Как запустить Дракона

Дмитрий Дагаев

Оглавление

Предисловие

«Мне никогда не постичь, как дракон,
Оседлав ветер и облака, взмывает в небо.
Сегодня я увидел Лао-цзы –
Воистину он подобен дракону!»
Конфуций

Приближаясь к определенному рубежу, люди часто склонны думать о том, что останется следующим поколениям после их трудовых свершений. Вот что думал я, мысленно обращаясь к потомкам:

- Где наши бывшие передовыми технологии в области электроники, приборостроения?
  - Все профукали японцам!
- Где наши бывшие передовыми разработки в области программных систем?
  - Все профукали американцам!
- Где нанотехнологии?
  - Чубайс их знает, увидите Вы нанотехнологии или нет!
- Что Вам от нас, грешных осталось?
  - Ничего!

Что делать, если Вам не досталось ничего? Из ничего сделать нечто! И для этого появились когнитивные технологии! И ничего ведь не надо!

Для создания когнитивных технологий

вам не нужно ни силикона, ни пластика,

лишь чистое небо над головой,

покой в душе

и твердую землю под ногами

Мы вступили уже в постиндустриальную эпоху. Тут столько систем, неплохих и разных, что любая мысль уже цепляет за собой реализацию какого-то софта, какого-то товарища или группы товарищей, этой проблемой занимавшейся. Информации столько, что переварить имеющуюся даже в узкой области затруднительно. И процесс этого метаболизма занимает месяцы и годы.

А переварivши, понимаешь, что опять было что-то несъедобное. И не греет душу-то. Спать только хочется от этого изобилия.

Так изменим мир к лучшему!

Дагаев Дмитрий Викторович dvdagaev@mail.ru .

Введение

«Митрофанушка, друг мой!
Если ученье так опасно для твоей головушки,
Так по мне перестань»
Недоросль

Здесь Вы увидите концепцию построения систем с использованием визуального языка ДРАКОН[1-4]. В «Алаверды Дракону»[5] я рассматривал аспекты того, как Дракона «поставить на землю». Ибо, перефразируя Чичикова, визуальный язык на бумаге – это «журавль в небе». А людям нужны конкретные реализации, т.е. «синицу в руку». И эти реализации существуют, правда, на языке блок-схем. И задач на блок-схемах большое количество: АСУТП, моделирование, экспертные системы. И алгоритмы там совсем разные. И еще стандарты есть, и мультиязыковые среды. Иногда витающим в облаках разработчикам весьма полезно понимание, что они не одни на этом празднике жизни, что есть другие, которые еще в чем-то вперед продвинулись. Иногда нужно поставить на землю.

Данный материал - о другом. Я попытаюсь на пройденных ранее своих разработках сформулировать видение Дракон-систем. Я хочу написать, как собрать эти новые Дракон-системы и «запустить в небо». Сделать концепцию такой инженерной конструкции, которая заработает и полетит. Поэтому ниже будут описаны некоторые системы в проекте. Концепции в предлагаемом автором подходе. При этом я не имел никакого отношения к разработчикам Дракона, недавно услышал о нем. Все, что раннее было сделано, реализовывалось на языке блок-схем. Поэтому я с пониманием отношусь к тому, что Дракон-сообщество может не принять мои подходы. Равно как и мировое сообщество может уйти вперед и никогда не принять подходы Дракона, оставив блок-схемы как язык для будущих поколений.

С этого момента настоятельно рекомендую прекратить чтение тем, кто считает, что:

- полностью завяз в плену своих программных продуктов и не готов ни под каким предлогом отказаться от имеющихся представлений;
- предложенного не может быть сделано, потому что не может быть сделано никогда. Или хотя бы в ближайшие 30 лет;
- не стоит этим заниматься. Разумеется, Вам не стоит.
- я популяризирую стандарты МЭК 61131-3. Нет, упаси боже. И так полно систем на них, которые будут покупать. Возможно, вместо Ваших.

Я пишу для тех, кто решится что-то делать. Если хотя бы один из читателей возьмет описанное за основу и создаст реальную большую систему, я пойму, что все было не напрасно.

1. Когнитивные технологии

1.1. Сделайте сначала фундамент грядущего

«Ты был главным инженером,
ты был главным консультантом,
ты был главным конструктором –
забудь, кем ты был!
Всё, ты теперь навоз для детей.»
Жванецкий

В документе «Два пернатых в одной берлоге не уживутся?» [6] я предложил фундамент для концепции когнитивных технологий: графический пакет и язык скрипта. Язык скрипта в свою очередь разбивается на скрипт для программ и псевдоязык. Здесь и далее я буду ссылаться на них, как на *уже разработанные*. Будут приведены на рисунках экранные слепки графических программ. Неважно, каковы детали графического пакета будут, важно, что его *эргономичность и функциональность будут, как минимум, не хуже* приведенных рисунков. Будут приведены фрагменты скриптов в синтаксисе tcl. Хотя псевдоязык может быть в другом виде, например в XML. Неважно, какой будет язык. Важно, что его *возможности и выразительность будут, как минимум, не хуже* приведенных во фрагментах.

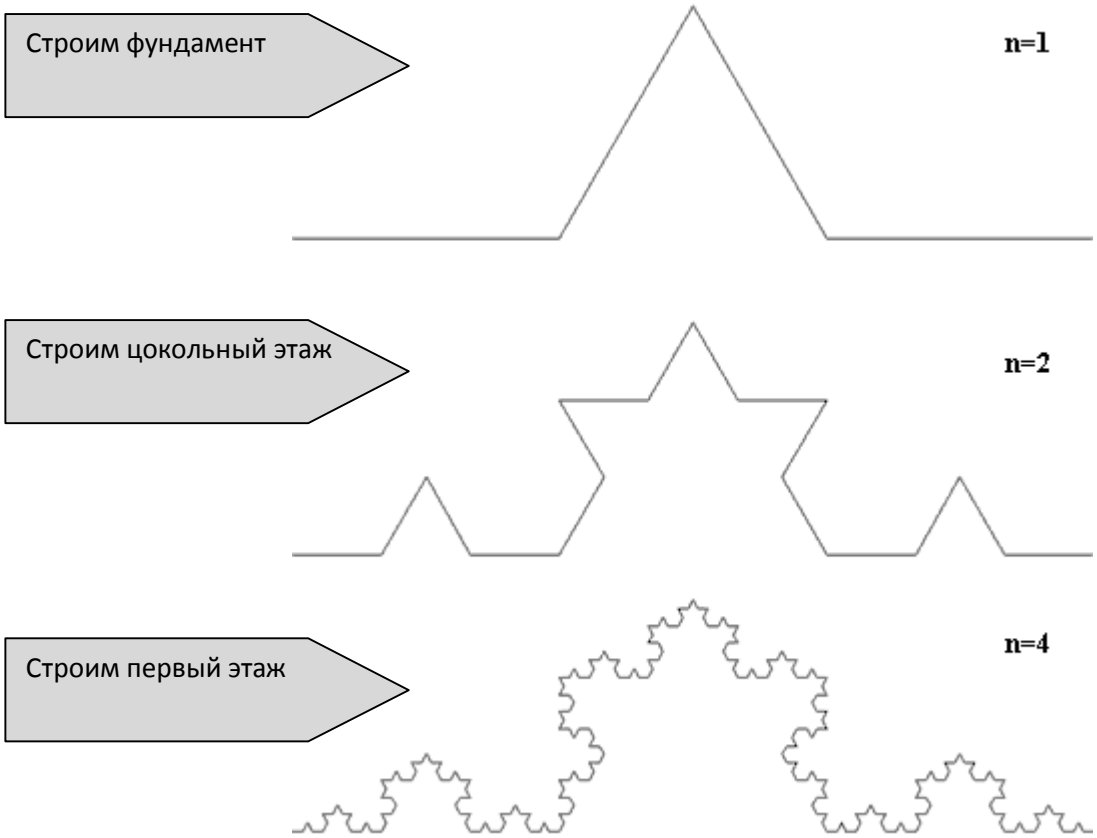
Поэтому, *глядя на рисунки по тексту, считайте их макетами*. Которые нужно было построить и сломать. И вычистить ошибки, чтобы не повторять впоследствии. И строить уже фундамент, учитывая эти ошибки. Чтобы двигаться дальше.

Когнитивные технологии – это то, что дальше!

Дракон, установленный на построенную конструкцию, вознесся на достойное место. Он ждет попутного ветра!

А как вы себе представляете зарождение новых технологий? На замену старым? Переоценка ценностей? Весь мир насилья мы разрушим до основанья, а затем … У нас же постиндустриальное общество с ограниченным и весьма цивилизованным пространством. Куда еще втиснуть новое то? Не на то же место?

Фрактальные линии дают замечательную графическую иллюстрацию, как это происходит. Первая кривая определяет базовую форму и занимает пространство. Вторая кривая накладывается на первую в том же пространстве. Третья на вторую.

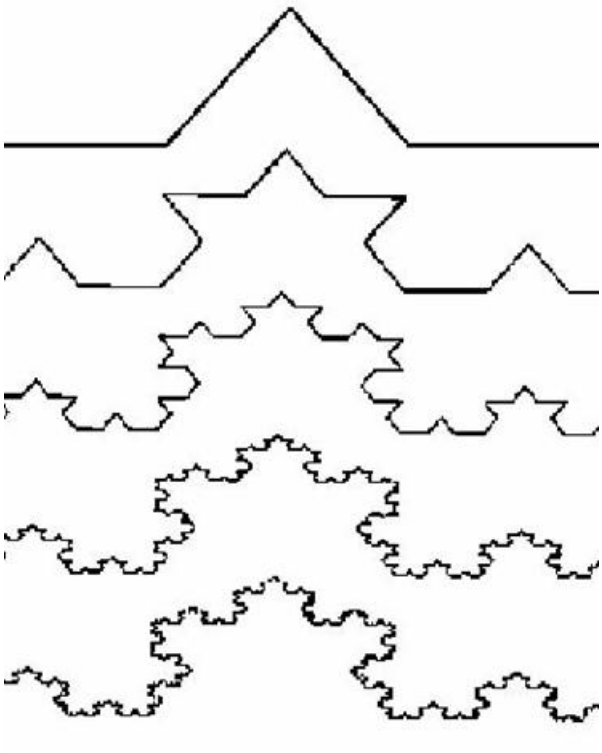


Каждый уровень необходим. Вы можете строить новые, можете брать существующие.

Каждый новый уровень линейно независим от предыдущих. Они не повторяются, они дополняют друг друга.

Допустим новые уровни, вносящие уникальные свойства, базируются на старых фундаментах. И графически очень хорошо видно, что нового места не потребовалось. Вы создадите принципиально новое сами, не выходя из дома. Без новых компьютеров. Без помощи группы товарищей из Apple и Microsoft.

Я предлагаю деление систем с когнитивным верхом на пять уровней.



1	Системный уровень	Компиляторы, интерпретаторы, среда разработки и базовая ОС
2	Архитектурный уровень	Фундамент и базовые системы: Графические оболочки, Браузеры
3	Прикладной уровень	Прикладное ПО, модели данных
4	Уровень контента	Псевдо язык. Представление данных и программ в понятной для компьютера форме
5	Когнитивный уровень	Ясное воплощение наших мыслей.

Системное программное обеспечение и интерпретатор скрипта поместим на первый уровень. Графические языки поместим на пятый, когнитивный уровень. Псевдоязык, к ним относящийся, на четвертый. Второй уровень будет представлять собой программное обеспечение графической оболочки, это – цокольный этаж. Далее я буду показывать, что для большого класса систем уровни 1 и 2 останутся неизменными, в то время, как уровни 3-5 будут варьироваться в зависимости от задачи. При этом, если используем Дракон как визуальный язык, 4 и 5 уровни будут от Дракона. Значит, для разработки реальных систем понадобится делать только 3 уровень.

1.2. GoogleMap как когнитивная 5-ти уровневая система

Можно бесконечно глядеть
На огонь, на воду
И на то, как другие работают.

Пароджанов в пишет: «язык ДРАКОН можно охарактеризовать как первый в истории эргономический суперъязык». Может, оно и так для алгоритмов. Но не все же людям смотреть на алгоритмы… Если они смотрят на данные, что можно найти эргономического? Я попытался найти живую приличную систему с когнитивным уровнем, и мне это удалось. Да, GoogleMap мне нравится. На эти данные смотреть приятно. Структурно это выглядит так.

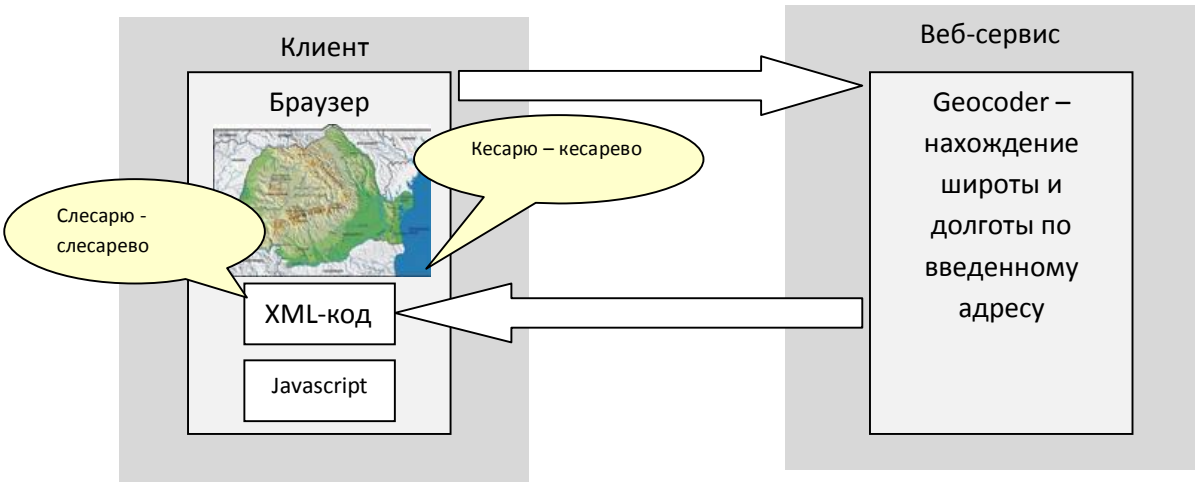


Рис.1 Упрощенная структура GoogleMaps

В системе GoogleMaps географическая информация запрашивается на сервере и показывается в браузере. Там посылается HTTP запрос на сервис http://maps.google.com/maps/geo с параметрами адрес,ключ,формат и сервис возвращает данные чаще в виде XML[7]. Формирование запросов и обработка данных осуществляется в браузере на языке JavaScript, который работает в браузере.

Получилась как раз 5-уровневая схема. Во-первых, Ваш браузер является хорошим примером графической оболочки в свете моей концепции в [6]. В браузере можно находиться постоянно, другие программы подгрузятся по мере необходимости. Во-вторых, что когнитивному уровню – красивой карте – однозначно соответствует четвертый уровень данных в виде XML, обрабатываемых компьютером. Кесарю – кесарево, а слесарю – слесарево! Заметим, что XML может представить не только данные, но и алгоритмы тоже.

Когнитивные системы уже появились!

1	Системный уровень	Операционные системы на клиенте и сервисе
2	Архитектурный уровень	Клиент: Браузер; Сервис: СУБД, ПО Веб-сервиса
3	Прикладной уровень	Клиент: Скрипты на Javascript; Сервис: задача geocoder
4	Уровень контента	Данные в виде XML
5	Когнитивный уровень	Красивые карты

2. Системы непроцедурные

Вас обманули! Вам дали гораздо лучший мех.
Это шанхайские барсы.
Остап Бендер

Если Вас поймают на улице, пристанут с ножом к горлу и потребуют признать, что Дракон-схема – это только программа, и она никогда не была и не станет структурой данных, соглашайтесь. Ибо Галилею было хуже, но он потом успел сказать «и все-таки она вертится».

И все-таки Дракон-схема может быть структурой данных.

Если Вы с этим не согласны, Вам повезло: Дракон-схема гораздо лучше, чем Вы о ней думали. Вы хотя бы можете видеть и лелеять ее структуру.

Если еще сомневаетесь, попробуйте «выполнить ее» наоборот, снизу вверх. Ничего не получится, если вы ее отобразили в код на процедурном языке. А отобразите в данные в виде графа, все получится. Можете попробовать.

Ниже даны примеры систем непроцедурных. Где схема суть данные. И обрабатываются они интерпретатором.

2.1. Человек работает, алгоритм за ним наблюдает!

Большой Брат смотрит на тебя!
Дж.Оруэлл

Я здесь остановлюсь на системе 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. Назначение системы по сути дела сводилось как к отслеживанию инструкций, так и к отслеживанию и фиксации изменений состояния оборудования вследствие действий оператора. Оператор управляет процессом в случае аварии, а инженер по безопасности фиксировал его действия. СИДОР расшифровывается как «Симптомно-ориентированные инструкции для оператора реактора». Но был еще один, «сакральный» смысл названия системы, ставившей оператора в положение обвиняемого, т.е. *сидоровой козы*. Система была предназначена и для АЭС, и для тренажера. Но была реализована только для тренажера (рис.2). Система СИДОР 1 раз в секунду получала данные о значениях переменных процесса тренажера через сервер обмена данными. Использовался полномасштабный тренажер, все пульты реальные!

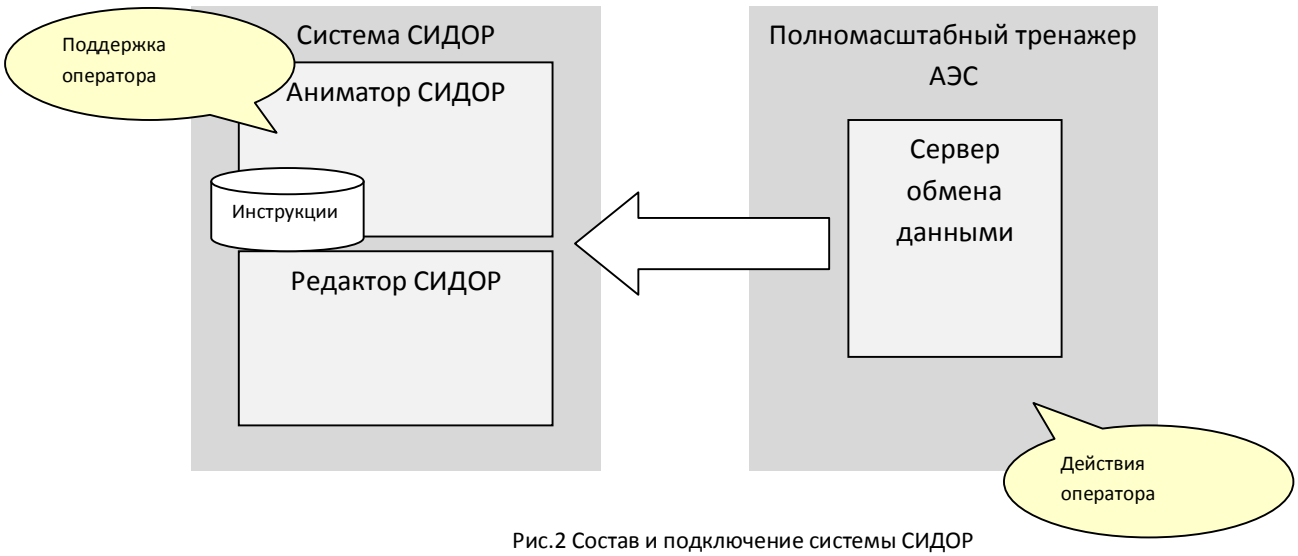


Рис.2 Состав и подключение системы СИДОР

При возникновении аварийной ситуации, смоделированной на тренажере, оператор начинает производить действия, согласно инструкциям. Формально берет в руки лист бумаги с инструкцией и производит. Однако, может и пользоваться системой поддержки (но не должен, система еще не штатная для АЭС). Заглядывает на другой компьютер, а там – Дракон-схема (как я и говорил, все реальные блок-схемы переводим в проектные Дракон-схемы). Которая предлагает и контролирует.

