            printf("Недолет");
        } else {
            printf("Перелет");
        }
        /*          * Вы не угадали.
         * Игра продолжается
         *
         */
        goto L2; // Угадывание числа
    }

L3:    ;        //ВЕТКА Похвала победителю
    printf("Молодец, умница");
}
```

5. Какие сферы применения можно обозначить?

Кто постигает новое, лелея старое,
Тот может быть учителем
Конфуций

Н.Вирт отмечает [6], что «пригодность и успех языков программирования высокого уровня основывается на принципе абстракции, скрытия деталей». Вводит ли ДРАКОН «новый уровень абстракции»? Очевидно, да. И на этом новом уровне схем основывается работа и взаимодействие людей. И, что самое важное, взаимодействие переходит в когнитивную область. В таблице ниже показана иерархия уровней программной системы.

5	Когнитивный уровень	Схема процесса
4	Уровень контента	Псевдо язык
3	Прикладной уровень	Язык оболочки
2	Архитектурный уровень	Средства программирования
1	Системный уровень	Библиотеки и система

Рассмотрим, как будет выглядеть современная распределенная система, например, тренажер для технологических процессов управления. На первом, системном уровне, будут базовые библиотеки доступа к ОС, например, реализация POSIX. Архитектурный уровень определит конструкцию прикладной системы, базу данных, коммуникационное ПО (типа CORBA), основные модули диспетчеризации задач, взаимодействия с клиентскими рабочими местами. На третьем будут языки моделирования, понимаемые оболочкой, для технологического оборудования, теплофизики расходов, давлений и уровней, силовой электрики. Четвертый уровень – промежуточный. На пятый, когнитивный уровень переходят процессы взаимодействия на технологическом языке.

Таковы современные подходы, лет по 20 применяющихся на тренажерах АЭС. Пятый уровень в каждом случае РАЗНЫЙ. Помимо автоматики, есть пакет CMS, описывающий технологические схемы (насосы, арматура, баки) на понятном для теплофизика языке. Есть пакет, описывающий силовую электрику. Это – на когнитивном уровне, для человека. Из схем генерятся данные и куски программ. А вся семантика обработки (численные схемы) находится в модуле на языке оболочки (ФОРТРАНЕ, между прочим), к которому технологические схемы никакого отношения не имеют.

Новые когнитивные технологии не приходят на смену старым. Они нужны не ВМЕСТО старых, а ВМЕСТЕ со старыми. Они вносят новый уровень абстракции. Но это не только новый уровень, но и НОВЫЕ ВОЗМОЖНОСТИ. Новые возможности появятся, когда дать в руки креативным людям оболочки с уровнями с 1 по 4. Особенно в России, ибо у нас наибольшая часть креативных одиночек при, мягко говоря, «не оптимальной» системе организации и управления как производством, так и разработкой ПО. При разработке систем-оболочек уровней 1-4 технический прогресс переместится на 5-й уровень. На этом когнитивном уровне, российские разработки смогут, получить конкурентное преимущество перед остальными. И пусть ДРАКОН будет первым среди этого семейства.

Далее я постараюсь сформулировать структуру и взаимодействие компонентов таких систем. Я буду рассматривать распределенные системы, ибо будущее за ними.

6. Какие нужны связи между языками?

Не важно, какие графические языки будут использованы, важно, чтобы было понимание на уровне прикладных оболочек. Например, языком оболочки может быть XML и построенный на нем протокол SOAP, при этом не важно, какая ОС реализует Web-сервис. Я рассмотрю системы, когда графический язык реализует императивный алгоритм, а языком оболочки является Оберон-2. Разные части разрабатывают разные программные модули, разные типы данных. Но эти данные должны быть совместимы и интегрированы для получения единой системы.

Первое, что нужно определить, что представляет собой схема как единое целое (если она разбита на несколько частей по причине принтера А4 – то целая схема). У нее есть имя, входные параметры, выходные параметры, переменные состояния (сохраняемые) и временные переменные (несохраняемые). Кроме этого, есть процесс управления выполнением схемы: циклическое (с периодом 1 секунда), событийное (по запросу), цикло-событийное. Напомним, что программная реализация схемы в распределенной среде будет вызываться по каким-то причинам, исходящим от других компьютеров.

Второе, важно как передаются параметры между точками распределенной системы при событийном взаимодействии. В сообщение должны входить параметры, передающиеся другим модулям при каких-то событиях.