Фрагмент двух из 29 шагов симптомно-ориентированной инструкции приведен на Рис 3.

[illegible]	[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.3 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.

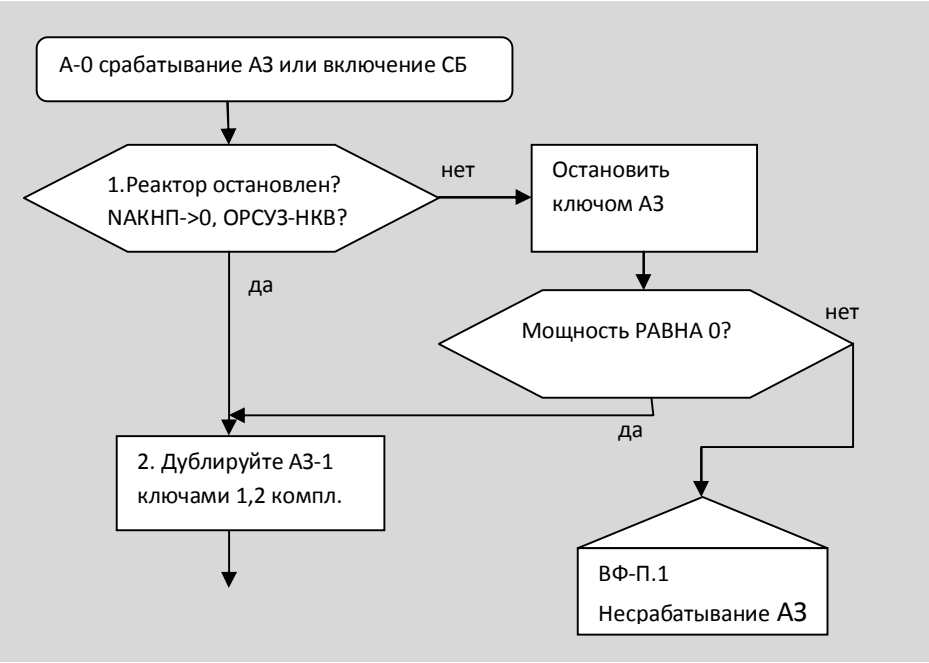


Рис.4 ДРАКОН-схема фрагмента аварийной процедуры А-0

На Рис.4 приведен алгоритм инструкции. А вот теперь займемся алгоритмом выполнения системы поддержки оператора. Как только аварийная защита сработала, СИДОР переходит к инструкции А-0 шагу 1 (Рис. 5). Он загружает правило для шага 1 (Реактор остановлен? NAKHП->0, ОРСУЗ-НКВ). Это – иконка «ВОПРОС» в Драконовском понимании. И, циклически, раз в секунду проверяет это правило, подсвечивая результат в нижней части экрана. До тех пор, пока не будет нажата кнопка «да» или «нет».

Алгоритм работы системы поддержки полностью отличен от алгоритма инструкции!

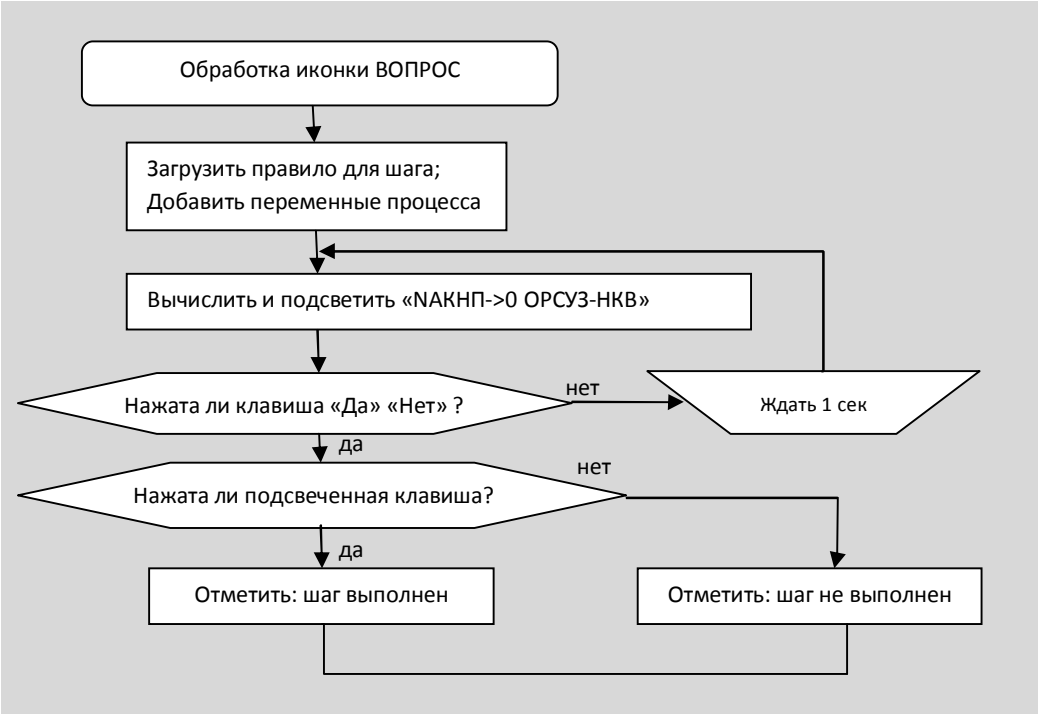


Рис.5 ДРАКОН-схема системы поддержки для иконки ВОПРОС

Также правило содержит гипертекст для объяснения, появляющийся по кнопке «Почему» (Рис.6).

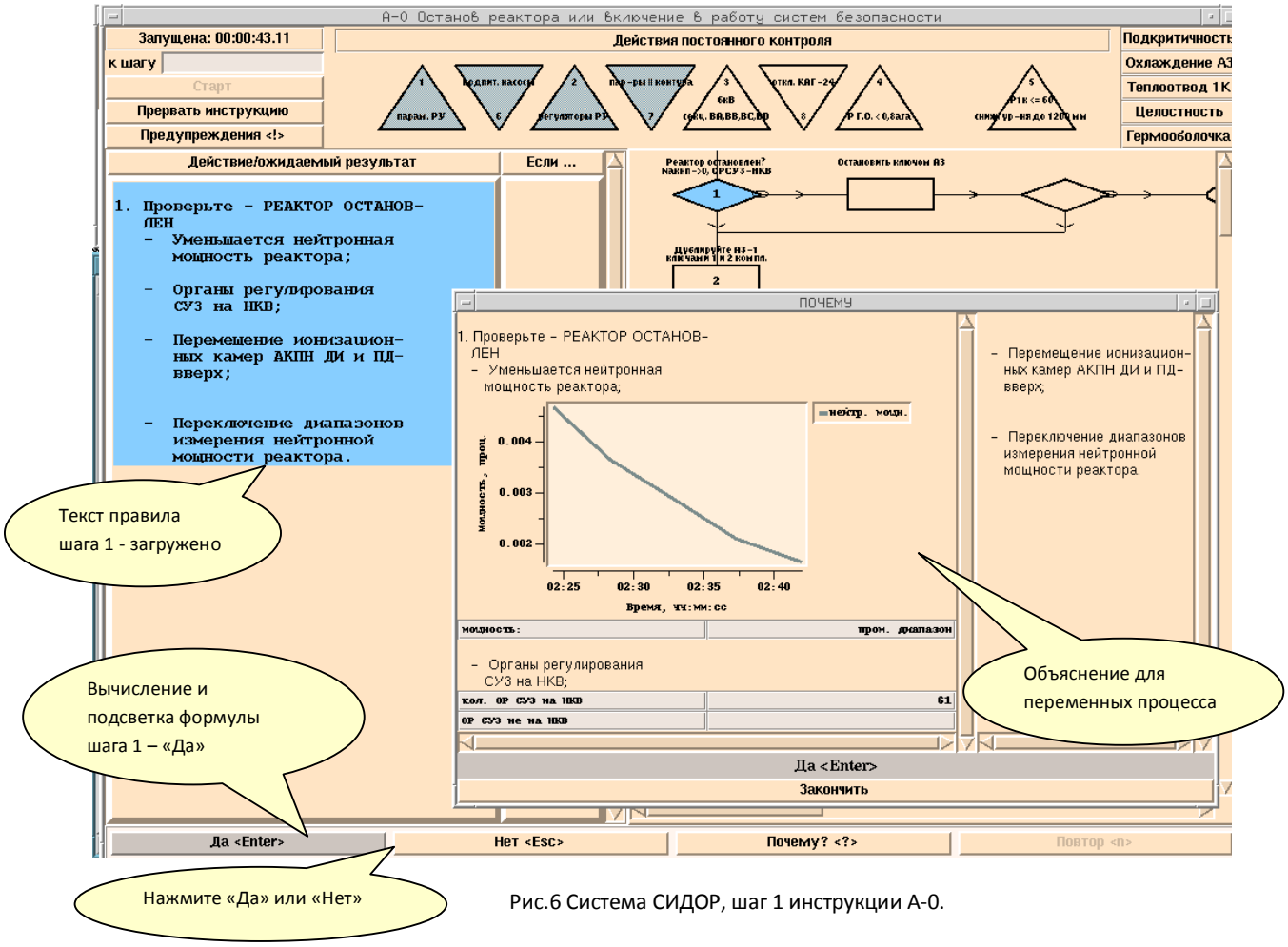


Рис.6 Система СИДОР, шаг 1 инструкции А-0.

При нажатии на вопрос «Почему?» для проверки уменьшения мощности выдается текущий график мощности. Механизм объяснений содержит график, диапазон мощности в виде текста, число сработавших органов управления.

Вычисление осуществляется на основании переменных процесса. Для нашего шага это – мощность, диапазон, число органов управления. Список переменных динамически загружается для каждого конкретного шага. При этом для мощности берется массив значений за интервал, иначе как решить, что «уменьшается»? Естественно, подсказка «Почему» выдает график и таблицы по тем же переменным процесса, что и система вычислений.

А что за информация содержалась внутри каждой иконки? В реализованной системе был псевдоязык.

```
ВОПРОС A-0//1 -main 1                  \
     -orient left        \
     -upflags {1 0 1 0}  \
     -flags {3 15 7 0}   \
     -dnflags {1 1 1 0}  \
     -htsplit 4t         \
     -subscr1 "Реактор остановлен?"  \
     -subscr2 "Накнп->0, ОРСУЗ-НКВ"
A-0//1 htext {
Проверьте - РЕАКТОР ОСТАНОВ-
ЛЕН
-  Уменьшается нейтронная
   мощность реактора;
   %%
Graph $v(0) -height 200
GraphCfg yaxis -title "Мощность, проц."
GraphCfg element $v(0) -label "нейтр. мощн."
%%
%%
Table tmp6__1_RANGE -left "мощность:"
%%

-  Органы регулирования
   СУЗ на НКВ;
%%
Table {tmp6__1_NUM tmp6__1_MFLIST} -left {"кол. ОР СУЗ на НКВ" \
    "ОР СУЗ не на НКВ"}
%%

-  Перемещение ионизацион-
   ных камер АКПН ДИ и ПД-
   вверх;


-  Переключение диапазонов
   измерения нейтронной
   мощности реактора.}
A-0//1 dp yu0049
A-0//1 dc yu0049 -during 12 -speedzone 0
for {set i 1; set tmp6s1rd ""} {$i <= 61} {incr i} {
    lappend tmp6s1rd rdvalue($i)
}
eval [concat A-0//1 dp $tmp6s1rd]
A-0//1 check proc6__1
A-0//1 goto A-0//1.1 A-0//2
```


Рис.7 – содержание внутри иконки

Псевдо язык реализован в виде скрипта. Скрипт задает поля структур (это – уровень 4). Например, `A-0//1 check proc6__1` задает для A-0 шага 1 поле check в виде процедуры «proc6__1», разработанной на том же языке скрипта.

Для иконки заданы были поля: subscr1 – текст внутри иконки, htext - гипертекст, dp – переменные процесса, dc – буфер для графиков, check – процедура вычисления, goto – к следующему шагу.

Для данной задачи в редакторе для каждой иконки надо заполнять целую структуру. И еще организовать проверку введенных данных.

Поле	Содержание	Проверка
subscr1,2	`Реактор остановлен?`	
htext	`Проверьте - РЕАКТОР ОСТАНОВ-ЛЕН - Уменьшается нейтронная`	Гипертекст надо отобразить при редактировании
dp	`yu0049 …`	Должна соответствовать базе данных
dc	`yu0049`	Одна из dp
check	`proc6__1`	Выражение или существующая функция

Еще раз обратим внимание на графическую программу. Она может занимать весь экран и предоставлять пользователю всю требуемую ему информацию (рис.8).

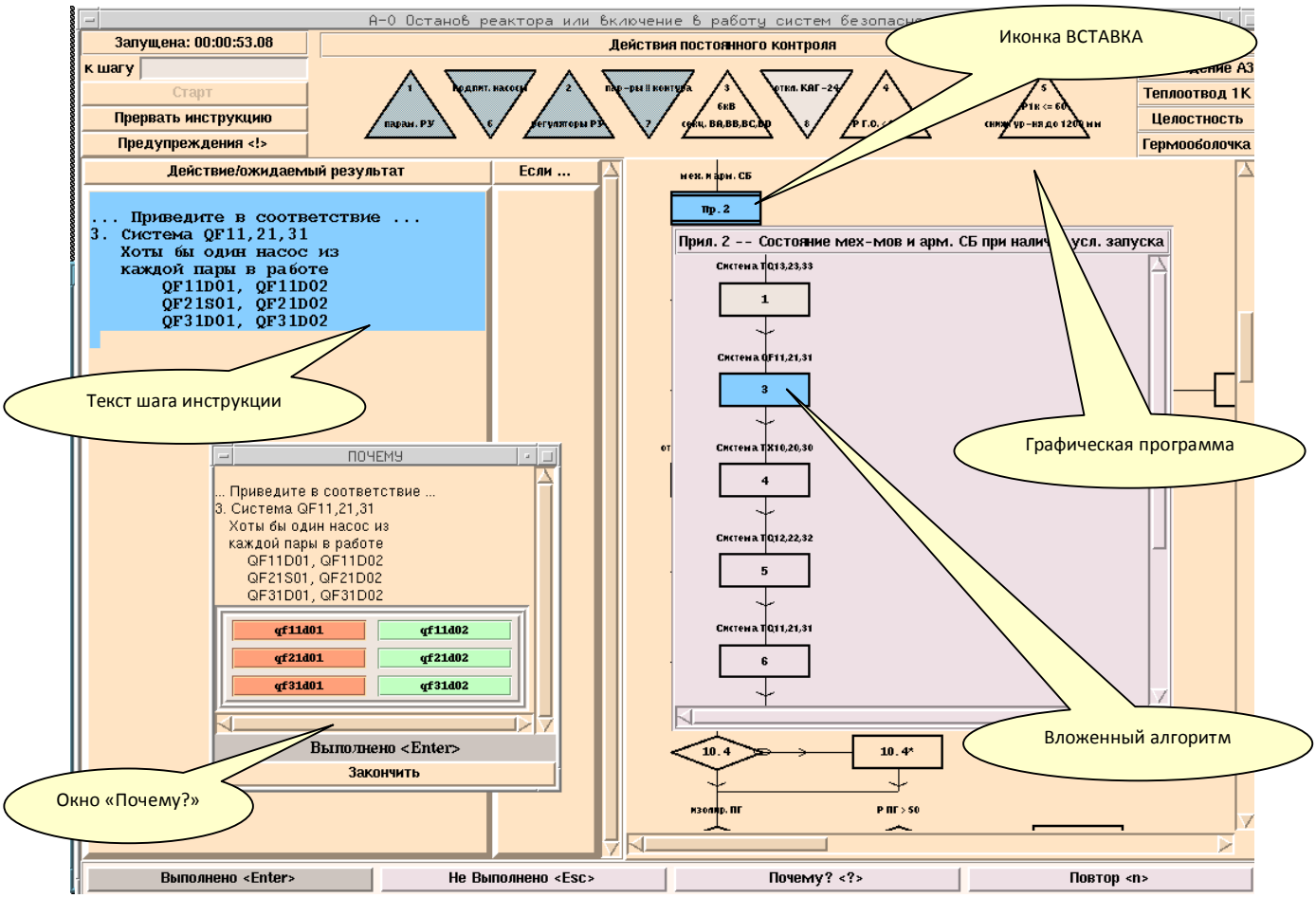


Рис.8 Выполнение вложенной инструкции

Справа на рисунке реализована анимация вложенных циклов из иконки ВСТАВКА! А еще есть скроллинг, ибо станиц там 29. (И не объяснишь Заказчику, что не эргономично).

Слева на рисунке выдается текст текущего шага инструкции. Оператор читает и выполняет. Компьютер контролирует каждый шаг и выдает решение: ВЫПОЛНЕНО/НЕ ВЫПОЛНЕНО. Оператор запрашивает объяснение от компьютера: окно «Почему?» выводит объяснение для переменных процесса.

Помимо этого, в верхней части отображаются «Действия постоянного контроля». Они проверяются постоянно, вне зависимости от выполняемого шага. И все на одном экране!

Начиная с этого момента я буду отмечать некоторые принципы, являющиеся необходимыми при построении эргономичных систем.

> Принцип 1. Предоставьте всю необходимую информацию в поле зрения пользователя.

Этот принцип не нов. Описанный Пароннджановым в [4] опыт над обезьяной, палкой и бананом приводил к выводу: вероятность успешного решения увеличивается, если обезьяна в состоянии одновременно видеть плод и палку. Я предлагаю *принцип как программный способ реализации одновременного предъявления предметов*.

А теперь попробуем инструкцию отредактировать. На рисунке 9 приведен вид задачи редактирования. Загружена уже другая инструкция А-1, выбран шаг 3 а.

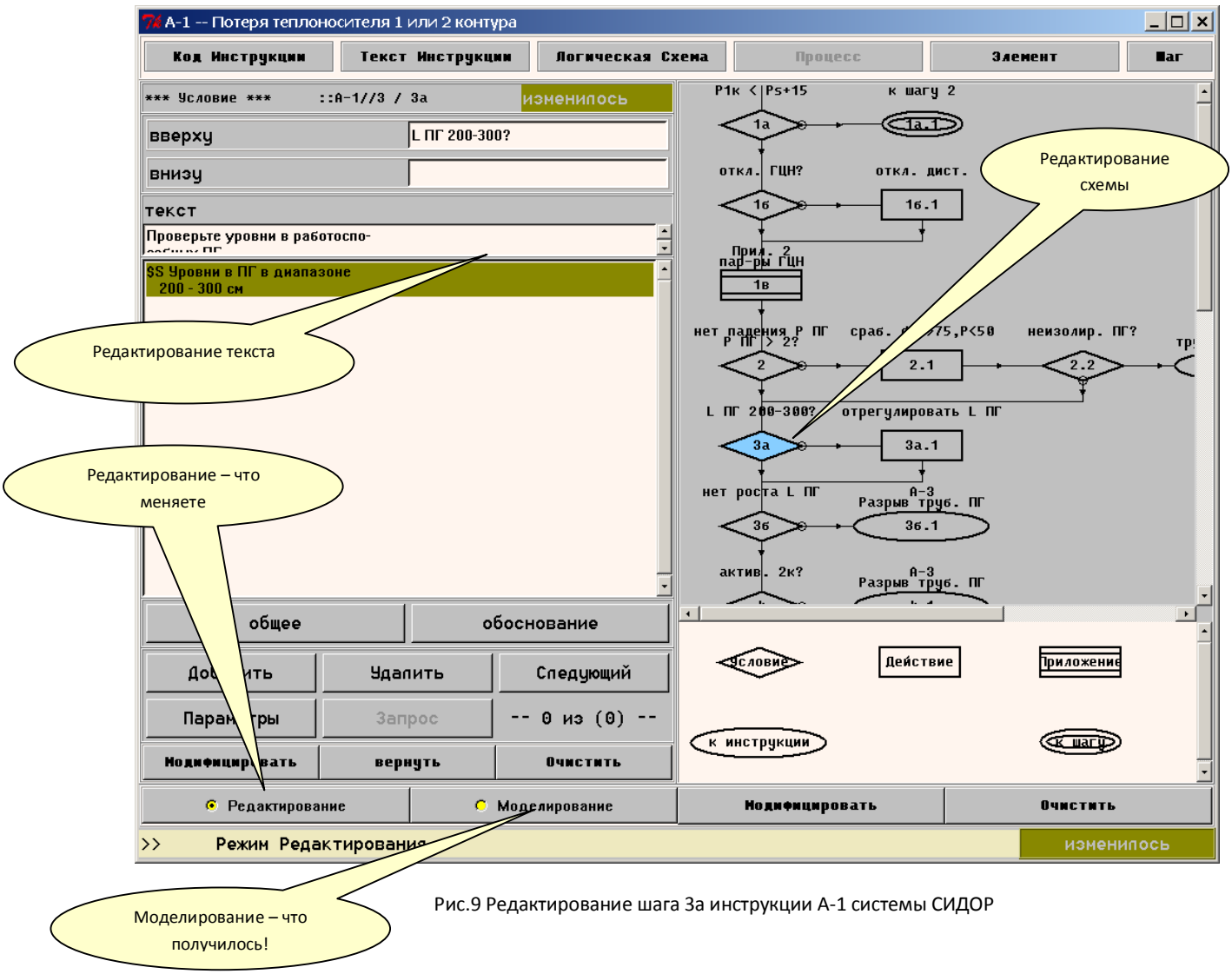


Рис.9 Редактирование шага 3а инструкции А-1 системы СИДОР

Расположение редактора аналогично стремится использовать все пространство экрана. Слева – текст, справа – схема. Вы видите и схему, и текст. И тут же можете вводить и править. Режим редактирования, это Вы правите, режим моделирования, - смотрите поправленное.

Что мы редактируем?

Для системы поддержки мы имеем Дракон-схему инструкции как данные на псевдоязыке.

Для иконок системы поддержки мы имеем сложную структуру полей с необходимостью проверки ввода.

Для графических средств системы поддержки мы имеем уникальный набор дозагружаемых модулей для данной прикладной задачи.

Скрипт правится в отдельном окошке (Рис.10). Поле cmd составляет программную часть гипертекста, поля dpc и icheck соответствуют полям dp и check.

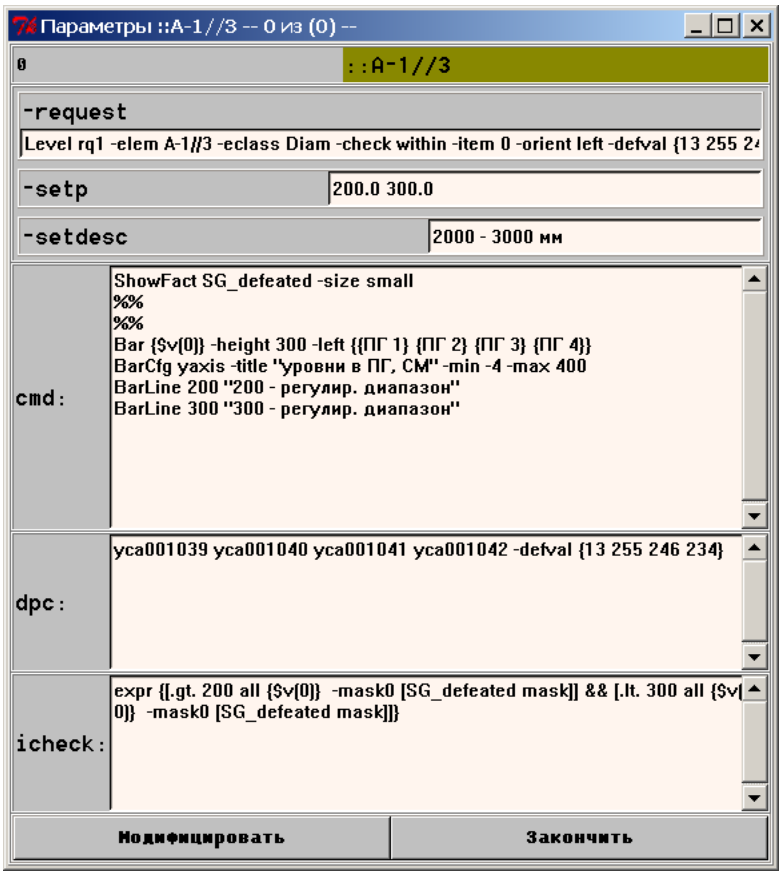


Рис.10 Редактирование скрипта для шага 3а

Получается, для определенной задачи нужен определенный скрипт для редактора. И определенный встраиваемый в графическую систему программный модуль. Как это делается в веб-браузере. Более того, поскольку редактор должен проверять введенные данные, например, на соответствие базе данных, то такое же встраивание нужно и по отношению к редактору.

Теперь хочется вернуться к языку представления данных внутри иконки. Тот же самый скрипт. Сначала создается экземпляр объекта A-0//1:

```
ВОПРОС A-0//1 -subscr1 ..
```

Параметры инициализации задаются путем –опция «значение». Затем просто заполняются его поля.

```
A-0//1 htext {…}
A-0//1 dp yu0049
```

И он готов к использованию. Тем же языком, загруженным в редактор.

> Мне уже высказывали возражения, что «объектно-ориентированного Дракона нет», поэтому поясняю сказанное. Иконки рассматриваются как структуры с полями для каждой задачи. Иконки наследуются от стандартных иконок «ВОПРОС», «ДЕЙСТВИЕ». Таким образом, мы *имеем расширение типов*. Другие свойства ООП я пока не обозначаю. Будем такие расширенные новыми данными иконки называть объектами или структурами.

Переменные процесса, такие, как yu0049, представляют собой динамически изменяемый список. Этот список периодически обновляется значениями полномасштабного тренажера через сервер обмена данными.

Если можно из скрипта построить блок-схему, то из Дракон-схемы можно построить скрипт!

Помимо всего прочего, из псевдоязыка скрипта понадобилось, не загружая схему, генерить HTML-страницу(рис.11). Из скрипта Вы это *делается легко*.

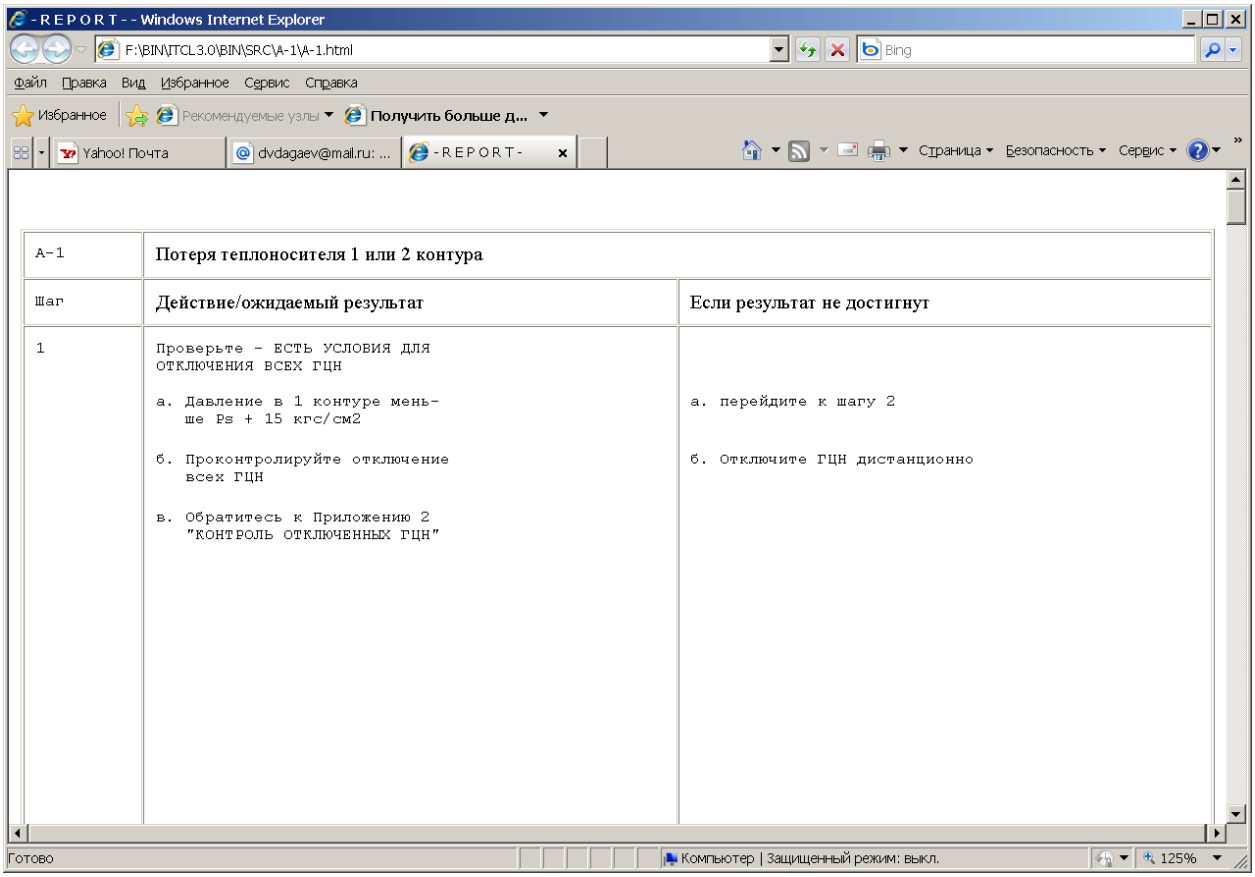


Рис.11 Инструкции в виде HTML

2.2. Экспертная система диагностики

Мы не хотим винить как Компьютер
за неспособность к пониманию прекрасного,
Так и Человека за проигрыш гонки против аэроплана.
Алан Тьюринг

Я уже критиковал учебную экспертную систему по химии за 10 класс[5], т.к. Дракон-схема к ней не подходит.

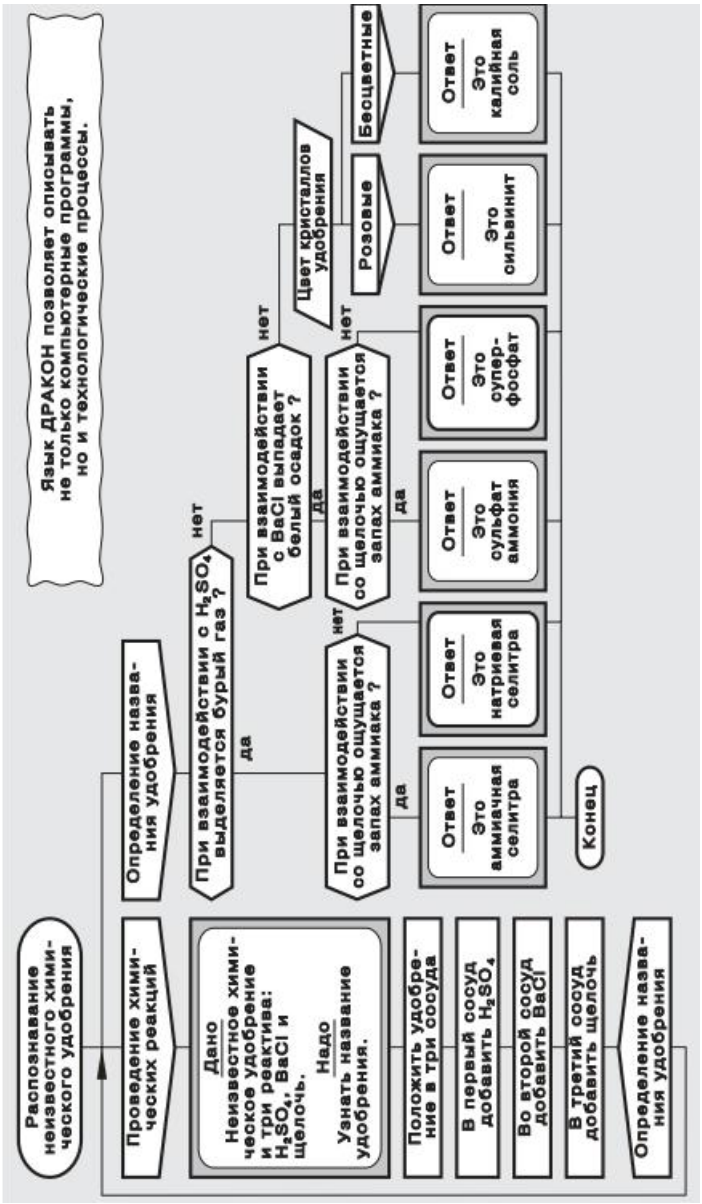


Рис.11 Предлагаемая технология химической диагностики

Ибо не могут быть решены задачи, необходимые для ЭС[9], как-то: распознавания смесей, вероятностный логический вывод, добавление новых правил.

В таких системах правило, а не алгоритм представляет собой когнитивные знания. Правило обрабатывается, объясняется, добавляется и удаляется как целое.

Задача диагностики как для химии, так и для АЭС аналогична по подходам.

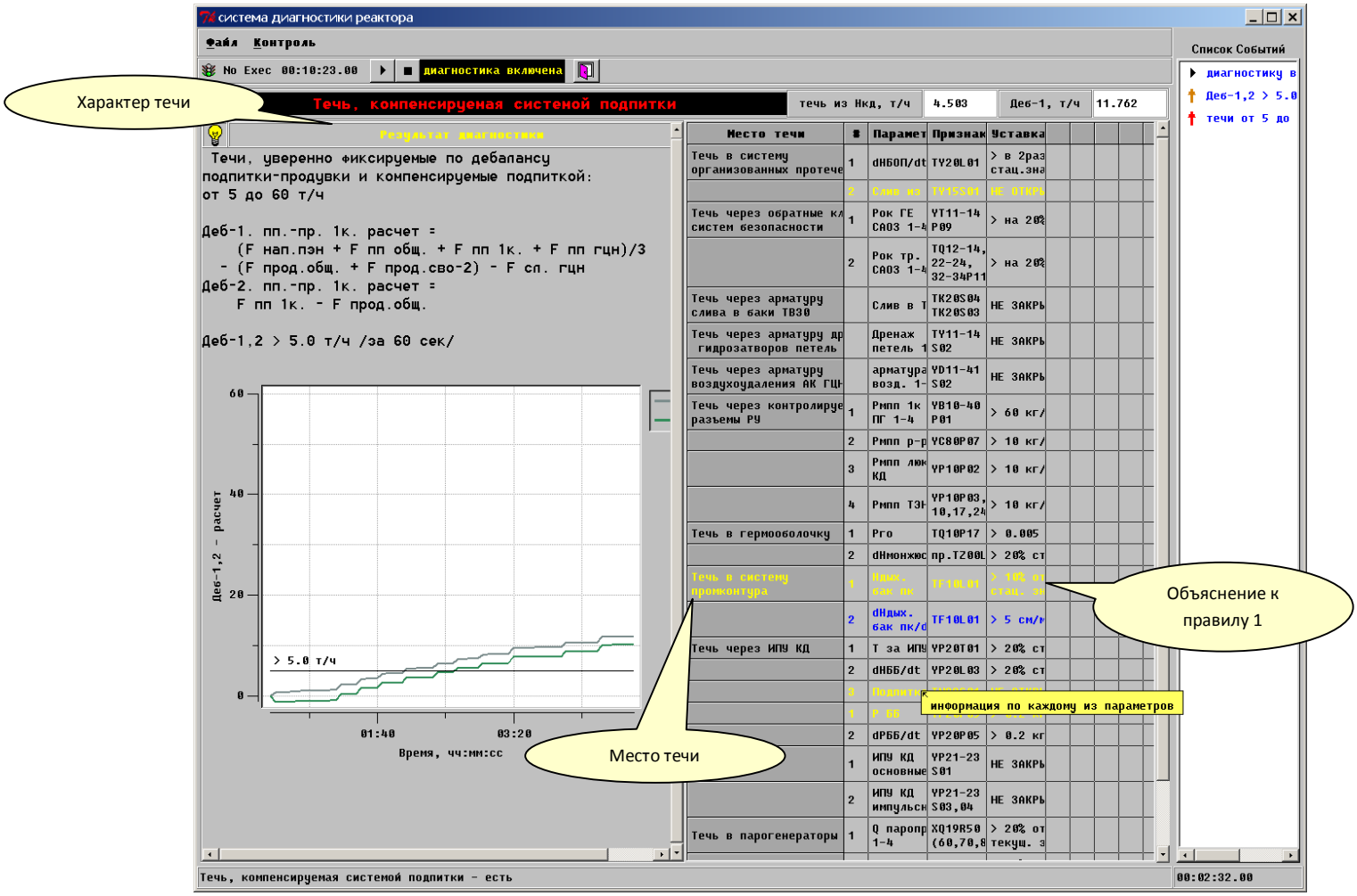


Рис.12 Диагностика течей теплоносителя

На Рис.12 приведен фрагмент системы Диагностики течей теплоносителя 1 контура, разработанной автором в 1998г. Слева на экране диагностируется характер течи, справа – место течи. Данная диагностика отловила характер: «Течь, компенсируемая системой подпитки». Место течи: «Течь в систему промконтура». Объяснение диагностики характера Вы видите слева. Расчетное значение дебалансов подпитки-продувки больше 5т/ч за 60 сек. Это видно на графике. Тренды расчетных величин пересекли отметку 5т/ч.

Место течи определилось. Это отобразилось *желтым*. Это означает, что правила 1 и 2 выполнились как *истина*. Для того, чтобы получить объяснения(Почему?), нужно нажать на соответствующую ячейку и появится окно с гипертекстом(подсказка: информация по каждому из параметров).

На рисунке 13 представлены окна определения места течи.