Третье, алгоритм может часто относится к типу процесса, а не к конкретному экземпляру объекта. В примере ниже на каждое оборудование требуется один экземпляр объекта, содержащего входы/выходы и алгоритм.

На Рис.16 приведены алгоритмы системы АСУТП, левый – с использованием паузы, правый – с пуском и синхронизатором. В современных АСУ – таких алгоритмов тысячи.

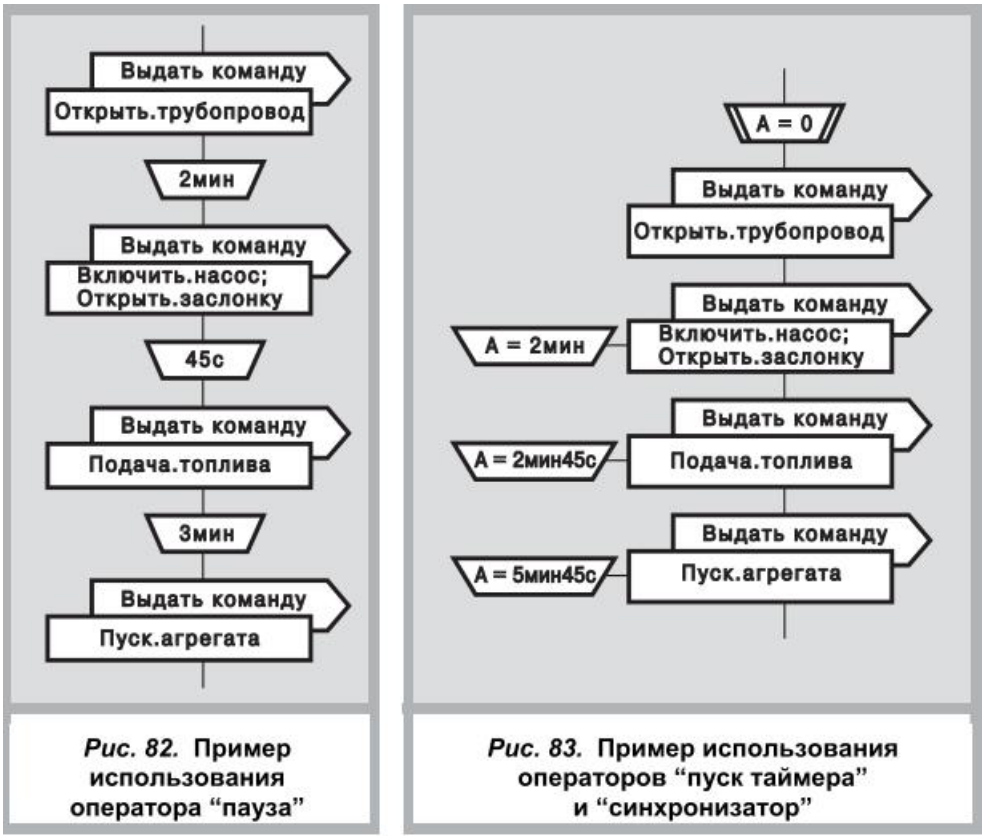


Рис.16 Пауза и таймеры управляют автоматикой

Левый алгоритм выполняется в отдельном потоке (на отдельном контроллере) с остановками на 2 мин и др., причем логика отключения насоса по блокировке и останова алгоритма из другого процесса не представлена. Правый же может работать и по-другому (в реализации ДРАКОН-2 он, видимо, эквивалентен левому), как конечный автомат:

1. Счетчик «2 мин», команда Включить.насос, Переход состояния, Выход;
2. Счетчик «2 мин 45с», команда Подача.топлива, Переход состояния, Выход;
3. Счетчик «5 мин 45с», команда Пуск.Агрегата, Переход состояния, Выход;

Причем в последний случай легко вставить событие «Блокировка» с прерыванием алгоритма и отключением насоса. Поскольку поток возвращает управление, может следом выполняться другой аналогичный алгоритм в том же потоке. Заметьте, программа во втором случае будет отличаться от последовательности: Выполнить, Пауза, Выполнить.

Во-первых, предположим все-таки наличие точки входа по блокировке наряду с точкой входа Старт. Добавлю вход Включен.насос, определяющий его состояние. На выходе – команды Открыть.трубопровод, Включить.насос, Открыть.заслонку, Подача.топлива, Пуск.агрегата. Добавлю выход Номер.иконы для визуализации. Переменная состояния A (счетчик времени).