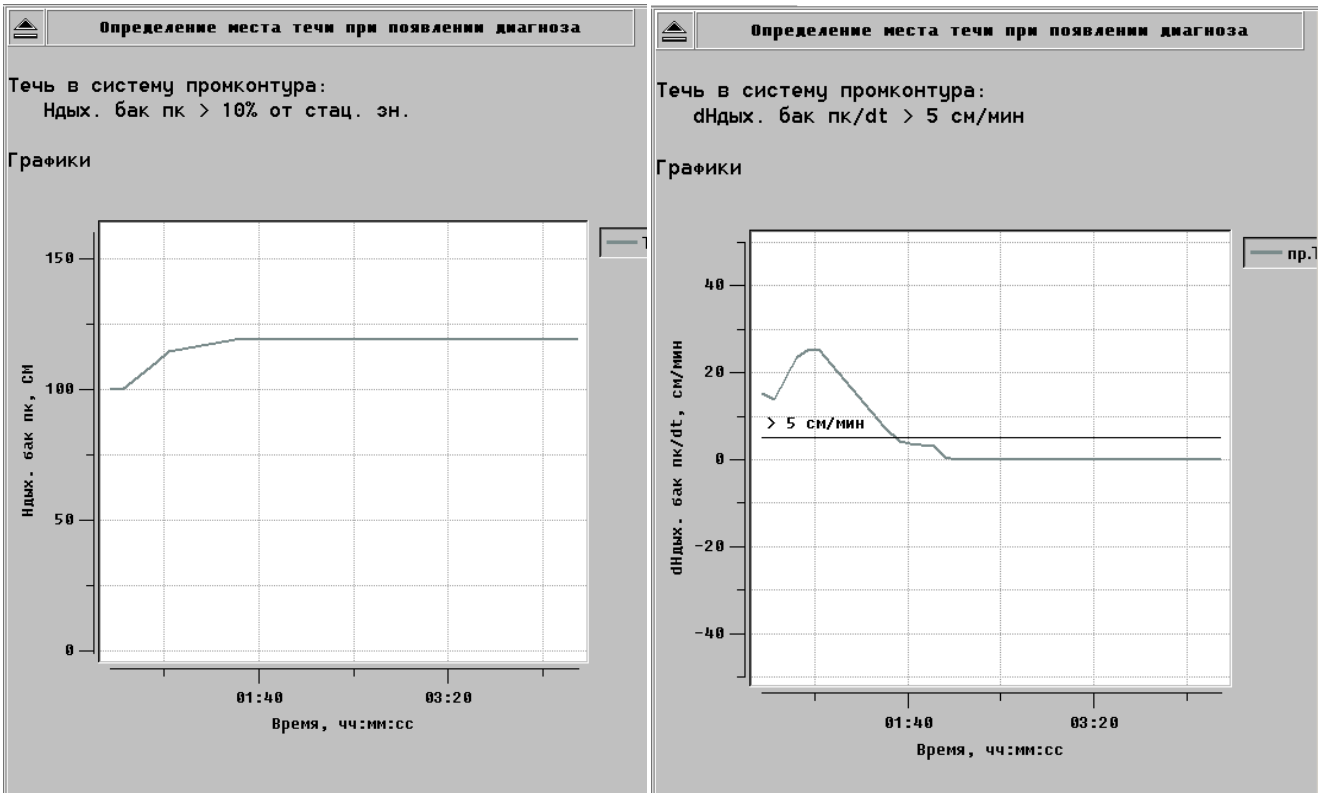


Рис.13 Объяснение: 1 и 2 правила для места течи

Первое правило слева гласит, что уровень должен превысить 10% от стационарного значения. Действительно, взяв 100 за стационарное.

Второе правило гласит, что производная по уровню больше 5см/мин. Действительно, была превышена. Кстати, такое *импульсное срабатывание на рис.12 отмечено синим*.

База знаний данной системы построена на структурах, каждая из которых включает в себя правило на непроцедурном языке, гипертекст для объяснения, краткие текстовые пояснения.

А что за информация содержалась внутри каждого правила? В реализованной системе был тот же самый псевдоязык в виде скрипта.

```
# ----------------------------------------------------------------
# ПРАВИЛО для n1_l81
# ----------------------------------------------------------------
ДИАГНОЗ Leak1l/n1_l81 -longname "Течь в систему промконтура"\
    -cycle 30
Leak1l/n1_l81 parm "Течь в систему\nпромконтура"\
    1 "Ндых.\nбак пк"\
    "TF10L01" "> 10% от\nстац. зн."

Leak1l/n1_l81 dp TF10L01
Leak1l/n1_l81 dc TF10L01 -tchecked 30
Leak1l/n1_l81 check {
    expr {[TF10L01 last value -for dpercent -tbuffer 300 -bias first] > 0.1}
}

Leak1l/n1_l81 htext {
Течь в систему промконтура:
   Ндых. бак пк > 10% от стац. зн.

Графики
%%
Graph {TF10L01}
GraphCfg element TF10L01 -label TF10L01
GraphCfg yaxis -min -1 -max 160 -title "Ндых. бак пк, СМ"
%%
Leak1l/n1_l81 RF 0.5
}
# ----------------------------------------------------------------
# ПРАВИЛО для n1_l82
# ----------------------------------------------------------------
ДИАГНОЗ Leak1l/n1_l82 -longname "Течь в систему промконтура"\
    -cycle 30
Leak1l/n1_l82 parm ""\
    2 "dНдых.\nбак пк/dt"\
    "TF10L01" "> $sn1_l82(1) см/мин"

Leak1l/n1_l82 dp TF10L01
Leak1l/n1_l82 dc TF10L01
Leak1l/n1_l82 check {
    expr {[TF10L01 min value -for df/dt -in mm -tbuffer 30] > $sn1_l82(1)}
}

Leak1l/n1_l82 htext {
Течь в систему промконтура:
   dНдых. бак пк/dt > $sn1_l82(1) см/мин

Графики
%%
Graph {TF10L01} -take {-for df/dt -in mm}
GraphCfg element TF10L01 -label пр.TF10L01
GraphCfg yaxis -min -50 -max 50 -title "dНдых. бак пк/dt, см/мин"
GraphLine $sn1_l82(1) "> $sn1_l82(1) см/мин"
%%
}
Leak1l/n1_l82 RF 0.4
```

Правило 1

Переменные процесса

Процедура вычисления

Гипертекст объяснения

Фактор уверенности

Правило 2

Переменные процесса

Процедура вычисления

Гипертекст объяснения

Фактор уверенности

Рис.14 Скрипт записи правил

Вид скрипта аналогичен виду для системы СИДОР. Те же самые обновляемые переменные процесса, процедура вычисления и гипертекст. И аналогичного вида редактор понадобился бы для ввода/модификации правил.

Однако, логический вывод производится на основании факторов уверенности.

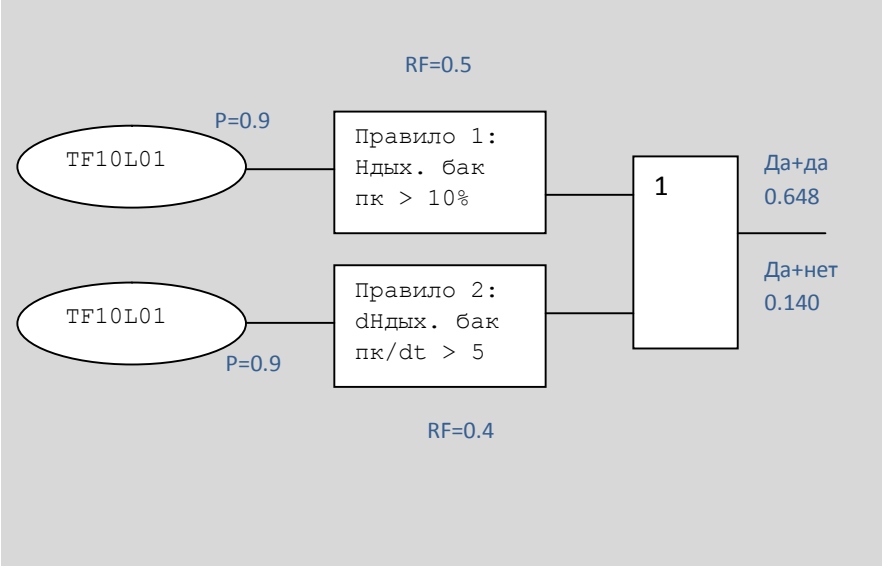


Рис.15 Вывод по факторам уверенности

Логический вывод основан на стэнфордовской теории факторов уверенности[10]. Учитываются вероятности измеряемых величин (в нашем случае считается процент достоверных значений – это для параметра TF10L01 P=0.9). Факторы уверенности для первого правила RF=0.5, для второго RF=0.4. Факторы уверенности имеют диапазон от -1 до 1. С приближением к 1 усиливается доверие к гипотезе, а с приближением к -1 – ее отрицание. При выполнении правила 2 фактор уверенности получается умножением на вероятность измерения как

*Фактор2 = P*RF2 = 0.36, если условие выполнено;*

*Фактор2 = P*RF2 = -0.36, если не выполнено.*

Объединение правил дает значение фактора как

*Фактор = Фактор1 + Фактор2 - Фактор1*Фактор2, если оба выполнены;*

*Фактор = Фактор1 + Фактор2 + Фактор1*Фактор2, если оба не выполнены;*

Фактор = (Фактор1 + Фактор2)/(1 – MIN(Фактор1,Фактор2)), иначе.

Вычислив, получим: если оба выполняются (да+да), фактор уверенности 0.648. Если только первое выполняется, фактор равен 0.140. На выходе ставится предел 0.5. Если фактор превысил 0.5, то общее условие «течь в систему промконтура» выполнено. Если меньше, то не выполнено.

Результат = (Фактор > 0.5)

По рисунку 11 Вы, конечно, сможете представить алгоритм химической диагностики учебной задачи. Работу реальной экспертной системы диагностики Вам отслеживать очень трудно. Почему?

Есть вещи, с которыми человек *справляется очень плохо*. И очень часто ошибочно пытается приспособить свой мозг к тем задачам, к которым этот мозг не приспособлен органически.

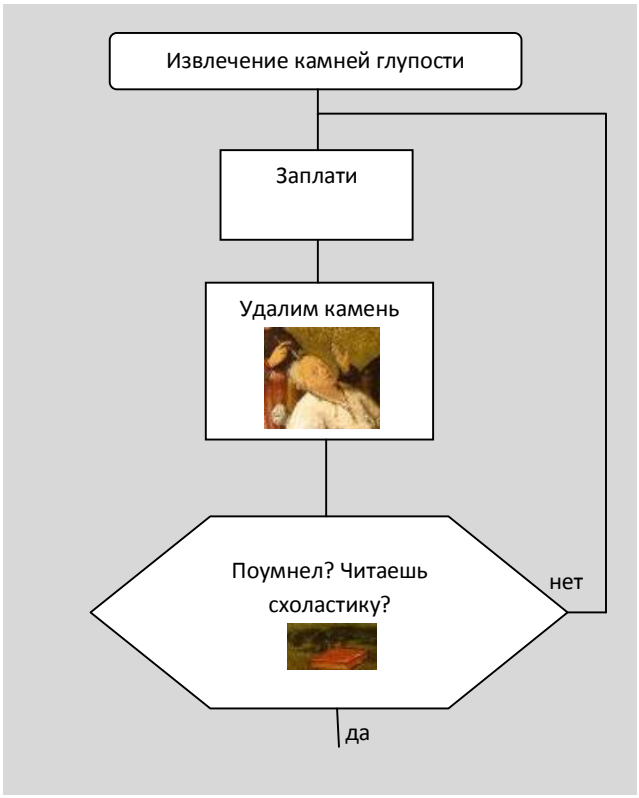


Рис.16 Извлечение камней глупости

На рисунке 16 представлено замечательное полотно И.Босха «Удаление камней глупости» и Дракон-схема этого алгоритма. Чтобы иметь постоянный доход, шарлатан-лекарь должен быть кровно заинтересован в непонятности схоластического труда. Ибо читая этот схоластический труд, простодушный заказчик почувствует себя глупцом. Конечно, отдавая деньги, он со временем *поумнеет*.

Алгебра факторов уверенности для человека такая же схоластика. Но компьютер эту работу выполняет очень хорошо!

Принцип 2. Выполните за человека всю рутинную и непосильную для него работу.

2.1. Деревья и softlogic

На высокой насыпи – сосны,
А меж ними вишни видны и дворец
В глубине цветущих деревьев
Басе

В качестве примера приведу видеокадр разработанной автором и введенной в промышленную эксплуатацию системы ИВС/СППБ 1блока Клн АЭС. Деревья СППБ(Системы Представления Параметров Безопасности) 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, «Давление в 1к менее 35 кг/см2» (Рис.17).

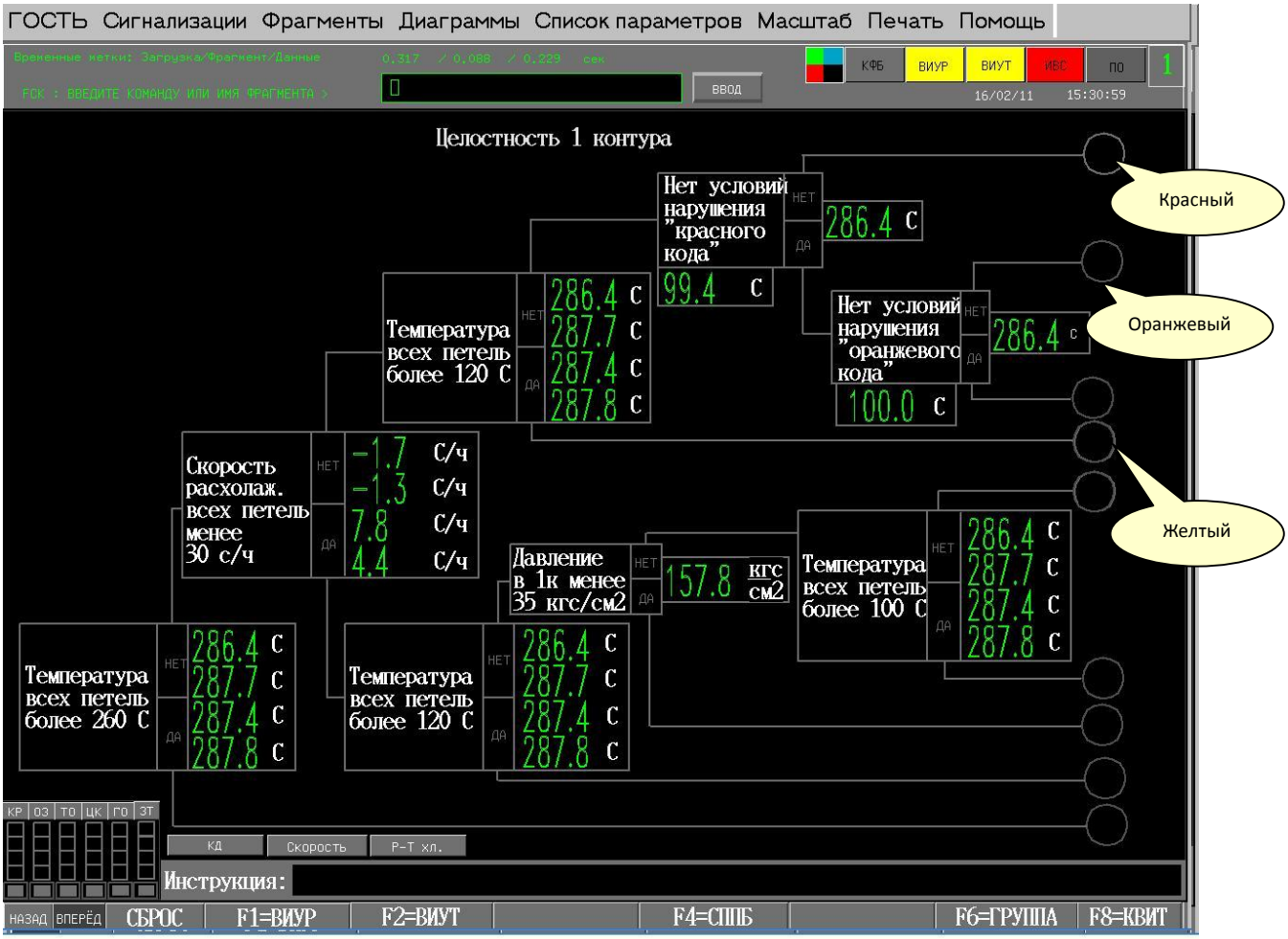


Рис.17 Видеокадр СППБ Целостность 1 контура

При наличие отклонений, и только при них, появляется красная, желтая и оранжевая индикация. По ней делается обобщенный вывод о безопасном состоянии энергоблока.

Попробуем нарисовать фрагмент Дракон-схемы для СППБ.

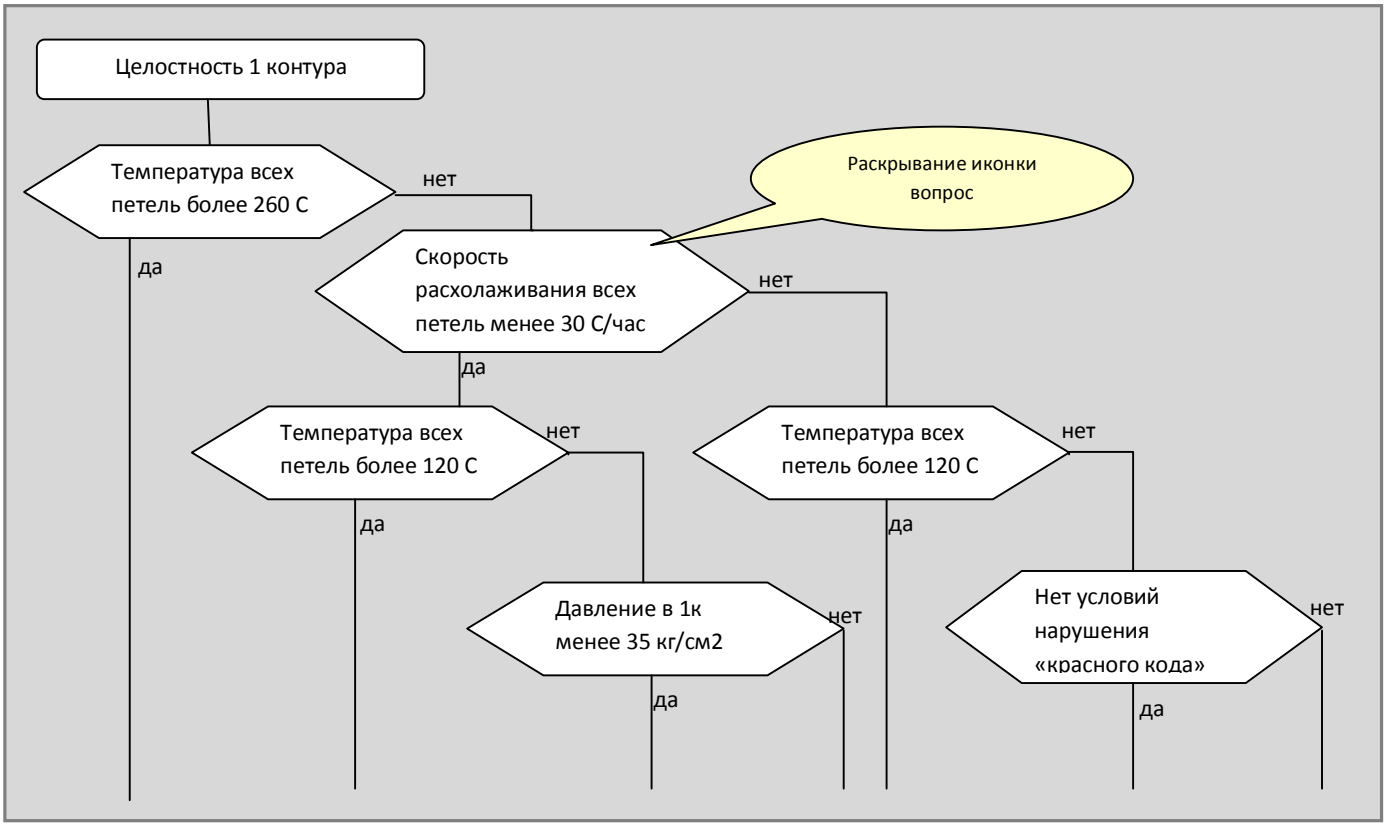


Рис.18 Фрагмент Дракон-схемы Целостность 1 контура

Дерево на схеме становится в нормальное положение, т.е. *корнем вверх*. Сама по себе схема не очень интересна, интересны условия в иконках ВОПРОС. Как вычислять «скорость расхолаживания всех петель менее 30 град.С/час»? Петель там четыре. И условие в иконке ВОПРОС *сложное*.

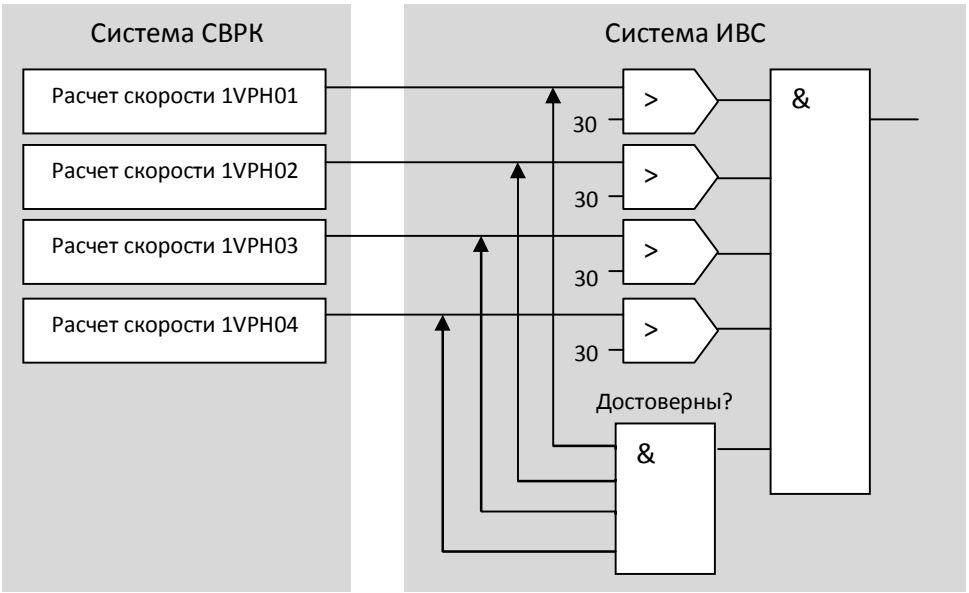


Рис.19 Расчет «Скорость расхолаживания менее 30 град.С/час»

Рисунок 19 раскрывает иконку ВОПРОС на языке логических схем FBD[5].

Условие это рассчитывается в двух, физически разделенных системах. В системе СВРК рассчитываются скорости изменения. В системе ИВС они сравниваются с уставками. Кроме того, осуществляется контроль достоверности. В той же системе ИВС отображаются деревья.

В этом примере возникает вопрос *гранулярности*. С какой степенью детализации нужно представлять схемы для человека-оператора? С точностью до одной машинной команды? Или с точностью, определяемой теми категориями, которыми мыслит человек? Заметьте, в естественном языке (пусть и с технологическими терминами) это выражается одной фразой: «Скорость расхолаживания всех петель меньше 30 град.С/час».

Одна фраза человеческого языка соответствует десяти блокам языка визуального.

Если Вы дальше раскроете иконки «Расчет скорости», то увидите целую программу. Эта программа набирает в буфер значения за определенный интервал (например, 1 час). Далее отбрасываются недостоверные значения. Далее численным методом вычисляется скорость по данным в интервале. И окончательно новое значение выставляется для передачи в смежные системы.

Я считаю, что визуальный язык должен быть рассчитан на человека. И схемы должны быть структурированы таким образом, чтобы соответствовать понятиям человеческого языка. При этом в схемах должен быть реализован масштабируемый интерфейс.

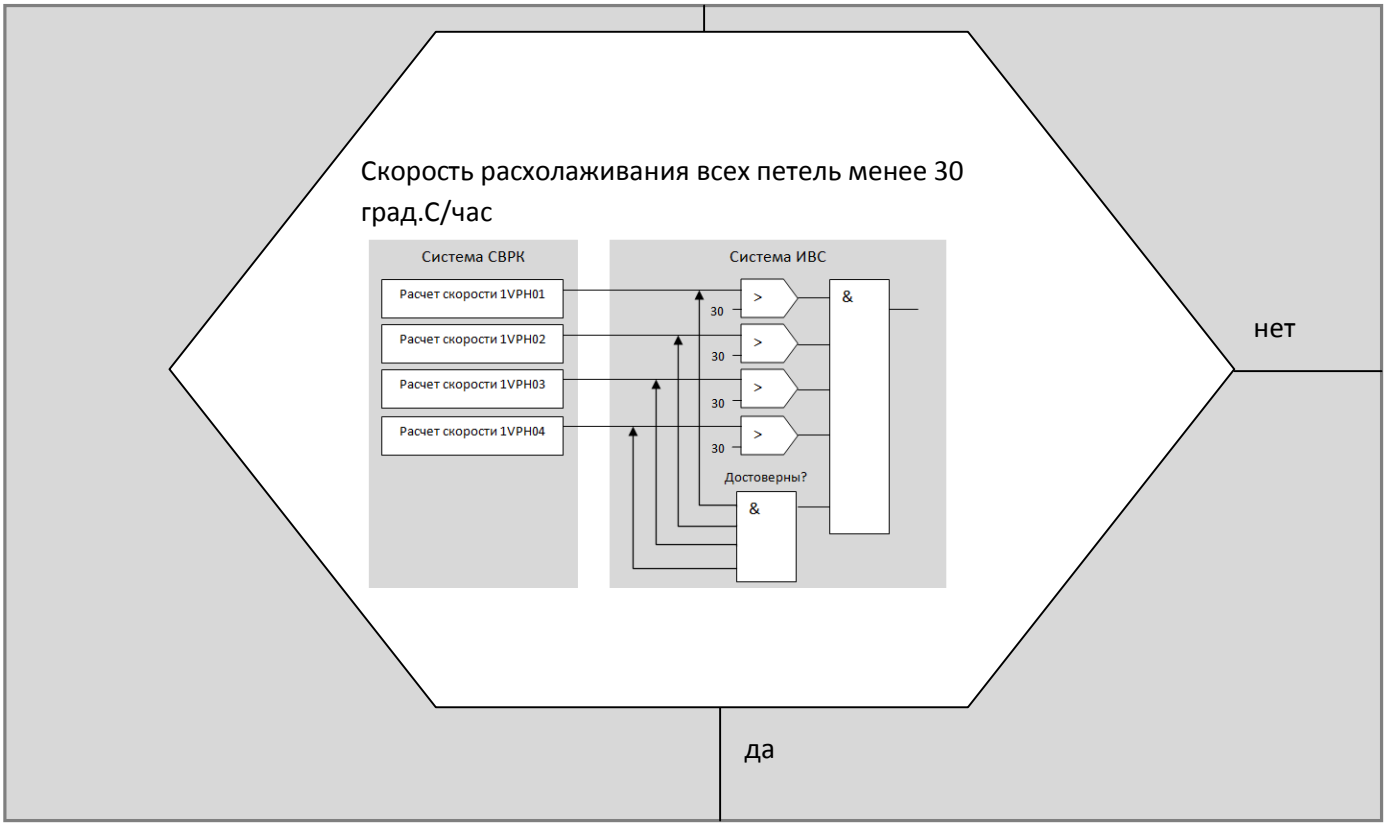


Рис.20 Масштабируемый интерфейс для иконки ВОПРОС

Концепция масштабируемого интерфейса[6] означает, что при приближении точки зрения к иконке появляются более мелкие детали изображения. Алгоритм расчета условия для ВОПРОСа в данном случае.

> Принцип 3. Структурируйте схемы в соответствии с категориями, которыми мыслит человек.

Вообще говоря, не факт, что данное визуальное представление будет лучше, чем текстовое. В тексте задано *одной строкой*:

```
-expr ($A1>$f1)&&($A2>$f1)&&($A3>$f1)&&($A4>$f1) -f1 -30 -in1 #1VPH01 -in2 #1VPH02 -in3 #1VPH03 -in4 #1VPH04
```

Следует заметить, что расчет достоверности уже выполняется внутри интерпретатора. Поэтому выражение содержит формулу, уставку и четыре адреса. Как вы считаете, какое представление лучше? Если текстовое, значит, нужно текст и выводить при масштабировании.

Есть еще один аргумент в пользу текста. С текстом проще работать. Представьте, что Вам нужно модифицировать алгоритм на *работающей системе*. Как Вы поступите с ДРАКОН-схемой или FBD-схемой? Между прочим, SCADA-платформа СУОК, на которой реализована система рис.17, *позволяет обновлять текстовые формулы в базе данных реального времени на двух серверах за одну транзакцию*.

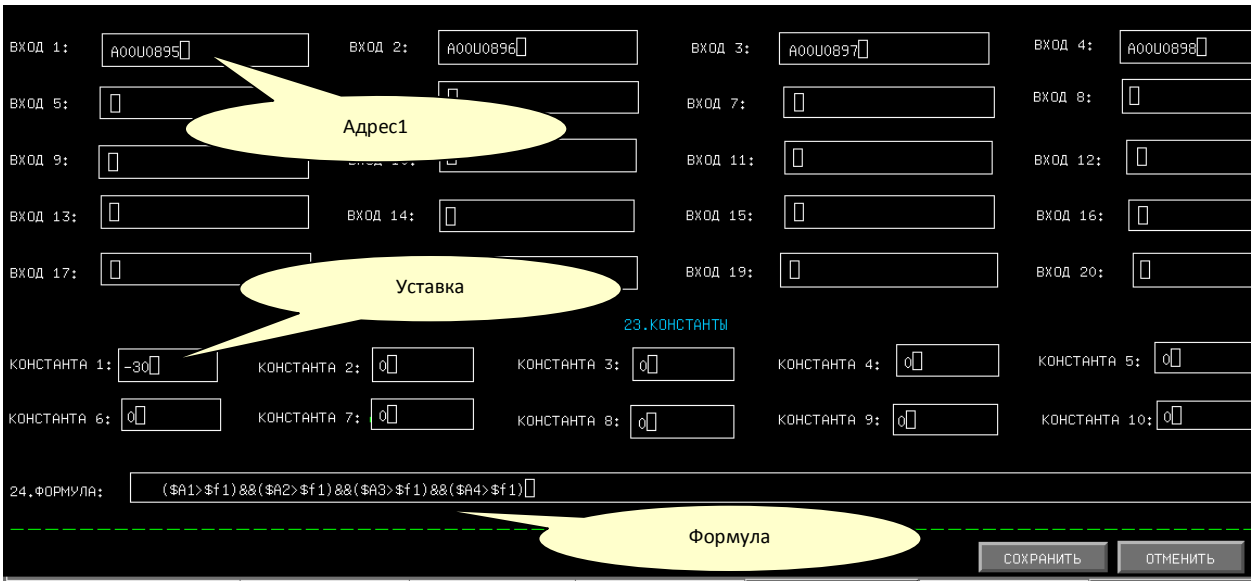


Рис.21 Редактирование формул

На рисунке 21 представлен экран для обслуживающего персонала, который имеет доступ к формулам.

3. Системы-тренажеры

3.1. История с картинками

"Римский император требует тебя в начальники своей армии
и вручает тебе судьбу Австрии и Италии.
Иди спасай царей..."
Павел I – Суворову перед походом через Альпы

Я слегка занимался САПРом для тренажеров в виде языков блок-схем и логических схем(FBD). Когда в один прекрасный момент меня вызвали к зам.генерального и приказали срочно собираться в США. Там у нас были 2 совместных с американцами проекта. С одного из них юноша, мной обучаемый, сбежал, найдя в другом месте работу. А на другом девушка имела большие проблемы со сдачей моделей цифровых регуляторов на блок-схемах. «Иди, спасай честь фирмы», - сказал мне зам.генерального.

Ситуация с блок-схемами усложнялась еще и тем, что исходная документация с АЭС была, мягко говоря, неоднозначная. Описания работы – скупые. Блок-схемы изрисованы карандашом. Любой человек, глянув на эти альбомы, впадал в состояние *экзистенциальной фрустрации*. К тому же еще и Али, менеджер восточного происхождения с американской стороны, был очень недоволен, что такую сложную систему делала девушка. Он был очень хороший менеджер, кто работал, тот знает организаторские способности американцев. На каждом этапе сдавались альбомы технической документации, защищалась каждая строчка. Но я приехал с редактором блок-схем и кодогенератором к нему, и девушка сама нарисовала, защитила и сдала тот этап системы из десятков регуляторов по тридцать листов в каждом (Рис.22).

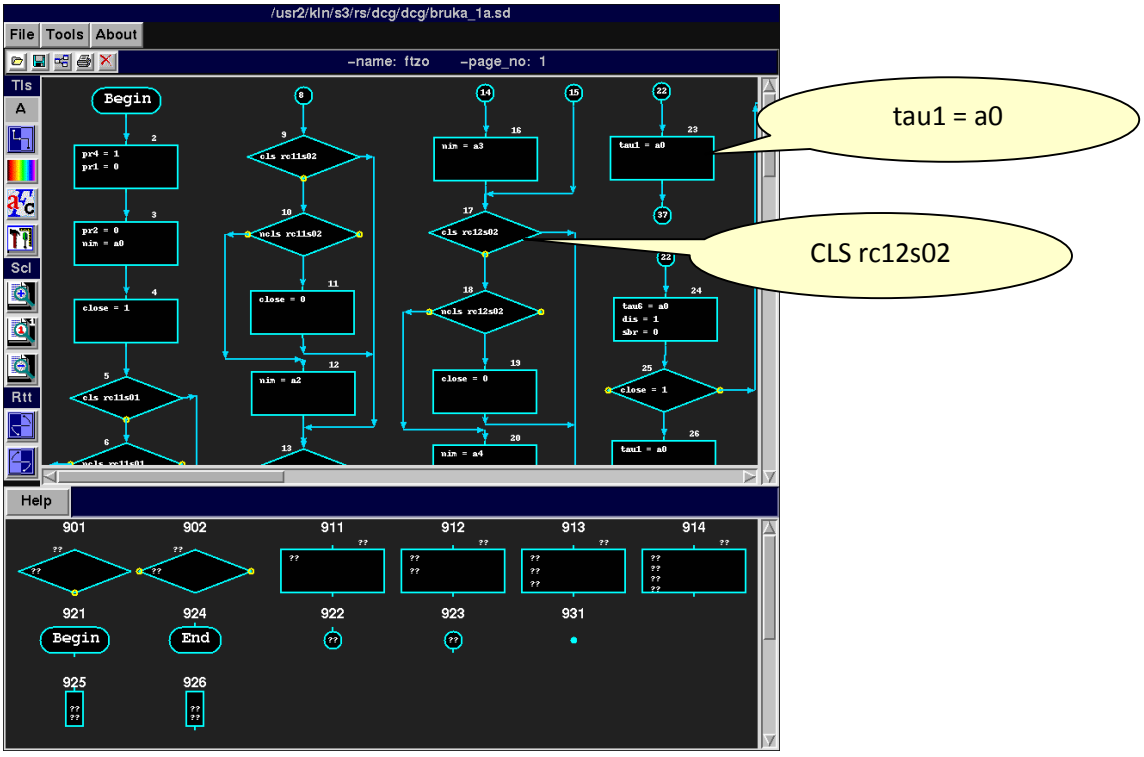


Рис.22 Редактор алгоритмов блок-схем. Регулятор БРУК-А.

Это была первая для меня «пущенная в ход» система с графическими языками. Все алгоритмы были распечатаны. В заголовок шапки входила структура с полями: имя системы, имя схемы, имя разработчика, кто принял.

Когда менеджеру были представлены отчеты в напечатанном виде, он был очень доволен. Включился в работу и представитель заказчика, как только увидел схемы. Он стал их корректировать.

> Принцип 4. Предоставьте хорошие средства создания твердых копий.

Спроецируем имевшиеся тогда подходы на Дракон-схему сегодняшнего дня. Каковы методы генерации кода для схемы, представленной в Дракон-редакторе? Те же.

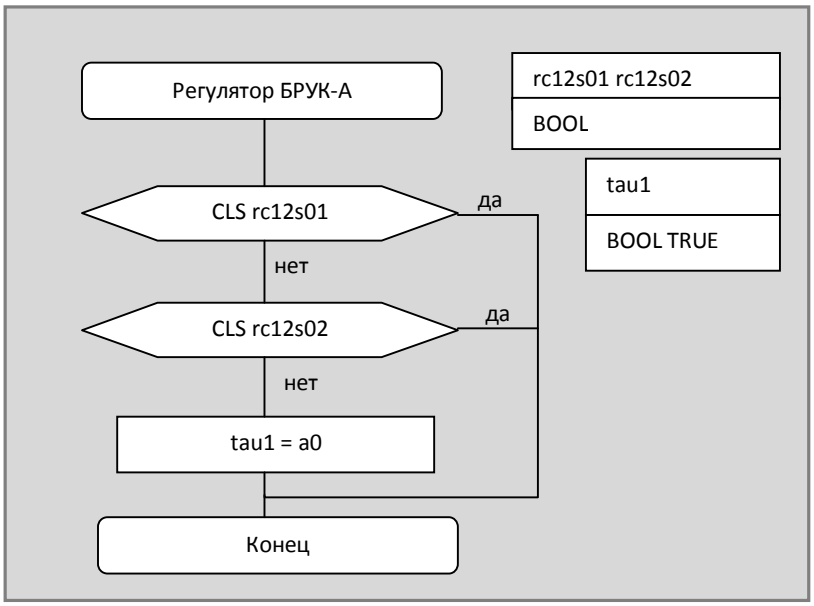


Рис.22 Фрагмент алгоритма как Дракон-схема

В каждую иконку вводился тот же видимый текст. Из него генерился код уровня 4, аналогичный скрипту выше. А затем преобразовывался в программу на Фортране в системе моделирования US3. Имя системы было введено: rx. Код генерился построчно:

```
Исходный:  tau1 = a0
Сгенеренный: rx:tau1 = a0
Исходный:  CLS rc12s01
Сгенеренный: IF (rx:rc12s01_cls)
```

Если программа по алгоритму использует глобальные переменные (как в этом коде), для каждой новой переменной нужно сформировать описание для импорта в базу данных.

```
/ADD rx:rc12s01_cls
.DESC valve closing status
.UNIT -
.TYPE L1
.SYS rx
.DIM 1
```

Псевдоязык представляется аналогичным непроцедурным системам. Для данной схемы получим языковое описание.

```
ЗАГОЛОВОК bruka_1a «Регулятор БРУК-А»
ПОЛКА x1 –vars {rc12s01 rc12s02} –type BOOL –descr {«Отсечной клапан 1» «Отсечной
клапан 2»} –kind input
ПОЛКА x1 –vars tau1 –type BOOL –kind output
ВОПРОС o1 –command {CLS rc12s01}
o1 goto o2 o4
ВОПРОС o2 –command {CLS rc12s02}
o2 goto o3 o4
ДЕЙСТВИЕ o3 –command {tau1 = a0}
o3 goto o4
КОНЕЦ o4
```

Рис.23 Представление алгоритма на псевдоязыке

Этот скрипт преобразуется в программу. Если этот алгоритм используется в единственном экземпляре, то из одного скрипта получается одна программа. И один импортный файл для ввода в базу данных.

Если этот алгоритм используется не в единственном экземпляре, то он в разных случаях будет работать с разными параметрами. Об этом речь пойдет в следующем разделе.