Во-вторых, входные события: команда Старт и команда Блокировка с параметром Включен.насос. Выходные события: команда на трубопровод (Открыть.трубопровод, Подача.топлива) и На оборудование (Включить.насос, Открыть.заслонку, Пуск.агрегата).

В третьих, экземпляр объекта будет иметь название Algorithm1. На рисунке 17 представлено изображение означенного объекта в соответствии со стандартом МЭК 61499 [7].

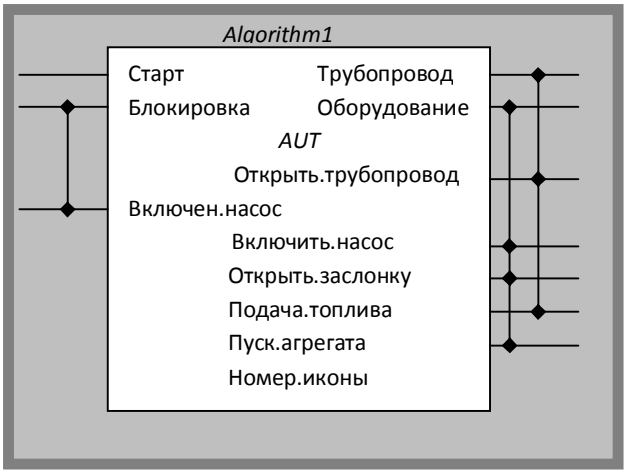


Рис.17 Функциональный блок управления автоматикой

Новый стандарт МЭК 61499 предлагает «новый уровень», интегрирующий имеющиеся алгоритмы программных языков автоматики в единые схемы в виде блоков. Стандарт направлен на встраиваемые и распределенные системы. Это означает системы из нескольких компьютеров (цифровые датчики, контроллеры на разных машинах и цифровые исполнительные механизмы). Предлагаю использовать этом стандарт и для распределенных систем, не относящихся к автоматике.

Тип объекта является либо функцией, либо функциональным блоком, либо процедурой.

Тип объекта	Параметры	Размещение	Оберон-Аналог
Функция	Несколько входов, один выход	Один экземпляр в памяти, не сохраняет ничего	PROCEDURE f(..) : INTEGER
Функциональный блок	Входы, выходы, состояния	Экземпляр объекта	RECORD (FB) .. END
Процедура	Входы, выходы, состояния	Одна процедура, произвольно число выходов, состояний	PROCEDURE f(.., VAR ..)

На рисунке имеется экземпляр объекта типа AUT как функциональный блок. Слева – входы, справа – выходы. Сверху перед уступом – входные события, справа – выходные. Я не стал переименовывать русские идентификаторы.

```
AUT = RECORD (FB)
       handle Handler;
       update: PROCEDURE(f:FB);
       Включен_насос: BOOLEAN;
       Открыть_Трубопровод: BOOLEAN;
       Включить_насос: BOOLEAN;
       Открыть_заслонку: BOOLEAN;
       Подача_топлива: BOOLEAN;
       Пуск_агрегата: BOOLEAN;
       Номер_иконы: INTEGER;
END;
StartMsg = RECORD(Message) END;
BlkMsg = RECORD(Message) Включен_насос: BOOLEAN; END;
PipeMsg = RECORD(Message) Открыть_трубопровод, Подача-топлива: BOOLEAN; END;
EquipmentMsg = RECORD(Message) Включить_насос, Открыть_заслонку, Пуск_агрегата:
BOOLEAN; END;
```

При работе входам присваиваются значения и вызывается циклическая процедура update. С выходов снимаются значения Algorithm1.Подача_топлива. Входное событие с другого (возможно, удаленного) объекта передается как сообщение на функцию-обработчик Handler. Например, событие старт приходит в виде StartMsg, блокировка в виде BlkMsg. Для реальных систем сообщения будут представлять собой последовательность из блоков, включающих в себя сигналы разных типов, чтобы система могла быть расширяема в дальнейшем.

Обработка происходит универсальным, рекомендованным Виртом, способом

```
PROCEDURE handle(f: FB, VAR msg: Message)
	VAR alg: AUT;
BEGIN
	alg = f(AUT);
	IF msg IS StartMsg THEN
		BEGIN alg.Включен_насос = msg(StartMsg).Включен_насос; update(f); END;
	ELSIF msg IS BlkMsg THEN Run_Blk(alg);
END
```

Универсальная процедура handle обрабатывает приходящие извне события, передаваемые адресатам с помощью процедуры emit(StartMsg). Реализация процедуры emit должна учитывать реальные связи между реальными блоками распределенной системы. Сетевое взаимодействие при этом может быть и широковещательным, и нет (broadcast, multicast, unicast).
Аналогично, изменения некоторых выходных сигналов генерят события как emit(PipeMsg). Эти события передаются другим модулям. По приему событий вызывается пересчет модуля в той или иной его части.
Рассмотренный ранее язык FBD позволяет соединять не только логические схемы, но и функциональные блоки. Внутри каждый функциональный блок может быть чем угодно. Но для нас важны 3 языка: логические схемы FBD, ДРАКОН и Оберон (как и любой язык высокого уровня).

Ниже приведена схема соединений из двух алгоритмов распределенной системы управления для насосов и трубопроводов.

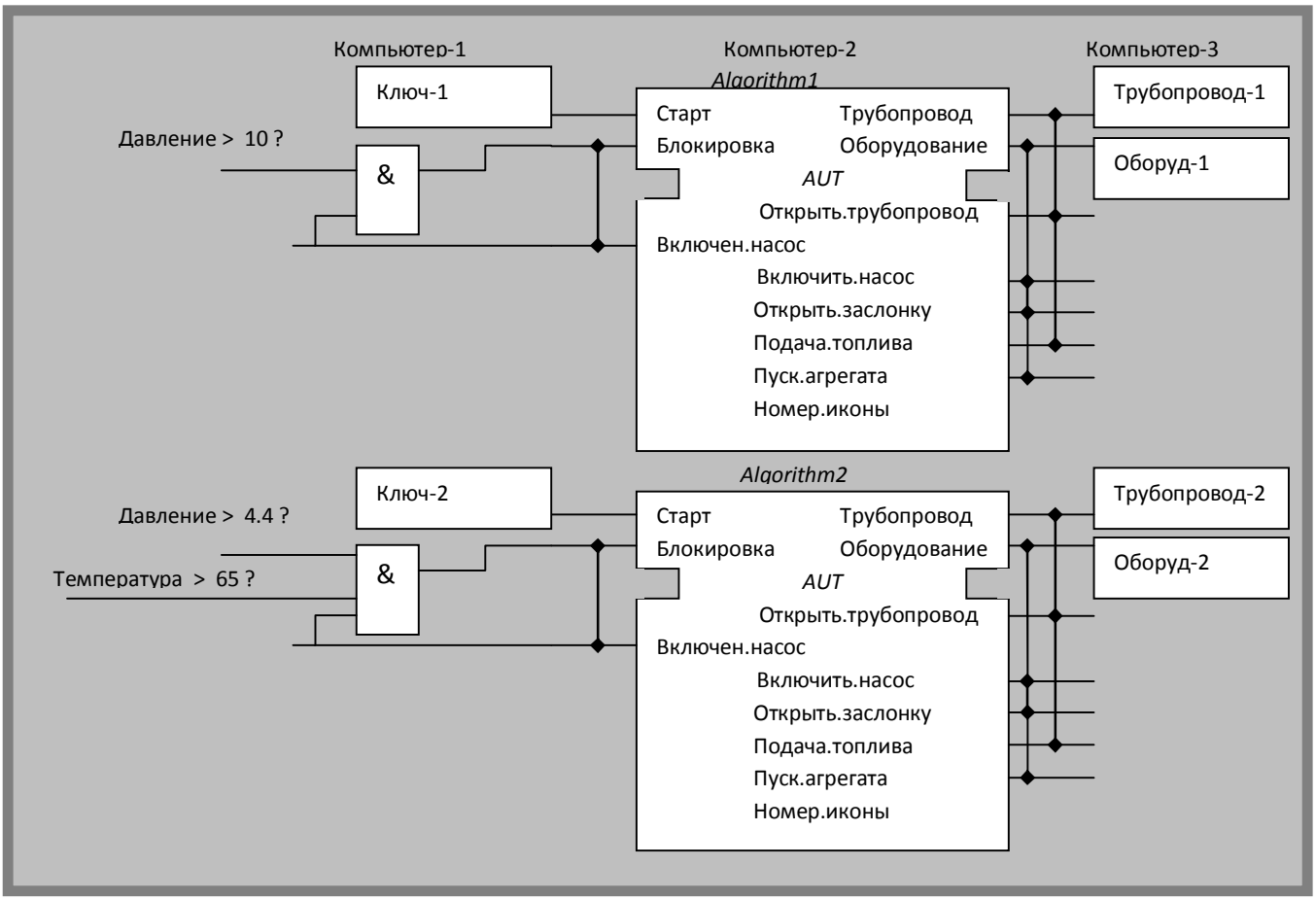


Рис.18 Схема их функциональных блоков

Для данной схемы FBD будет как объединяющий уровень. Для передачи сообщений по сети нужно использовать асинхронные современные пакеты на модели публикации/подписки. Нужен событийный механизм, появление буфера->преобразование в сообщение->вызов Handler(сообщение).