3.2. Три схемы передачи параметров

«Перва дуката Вишну Бога
Втора дуката Коляда Бога
Трета дуката Бела Бога»
Золотая книга Коляды

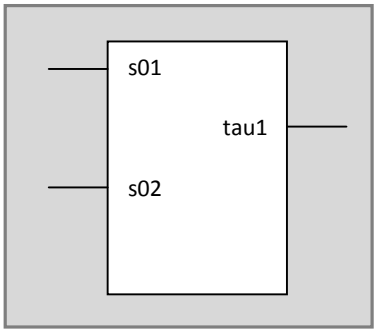
В соответствии со стандартом МЭК 61499 [11] предлагает «новый уровень», интегрирующий имеющиеся алгоритмы программных языков автоматики в единые схемы в виде блоков. Стандарт направлен на встраиваемые и распределенные системы. Это применимо и к тренажерам.

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

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

Если алгоритм рисунка 22 используется не в единственном экземпляре, то он может быть вызван или как функция, или как процедура, или как функциональный блок. Все тексты программ должны автоматически генериться из псевдоязыка рис.23. Префикс «rc12» отбрасываем.

Алгоритм как функция.



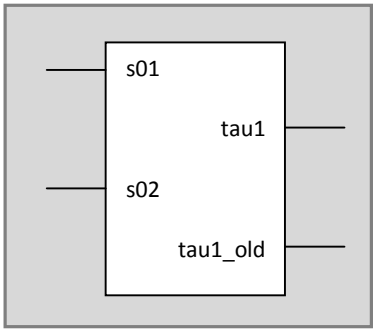

```
PROCEDURE bruka_1a(s01, s02: BOOLEAN): BOOLEAN;
      (*Регулятор БРУК-А*)
      VAR tau1: BOOLEAN;
BEGIN
      tau1 := TRUE;
      IF ~s01 THEN
            IF ~s02 THEN
                  tau1 := FALSE;
            END
      END;
      RETURN tau1
END bruka_1a
```

Рис.24 Внешний вид и реализация в виде функции

Если схема используется вызывающей программой как функция, то в вызывающей программе вставляется графический блок как на рисунке 24. У функции только один выход. Если нужна реализация программного кода в виде функции, то из псевдоязыка рис.23 генерится код вида рис.24. В заголовке схемы можно определять вид объекта: функция, процедура или функциональный блок.

Вызов `outval := bruka_1a(rx_rc22s01_cls, rx_rc22_s02_cls);`

Алгоритм как процедура.



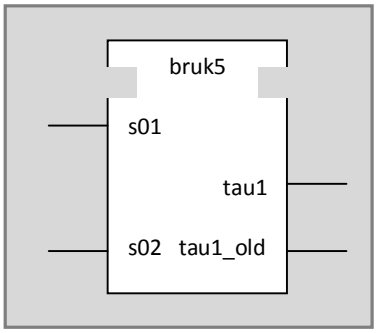

```
PROCEDURE bruka_1a(s01, s02: BOOLEAN; VAR tau1, tau1_old: BOOLEAN);
      (*Регулятор БРУК-А*)
BEGIN
      tau1_old := tau1;
      tau1 := TRUE;
      IF ~s01 THEN
            IF ~s02 THEN
                  tau1 := FALSE;
            END
      END
END bruka_1a
```

Рис.25 Внешний вид и реализация в виде процедуры

В вызывающей программе появляется графический блок как на рисунке 25. У процедуры может быть несколько выходов. Я искусственно добавил сохранение предыдущего значения в переменной tau1_old. Код в виде процедуры генерится из псевдоязыка.

Вызов `bruka_1a(rx_rc22s01_cls, rx_rc22_s02_cls, out_val, out_val_old);`

Алгоритм как функциональный блок.




```
(* Базовый тип *)
TYPE FB_p POINTER TO FunctionalBlock;
FunctionalBlock = RECORD
       update: PROCEDURE(f: FB_p);
       init: PROCEDURE(f: FB_p);
END;
(* Тип bruka_1a *)
TYPE bruka_1a_p POINTER TO bruka_1a;
bruka_1a = RECORD(FunctionalBlock)
       (* Inputs *)
       BOOLEAN s01;
       BOOLEAN s02;
       (* Outputs *)
       BOOLEAN tau1;
       BOOLEAN tau1_old;
       (* Internal *)
END;
PROCEDURE bruka_1a_update(f: FB_p);
       (*Регулятор БРУК-А*)
       VAR r: bruka_1a_p;
BEGIN
       r:= f(bruka_1a_p);
       r.tau1 := TRUE;
       IF ~r.s01 THEN
              IF ~r.s02 THEN
                     r.tau1 := FALSE;
              END
       END
END bruka_1a
PROCEDURE bruka_1a_init(f: FB_p);
       VAR r: bruka_1a_p;
BEGIN
       r:= f(bruka_1a_p);
       r.tau1_old := TRUE;
END
(* Инсталляция экземпляра bruk5 *)
VAR bruk5: bruka_1a;
bruk5.update := bruka_1a_update;
bruk5.init := bruka_1a_init;
```

Описание базового типа: инициализация и обновление

Тип bruka_1a есть структура и 2 метода

Инсталляция экземпляра объекта bruk5

Рис.26 Внешний вид и реализация в виде функционального блока

Если схема является функциональным блоком, то схеме ставится в соответствие тип (или класс) объекта. В нашем случае это bruka_1a. Этот тип содержит поля структуры и методы. Поля структуры это: входы, выходы, внутренние переменные (не отображаются на графическом блоке).

Функциональный блок bruka_1a наследуется от типа FunctionalBlock. Тип FunctionalBlock единый для всех типов блоков. Он имеет поля процедур (update, init), заданные в этом базовом классе. Первая вызывается периодически, вторая – однократно при инициализации. Программная часть для типа bruka_1a уже генерится для данного конкретного алгоритма. Из схемы алгоритма получается конкретная реализация bruka_1a_update. Если в иконках ПОЛКА алгоритма задаются начальные значения выходных или внутренних переменных, генерится реализация bruka_1a_init, устанавливающая эти начальные значения.

Так, как в тренажере все технологические переменные глобальные, каждый экземпляр переменной нужно вводить в базу данных. Такой экземпляр, как bruk5. И нужно инсталлировать процедуры update и init.

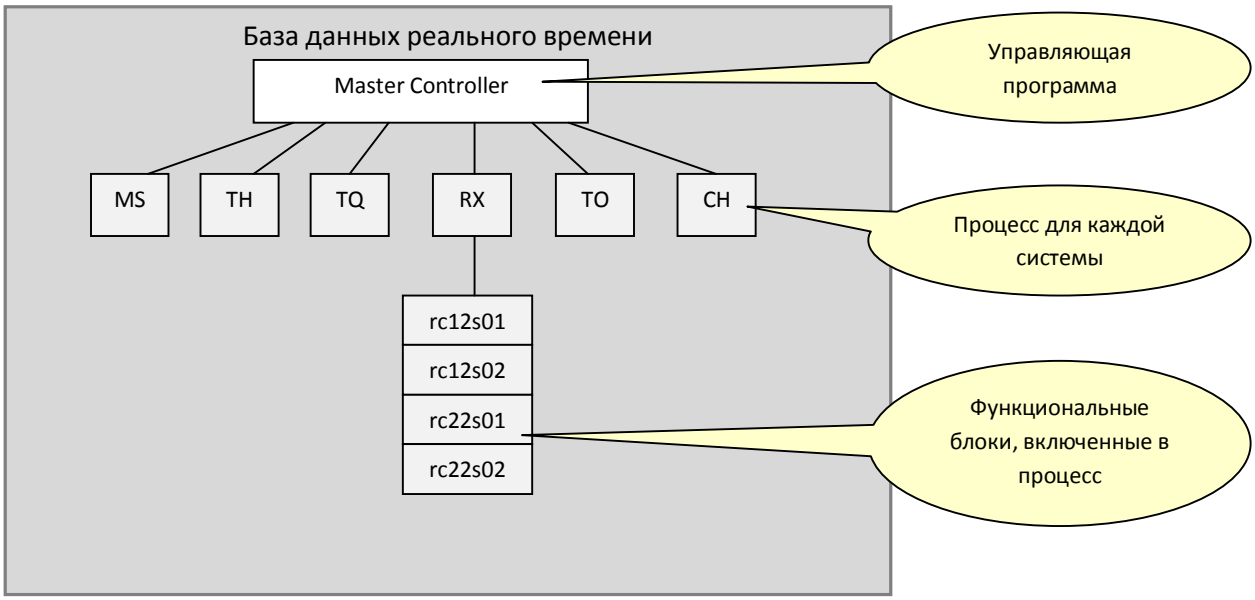


Рис.27 Программная архитектура тренажера

Программная архитектура тренажера (рис.27) построена на базе данных реального времени. Она представляет собой общую память, доступную всем процессам. Конкретно для каждого процесса реализована сегментация доступа. Общая память инициализируется до запуска конкретной программы. Поэтому все переменные в общей памяти *уже размещены до запуска программы*. Запущенный процесс поднимает за собой все зависимые программные модули. Таких функциональных блоков, каждый из которых привязан к единице оборудования, может быть до 10000.

Управляющая программа осуществляет диспетчеризацию работы процессов. Каждому процессу передается сигнал, вызывающий его с определенным периодом реального времени с определенной фазой (в определенном фрейме). Это, в конечном итоге, приведет в вызову метода update для каждого из функциональных блоков процесса (rc12s01, rc12s02, rc22s01, rc22s02, …).

Реализация механизма функциональных блоков в системе моделирования US3 была реализована интересным образом для неструктурированного языка Fortran. В базе данных реального времени заводили только переменные простых типов. Например, для нашего алгоритма заводились переменные rx:rc12s01_cls для задвижки 12 и rx_rc22s01_cls для задвижки 22. И в US3 был реализован механизм генерации, когда из одного шаблонного программного модуля (generic component) создавались несколько реальных программных модулей (specific component), которые и включались в систему под именами bruka_rc12, bruka_rc22, ...

```
PROCEDURE bruka_%tag#
BEGIN
      %id#:%tag#tau1 := TRUE;
      IF ~%id#:%tag#s01_cls THEN
            IF ~%id#:%tag#s02 THEN
                  %id#:%tag#tau1 := FALSE;
            END
      END;
END bruka_%tag#

PROCEDURE bruka_rc12
BEGIN
      rx:rc12tau1 := TRUE;
      IF rx:rc12s01_cls THEN
            IF rx:rc12s02 THEN
                  rx:rc12tau1 := FALSE;
            END
      END;
END bruka_rc12
```

Разрабатываемая шаблонная программа generic component

Сгенеренная программа specific component. Параметры id=rx, tag=rc12

Рис.28 Реализация функциональных блоков без использования структур

На рисунке 28 представлена эта механика. Схеме алгоритма соответствует шаблон, т.е. generic component. Далее нужно для каждого объекта выбрать параметры подстановки. Это – система и технологическое имя оборудования (id=rx, tag=rc12). Далее осуществляется генерация программных модулей. Одновременно генерится файл импорта для базы данных. Вводятся по 3 глобальные переменные в базу данных. Расширяется общая память. По сути дела, тот же процесс инсталляции.

Конечно, без структур представление не очень удобное. Но есть системы, построенные на таких принципах. И процедуры init там не требуется, ибо начальное значение задается базе данных.

Вам потребовалось такая реализация функциональных блоков?

Обратите внимание, что из *псевдоязыка нужно генерить прямо specific component*, минуя лишний шаблон. Задайте только параметры подстановки.

3.3. Объединим функциональные блоки, логические и Дракон-схемы

его смерть на конце иглы, та игла в яйце,
яйцо в утке, утка в зайце, тот заяц сидит в каменном сундуке,
а сундук стоит на высоком дубу,
и тот дуб Кощей Бессмертный, как свой глаз, бережет

В этом разделе я попытаюсь объединить в системе моделирования 3 уровня:

1. Коммутация функциональных блоков;
2. Реализация алгоритма функционального блока на языке логических схем FBD;
3. Реализация алгоритма вызываемой процедуры на Драконе.

Рассмотрим модель блока управления клапаном БУК (buk). Таких клапанов – сотни на тренажере АЭС. Каждый клапан будет представлять собой экземпляр функционального блока. Технологическое имя клапана RA14S05.

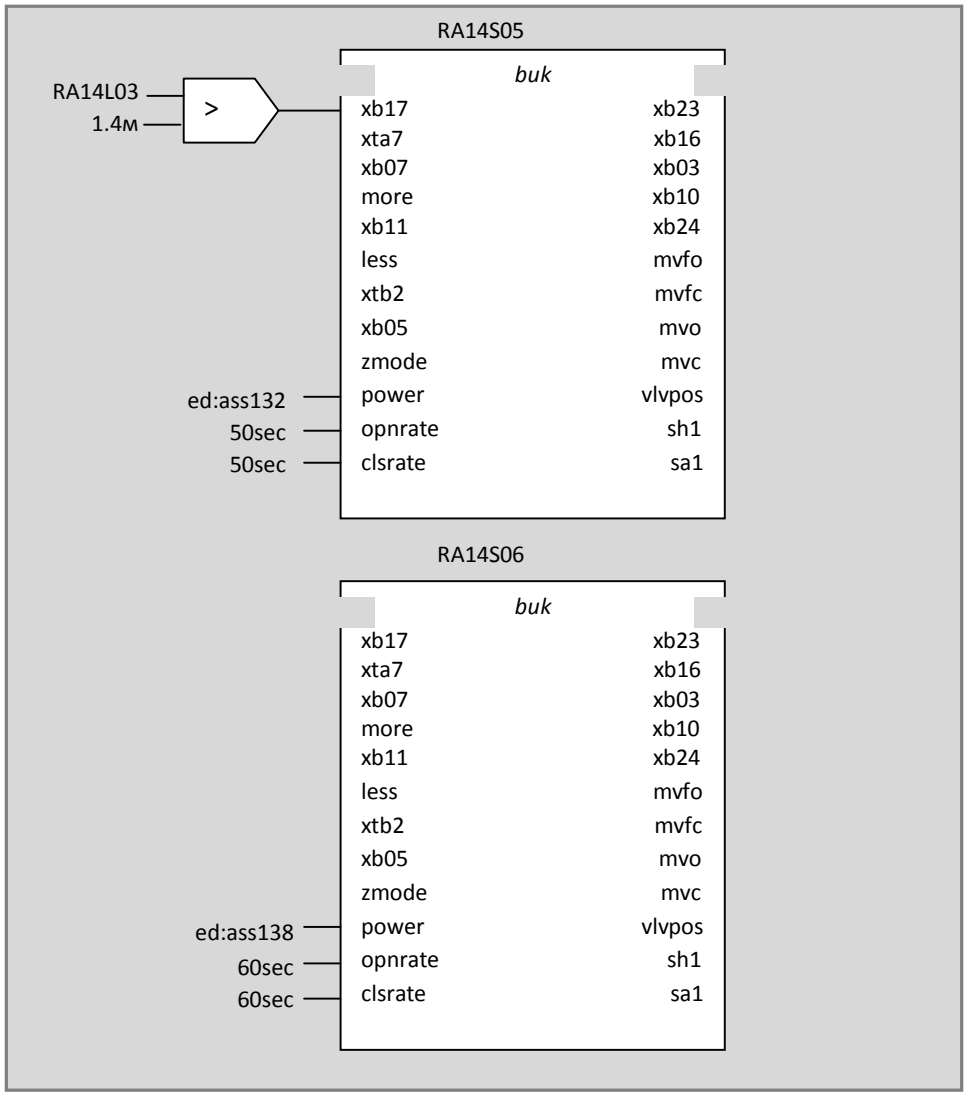


Рис.29 Схема функциональных блоков

На рисунке 29 представлено подключение функциональных блоков RA14S05, RA14S06. Это - два экземпляра объекта типа buk. Исследуемый нами клапан имеет подключенный сигнал xb17 «запрет на открытие», когда уровень RA14L03 более 1.4м. Кроме того, у него своя сборка питания 132 и время хода равно 50сек.

В соответствии с концепцией масштабируемого интерфейса[6] приближаемся к функциональному блоку и окно раскрывается. Мы видим логическую схему БУКа на основе языка FBD (Рис. 30). Коричневым цветом сверху и снизу представлены описания.

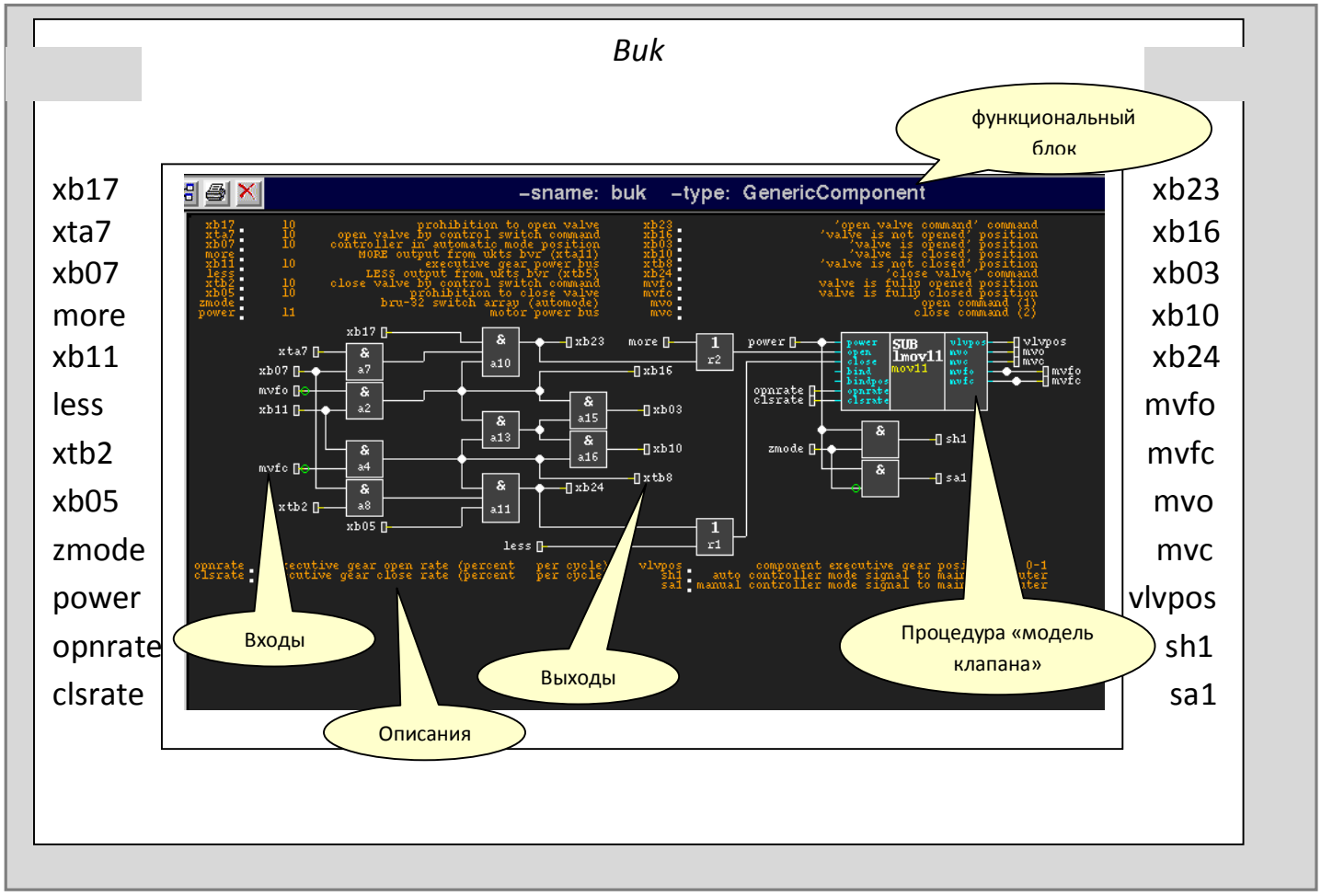


Рис.30 Схема блока управления клапаном для тренажера

Схема раскрылась, и мы видим, что те же имена входов и выходов участвуют в логической схеме(картинка взята из разработанного автором пакета для FBD). Белые прямоугольники с текстом, смотрящим влево – это входы. Белые прямоугольники с текстом, смотрящим вправо – это выходы. Они соответствуют входам/выходам функционального блока. Элементы – обычные функции И, ИЛИ, НЕ. Им соответствуют операторы, присваивающие логические выражения. Например, для элемента a7 логическое выражение будет представлено в виде

```
a7 = xta7 & xb07;
a2 = mvfo & xb11;
a10 = xb17 & a2 & a7;
```

Блок lmov11 справа – это процедура модели клапана. Она создавалась следующим образом: вводился блок GENERIC – создание блока процедуры, задавался тип lmov11, и для данного модуля генерилось графическое представление, которое Вы видите на экране. Вызывается процедура следующим образом:

```
lmov11(power, r2, r1, FALSE, 0, opnrate, clsrate,
       vlvpos, mvo, mvc, mvfo, mvfc);
```

Снова приближаемся к процедуре, и окно раскрывается. Внутри нее – уже Дракон-схема.