Таким образом, распределенные системы будут состоять из некоторых блоков определенных типов. И будут получать и выдавать данные. И неважно, что внутри – блочный уровень необходим. А попадая «внутрь блока», мы уже имеем дело с алгоритмом – другими данными в другом виде.

7. Какие возможности расширения?

А что будет, если вместо алгоритма управления по той же самой схеме Рис.16 нужно построить модель для тренажера? Тогда для существующей ДРАКОН-схемы семантика поменяется. Например,

1. для каждого шага алгоритма инструктору можно будет вводить ручное переопределение операции (override). Например, отменить ожидание 2 минут.
2. Для некоторых алгоритмов инструктору можно будет вводить моделируемые неисправности (например, невыдачу или ложную выдачу команды).

Это, во-первых, изменит интерфейс функционального блока: добавятся новые сообщения OverrideMsg и MalfunctionMsg, эмитируемые рабочим местом инструктора.

Во-вторых, при одинаковости схемы и псевдо-языка прикладной уровень будет отличаться. Это будет другая программа. В стандарте нет наследования функциональных блоков друг от друга. Зато можно сделать замену реального блока на модельный с 2-мя дополнительными событийными входами и интерфейсом ниже.

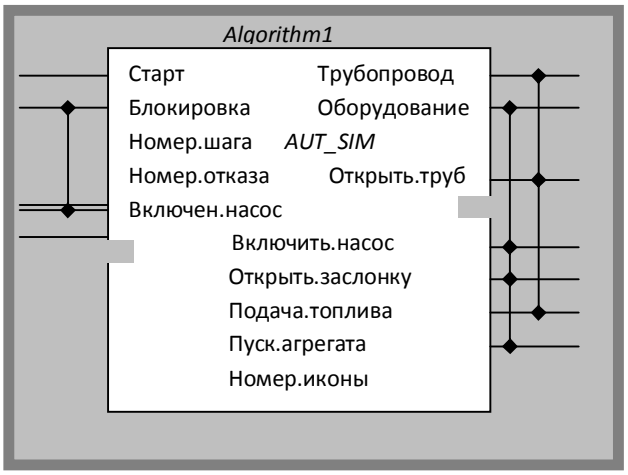


Рис.19 Функциональный блок модели

```
AUT_SIM = RECORD (AUT)
       Номер_шага: INTEGER;
       Номер_отказа: INTEGER;
END;
OverrideMsg = RECORD(Message) Номер_шага: INTEGER; END;
MalfunctionMsg = RECORD(Message) Номер_отказа: INTEGER; END;
```

Блоки останутся? Останутся, но добавятся событийные входы. Алгоритмы останутся теми же? Конечно, теми же. Но поведение системы поменяется от реальной к модельной.

Как видно, демонстрируется хорошая расширяемость даже при существенном изменении характера задачи.

8. Есть ли другие формы представления данных?

А как же! Более простые, чем ДРАКОН-схема, структуры данных ложатся, например, на реляционную модель. Инноватор М.Цукерберг раскрутил свой facebook. Новый когнитивный уровень есть? Есть. 540 млн человек посетило за апрель 2010 по статистике. Facebook также имеет иерархию уровней большой программной системы.

5	Когнитивный уровень	Схема процесса	Дружественные Веб-странички
4	Уровень контента	Псевдо язык	Facebook FBML, содержание базы данных
3	Прикладной уровень	Язык оболочки	Реляционная модель данных, скрипты уровня БД и Веб
2	Архитектурный уровень	Средства программирования	Браузеры, Веб-серверы, СУБД
1	Системный уровень	Библиотеки и система	Клиентская и серверные ОС

В данном примере новый когнитивный уровень легко ложится на существующие программные средства. Да, новое имеет место. Однако структура данных старая добрая реляционная. И любой другой, придумав, если повезет, новый когнитивный уровень на старой реляционной модели, может полностью использовать старые программные средства.