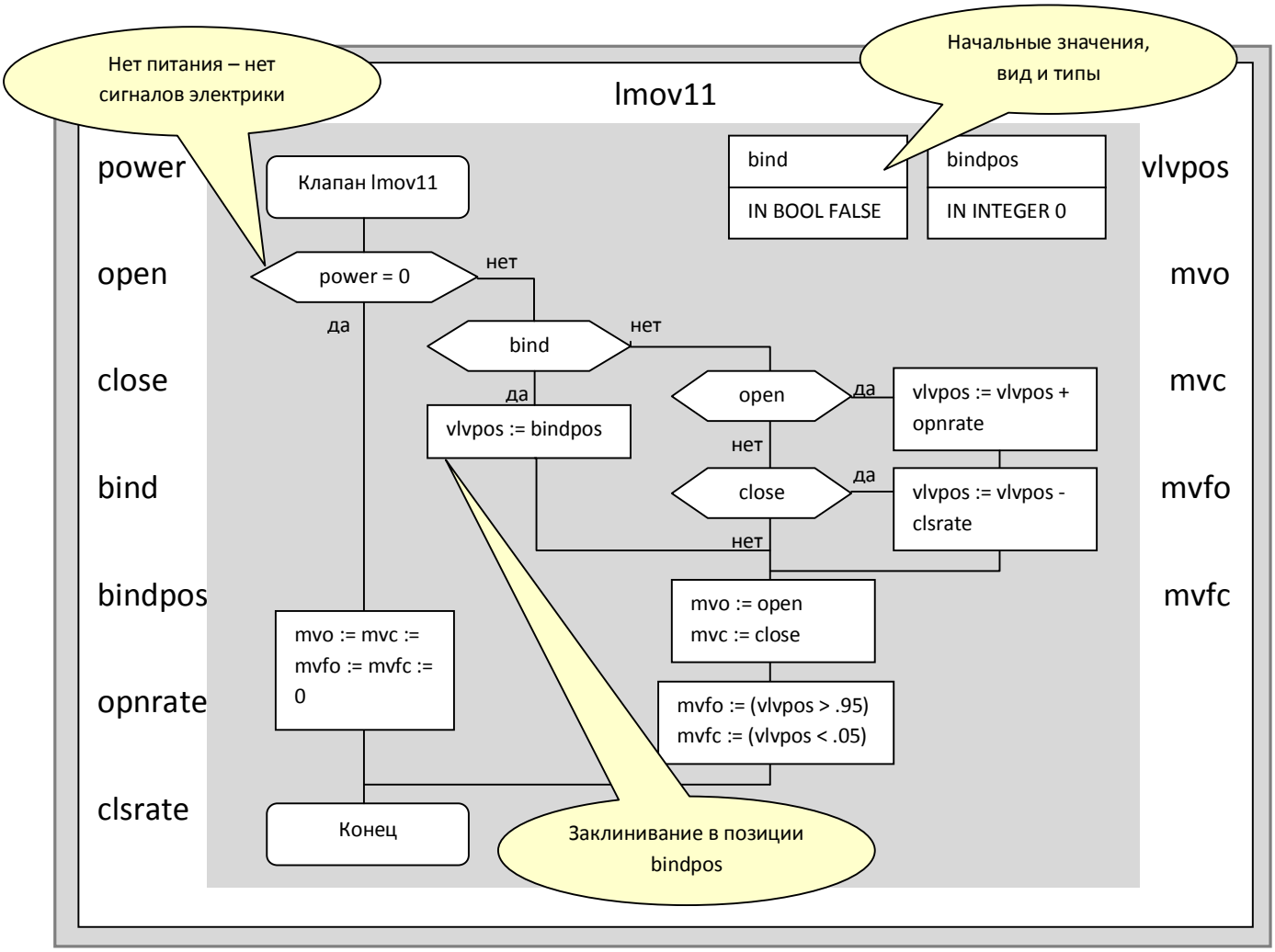


Рис.31 Процедура модель клапана

Алгоритм, реализованный на Дракон-схеме, вычисляет состояние клапана. Клапан характеризуется положением vlvpos. Значение 0 соответствует открытому состоянию, 1 – закрытому. Нижний концевой выключатель mvfc соответствует положению клапана менее 5%. Верхний концевик mvfo – более 95%. Электрические сигналы mvo, mvc сигнализируют, что клапан открывается и закрывается соответственно.

Если нет питания power, все электрические сигналы нулевые. Если есть введенный в модель отказ заклинивания bind, клапан останавливается в положении bindpos. Если есть сигнал открытия open, положение клапана меняется на величину opnrate. Это – константа, зависящая от интервала, с которым вызывается процедура. На закрытия такое же приращение clsrate. Константа вычисляется, чтобы клапан полностью закрывался за 50 секунд.

Программный код процедуры представлен на рисунке 32.

```
PROCEDURE lmov11(power, open, close, bind: BOOLEAN; bindpos, opnrate, clsrate: REAL;
VAR vlvpos: REAL; VAR mvo, mvc, mvfo, mvfc: BOOLEAN);
       (* Процедура модели клапана *)
BEGIN
       IF ~power THEN
              mvo := FALSE;
              mvc := FALSE;
              mvfo := FALSE;
              mvfc := FALSE
       ELSE
              IF bind THEN
                     vlvpos := bindpos
              ELSE
                     IF open THEN vlvpos := vlvpos + opnrate END;
                     IF close THEN vlvpos := vlvpos - clsrate END
              END
              mvo := open;
              mvc := close;
              mvfo := (vlvpos > 0.95);
              mvfc := (vlvpos < 0.05);
       END
END lmov11
```

Рис.32 Реализация процедуры блока управления клапаном

Таким образом, мы успешно соединили три уровня вложенности графических схем при помощи масштабируемого интерфейса. Верхний уровень – это язык FBD, расширенный функциональными блоками. Средний уровень – это язык логических схем FBD. Нижний уровень – это Дракон. Уровни 2 и 3 могут меняться местами: Дракон может вызывать логическую схему (вспомните условие для Дракон-схемы на рисунке 20). Но функциональные блоки должны быть на верхних уровнях, чтобы *обеспечить инсталляцию.*

Принцип 5. Сделайте ему масштабируемый интерфейс доступа к данным на всех уровнях.

4. Распределенные АСУТП

4.1. Структура и общие принципы

Опять скажу: никто не обнимет необъятного!
Козьма Прутков

Стандарт МЭК 61499 это - Функциональные блоки для проектирования встроенных и распределенных систем управления. В данном разделе я буду описывать концепцию построения именно распределенной АСУТП.

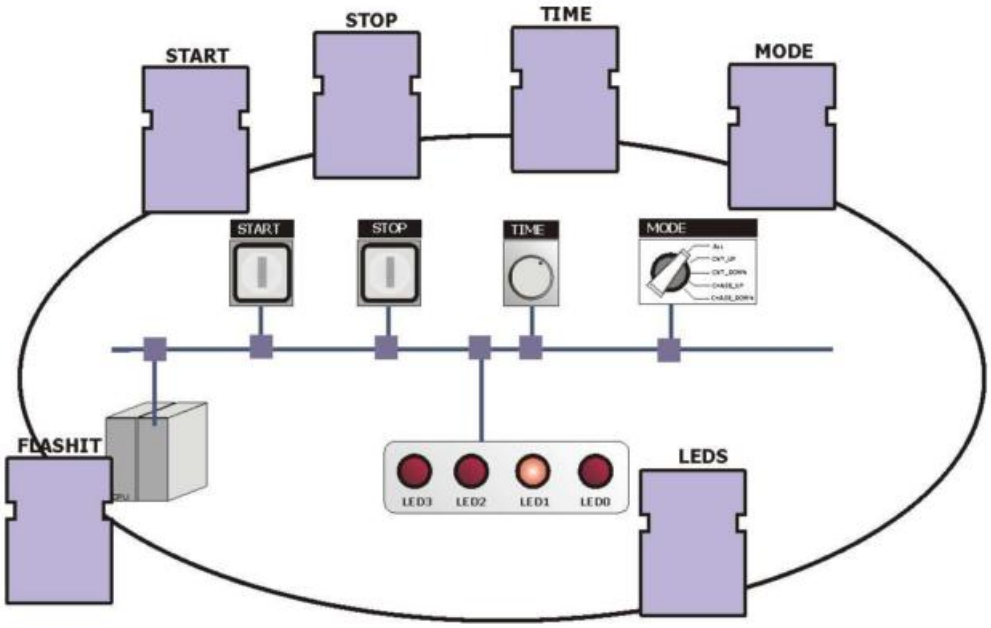


Рис.33 Распределенная сетевая АСУТП 4-го поколения

АСУТП 4-го поколения отличается тем, что все устройства: датчики, исполнительные механизмы, контроллеры,- имеют встроенные чипы и обмениваются цифровым способом по полевым шинам[12]. Верхний уровень схемы АСУТП на функциональных блоках рекомендуется делать таким образом, чтобы отражать физическую структуру системы. На рисунке:

- Датчиками здесь являются устройства ввода: START, STOP, TIME, MODE;
- Исполнительным механизмом табло подсветок: LEDS;
- Контроллер реализован в модуле FLASHIT.

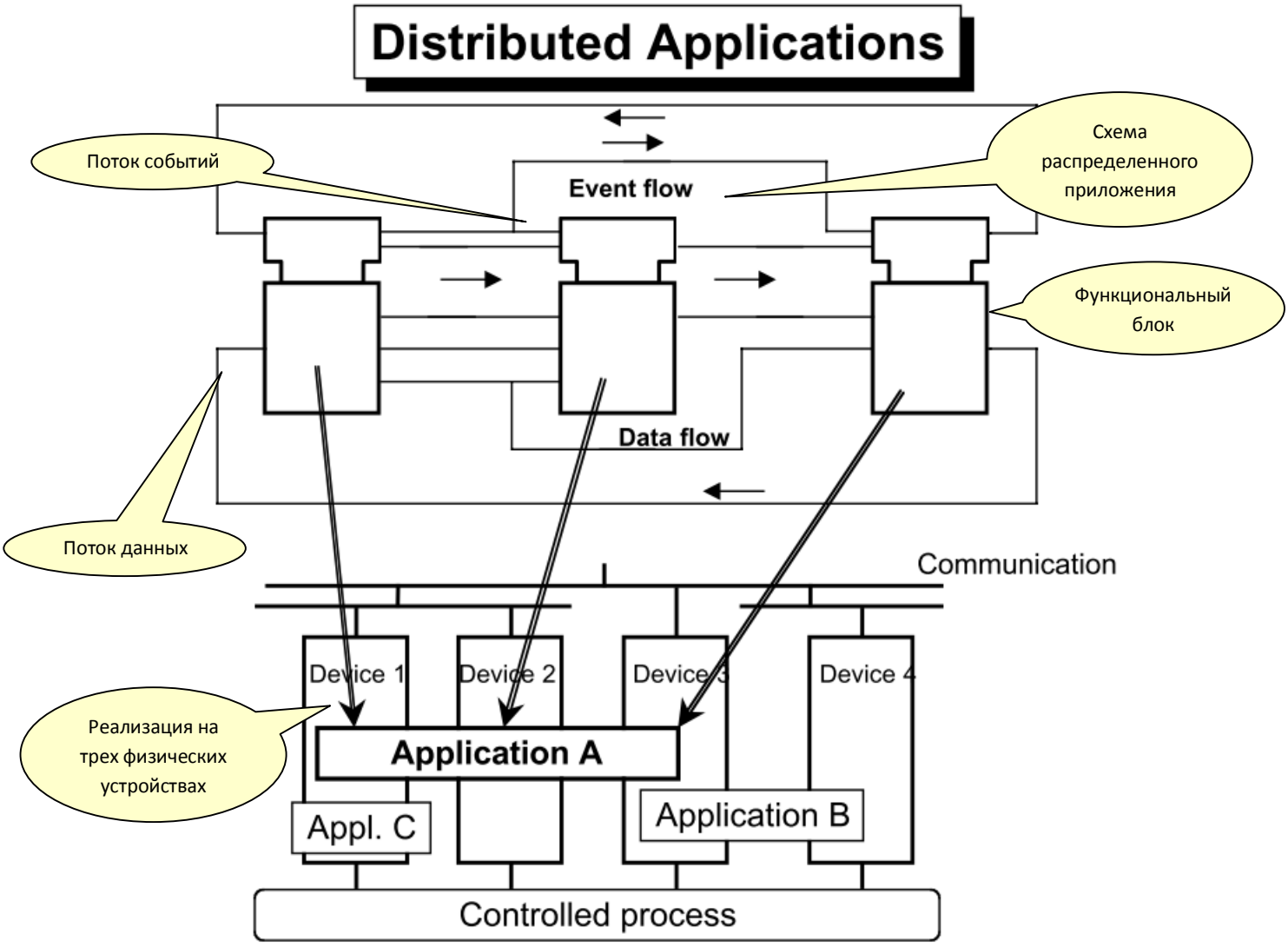


Рис.34 Проекция схемы из функциональных блоков на физическую систему

В распределенных приложениях один функциональный блок располагается только на одном физическом устройстве. Может быть несколько блоков, в том числе вложенных, на одном устройстве[13]. На рис.34 показано три таких блока.

Обмен данными происходит между функциональными блоками. Не между функциями и процедурами. Обмен построен на передаче событий и данных от одного блока к другим в соответствии с коммуникацией. Я напоминаю, что в рассмотренной ранее программной модели функциональные блоки представляли собой экземпляры объектов определенной типа. Теперь эти экземпляры могут принимать сообщения.

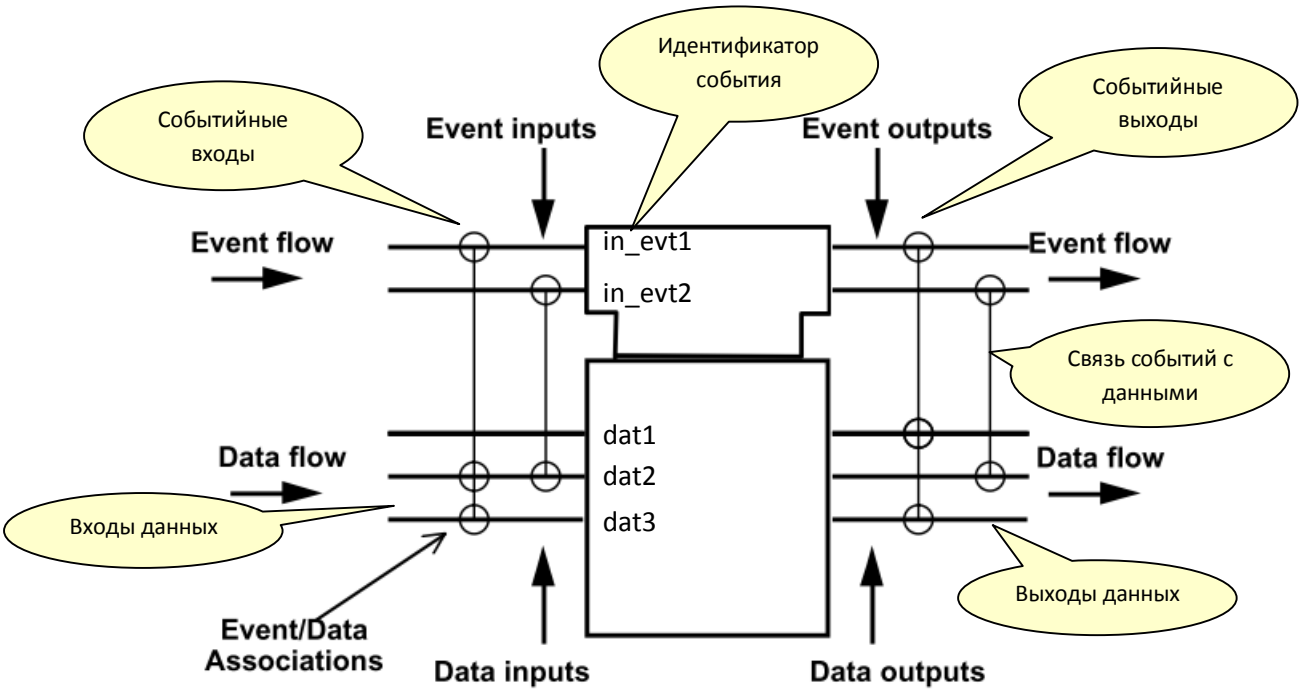


Рис.35 Интерфейсы событий и данных функционального блока

Общее обозначение функционального блока состоит из:

- Событийных входов;
- Входов данных;
- Событийных выходов;
- Выходов данных.

События представляют собой сообщения, содержащие идентификатор (такой как in_evt1) и набор ассоциированных с событием данных (для in_evt1 это - dat2, dat3). Выходные события инициируются функциональным блоком. Тогда передается сообщение связанным с данным событием объектам, содержащее выходное событие и набор ассоциированных данных.

Программа, реализующая работу функциональных блоков – это прикладной уровень. Есть приложение, которое получает данные и сообщения и обновляет данные и сообщения. С кем оно работает? С коммуникационной программой. Коммуникационные программы представляют собой отдельную единицу в устройстве.