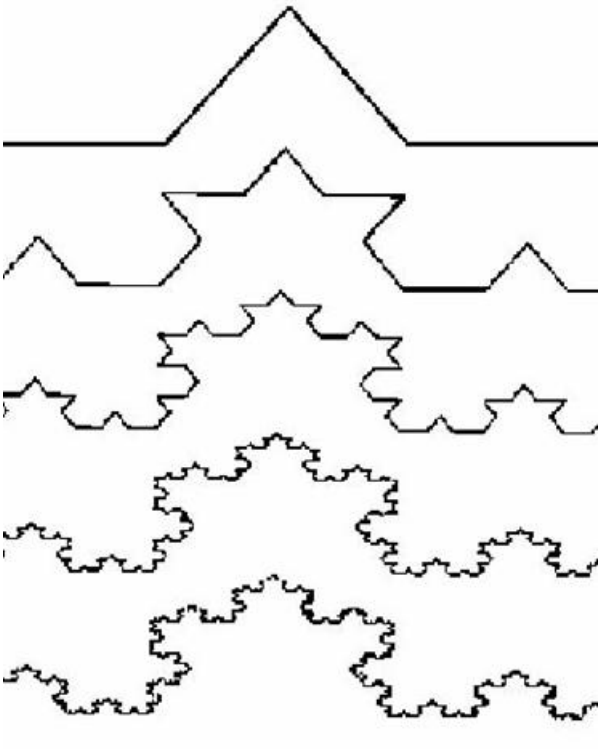
Хотя поле деятельности реляционных систем протоптано, однако и там случаются прорывы!

Выводы

Как обосновывается в [8], принцип абсолютизации текста является ошибочным и вредным. В свою очередь, принцип абсолютизации графики также неправилен. Поэтому предлагается гармоничное сочетание с широчайшими возможностями развития.

1. Язык ДРАКОН хорош тем, что вводит новый когнитивный уровень в многоярусной системе программного взаимодействия.
2. Неизвестно, что Вам из графической схемы понадобится: программа или данные. Поэтому используйте псевдо-язык, из которого можно получить и то, и другое.
3. Видно целое семейство когнитивных языков для разных задач, которые со временем оформятся в стандарты.
4. Как набор для решения сложных, не реляционных задач предлагается 3 языка: ДРАКОН, Логические Схемы(FBD) и текстовый язык высокого уровня. Для реальной большой системы нужно делать еще и оболочки.
5. Объединение ДРАКОН-схемы функциональными блоками по сути делает объектно-ориентированную систему из алгоритмов. (Пароджанов писал в [8], что «использование принципов ООП … исключает возможность целостного восприятия сложной алгоритмической картины». Не исключает, нужно правильно сочетать.)
6. Только графические языки являются когнитивными, все остальные уровни ПО предоставят для них площадку для роста.
7. Дальнейший прогресс информационных систем, по-видимому, будет связан с новыми когнитивными уровнями.

И, наконец, эволюция по каждому из уровней лучше всего иллюстрируется на фрактальной кривой ниже. Вектор развития проходит от простого ко все более изощренному!



Системный уровень	Компиляторы и базовая ОС
Архитектурный уровень	Фундамент и базовые системы: СУБД, SCADA, Браузеры
Прикладной уровень	Прикладное ПО, модели данных
Уровень контента	Языки представления данных и программ в понятной для компьютера форме
Когнитивный уровень	Ясное воплощение наших мыслей.

Список литературы

1. В.Д.Паронджанов. Как улучшить работу ума? М.: 2001
2. И.В.Петров. Программируемые контроллеры. Стандартные языки и инструменты. М.: 2003
3. В.Д.Паронджанов. Язык Дракон. Краткое описание.
4. Ф.Хейес-Рот, Д.Уотерман, Д.Ленат. Построение экспертных систем. М: 1987
5. Р.П.Богатырев. Нужна ли России своя операционная система? http://www.osp.ru/pcworld/2007/08/4388326/
6. N.Wirth. Programming in Oberon. A derivative programming in Modula-2 (2004)
7. IEC 61499. Function Blocks for Embedded and Distributed Control Systems Design
8. В.Д.Паронджанов. Дружелюбные алгоритмы, понятные каждому. М: 2010