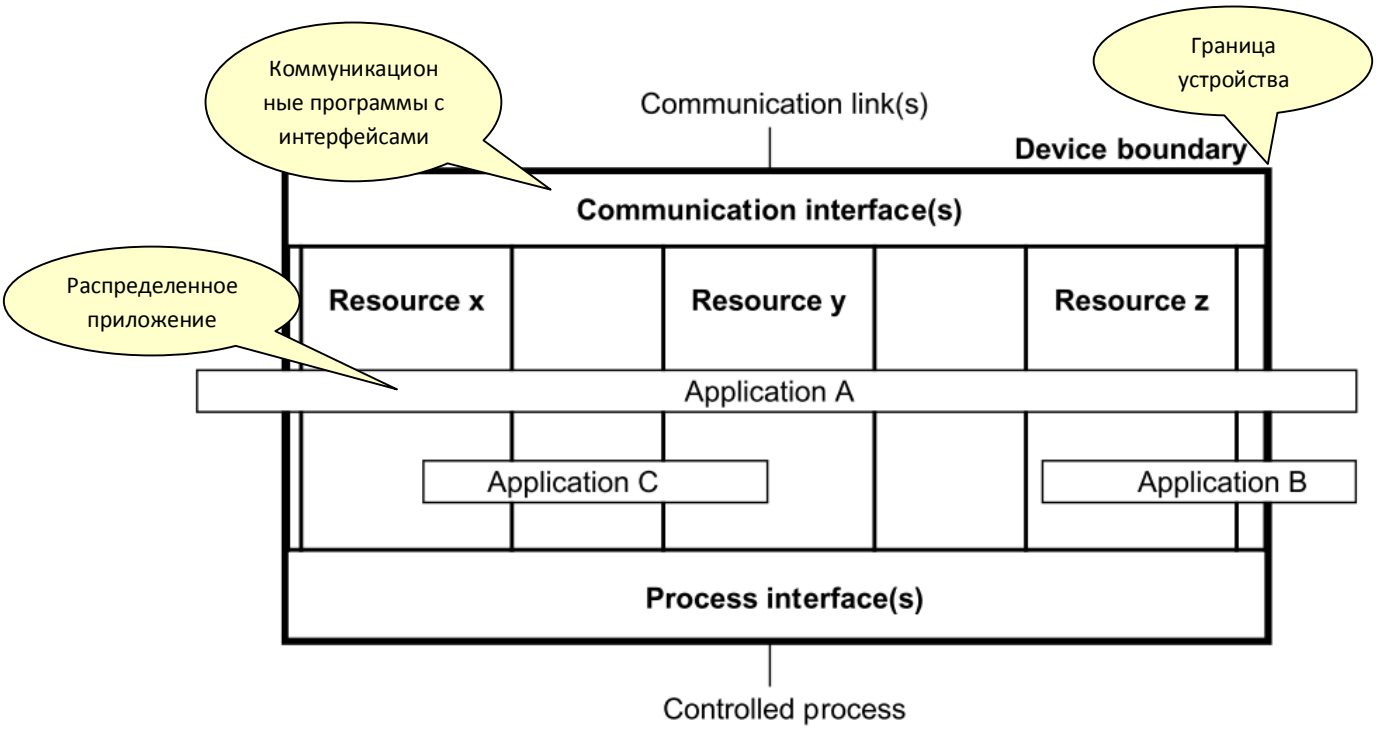


Рис.36 Модель устройства по стандарту МЭК 61499

Согласно стандарту 61499 коммуникационные программы входят в физические устройства как отдельные сущности. Со своими интерфейсами. Стандарт эти интерфейсы не описывает.

4.2. Механизмы коммуникации FieldBus Foundation

Наш девиз:
«Связь без брака!»

Здесь мы рассмотрим подход, выработанный организацией FieldBus Foundation[14]. Это – наиболее популярный и продвинутый подход реализации распределенных систем на функциональных блоках. Разработаны специальные полевые шины H1 и HSE. Большое количество производителей цифровых устройств поддерживает стандарты сетей и драйверы для этих шин. Наиболее предназначенная для реального времени шина H1 основана на том, что в сеть включаются специальные устройства – задатчики связей (LM – Link Master).

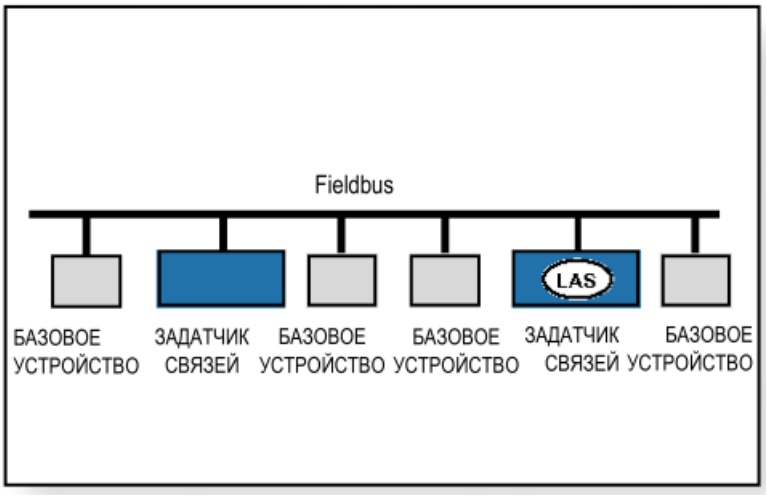


Рис.37 Включение в сеть задатчиков связей

Задатчик связей – это физическое устройство. Устройство может взять на себя функцию Активного Планировщика Связей (АПС или LAS – Links Active Scheduler).

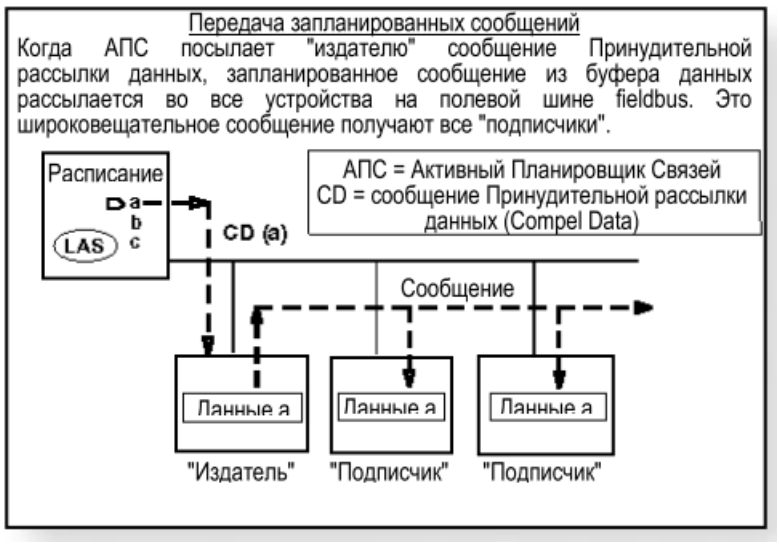


Рис.38 Работа Активного Планировщика Связей

Активный планировщик связей управляет процессом передаче по шине. Все устройства-абоненты подчиняются этой дисциплине. По команде данные и события от функционального блока передаются всем подписчикам широковещательной посылкой.

Обмен данными и событиями технологического процесса происходит по модели Публикации/Подписки. Устройство, выдающее в сеть данные, является издателем. Оно публикует данные (возможно, широковещательной посылкой). Устройство, принимающее из сети данные, является подписчиком (Рис.39).

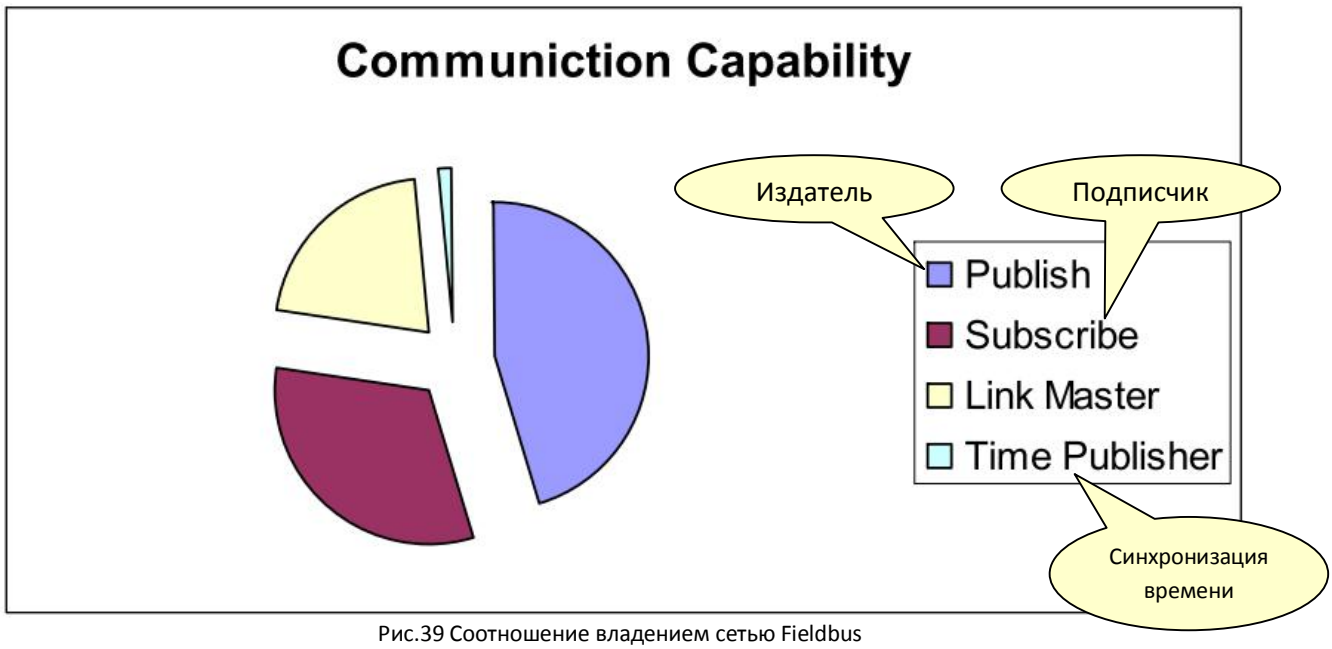


Рис.39 Соотношение владением сетью Fieldbus

С учетом планировщика Link Master все устройства получают контролируемый доступ к ресурсам сети. Устройства в нужный момент отправляют свои события данные и получают чужие.

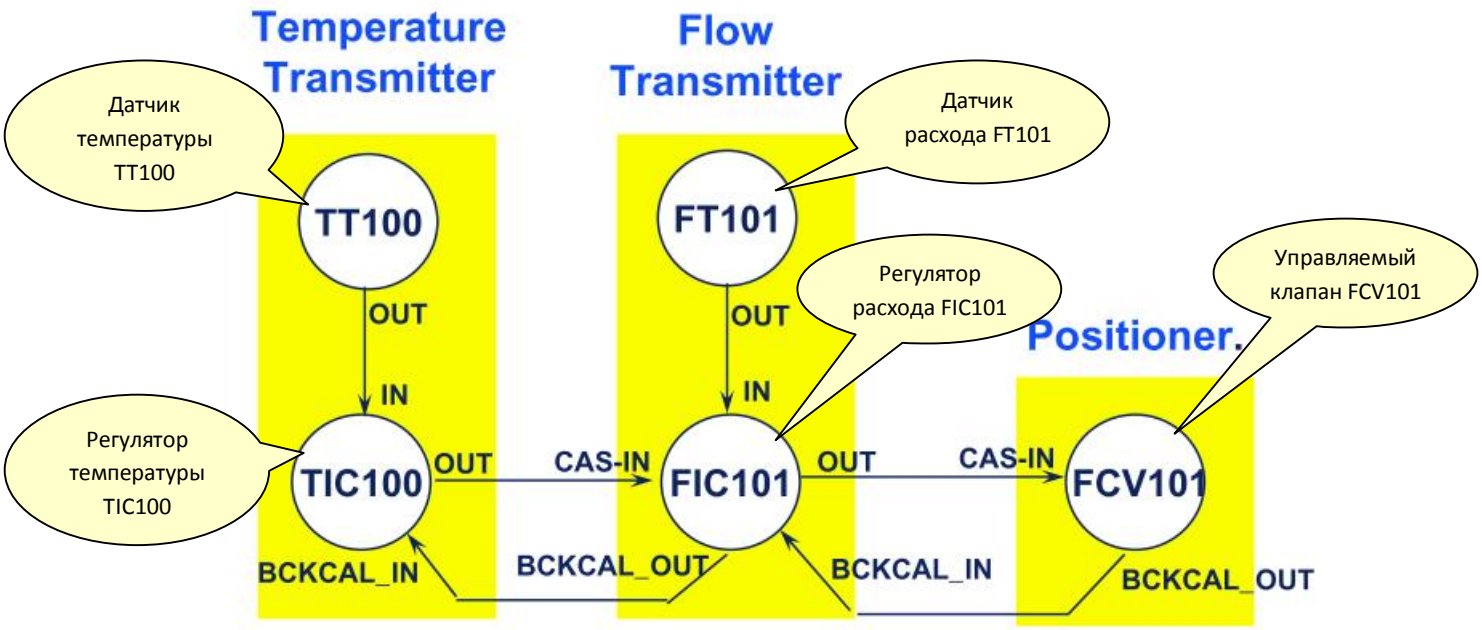


Рис.40 АСУТП с контролем температуры, расхода и исполнительным механизмом

На рис.40 представлена цифровая АСУТП из трех модулей: датчик+регулятор температуры, датчик+регулятор расхода, исполнительный механизм. Поддерживает температуру и расход воды.

Обмен данными под управлением планировщика для этой цифровой АСУТП представлена на рисунке. 41.

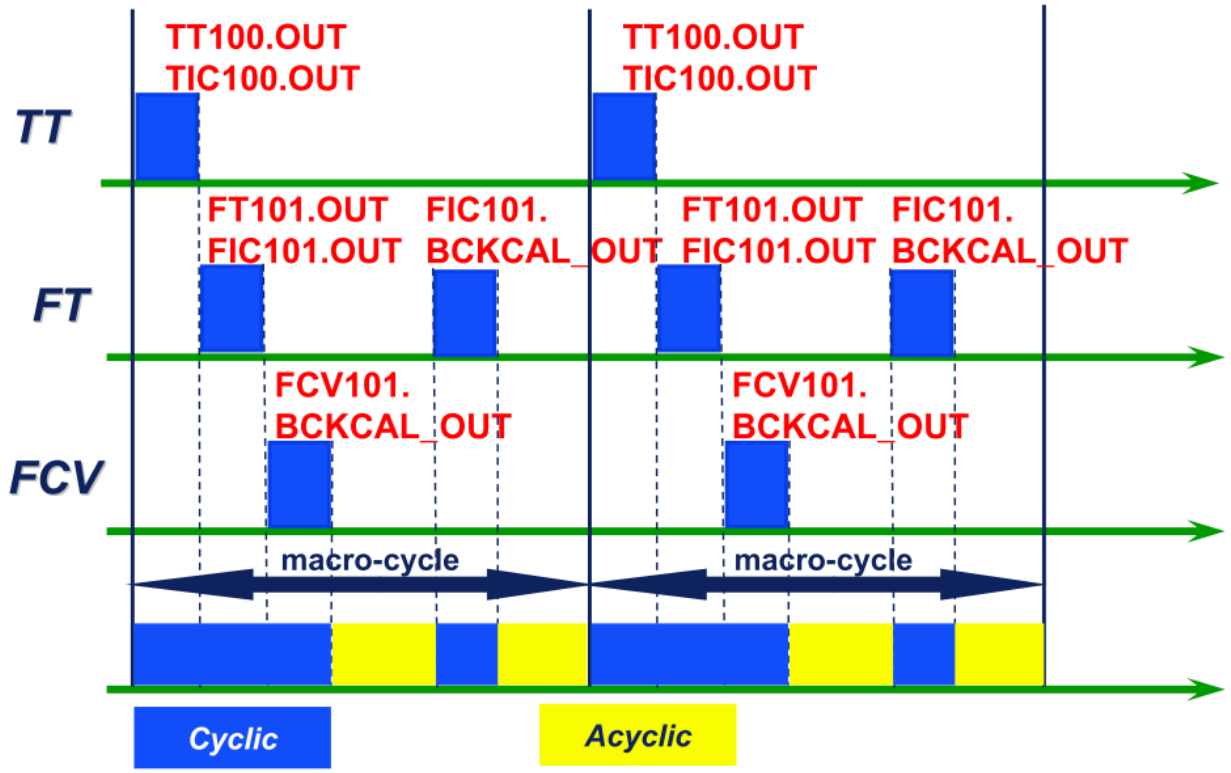


Рис.41 Обмен данными АСУТП рисунка 40

Планировщик LM организовывает обмен данными между устройствами. Каждое устройство передает измененные данные в строго заданной фазе цикла. Это – процедура обмена в жестком реальном времени. Разумеется, операционные системы всех устройств в узлах сети H1 должны обеспечивать жесткое реальное время.

С точки зрения программной реализации данные передаются и получаются уже от операционной системы в виде буферов. Эти буферы преобразуются в сообщения для функциональных блоков. Сообщения содержат наборы данных для выводов функциональных блоков. Индивидуально для каждого входа/события это будет либо атомарный тип (BOOLEAN, INTEGER, REAL), либо структура. Fieldbus имеет предопределенный набор структур. Ниже в таблице представлена DS-68, устанавливающая диапазоны измерений, индекс измеряемой величины и число значащих цифр после запятой.

E	Element Name	Data Type	Size
1	EU at 100%	Float	4
2	EU at 0%	Float	4
3	Units Index	Unsigned16	2
4	Decimal Point	Integer8	1

Диапазон измерений

Число значащих цифр

4.3. Схемы системы контроля

«Под всякой бездной раскрывается другая,
еще более глубокая»
Закон бесконечного падения Эмерсона

Я дальше буду предполагать, что имеется операционная система реального времени со всеми устройствами и драйверами Fieldbus Foundation. Это может быть система ОСРВ XOberon [15]. Я детально не знаю, были под нее разработки в части протоколов Fieldbus Foundation. Если это будет другой протокол, годится. Я еще раз подчеркну, что Foundation Fieldbus – не единственный способ реализации. Буферы данных состоят из сообщений, передаваемых между функциональными блоками.

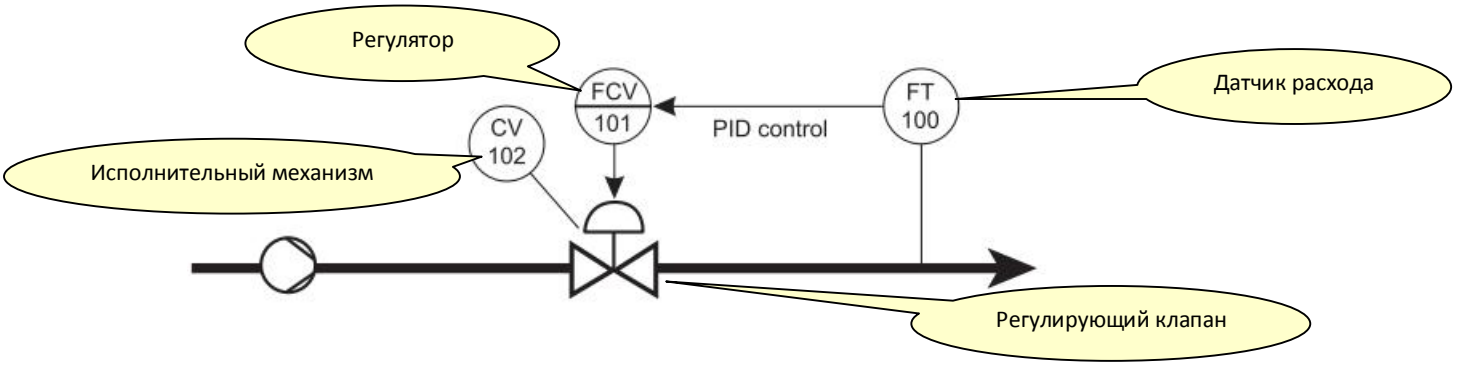


Рис.42 Система управления расходом

Мы будем создавать алгоритмы системы поддержание заданного расхода воды(Рис.42). В регуляторе задана уставка – это требуемое значение расхода. Если датчик дает сигнал меньше уставки, блок управления механизмом воздействует на клапан, и тот открывается.

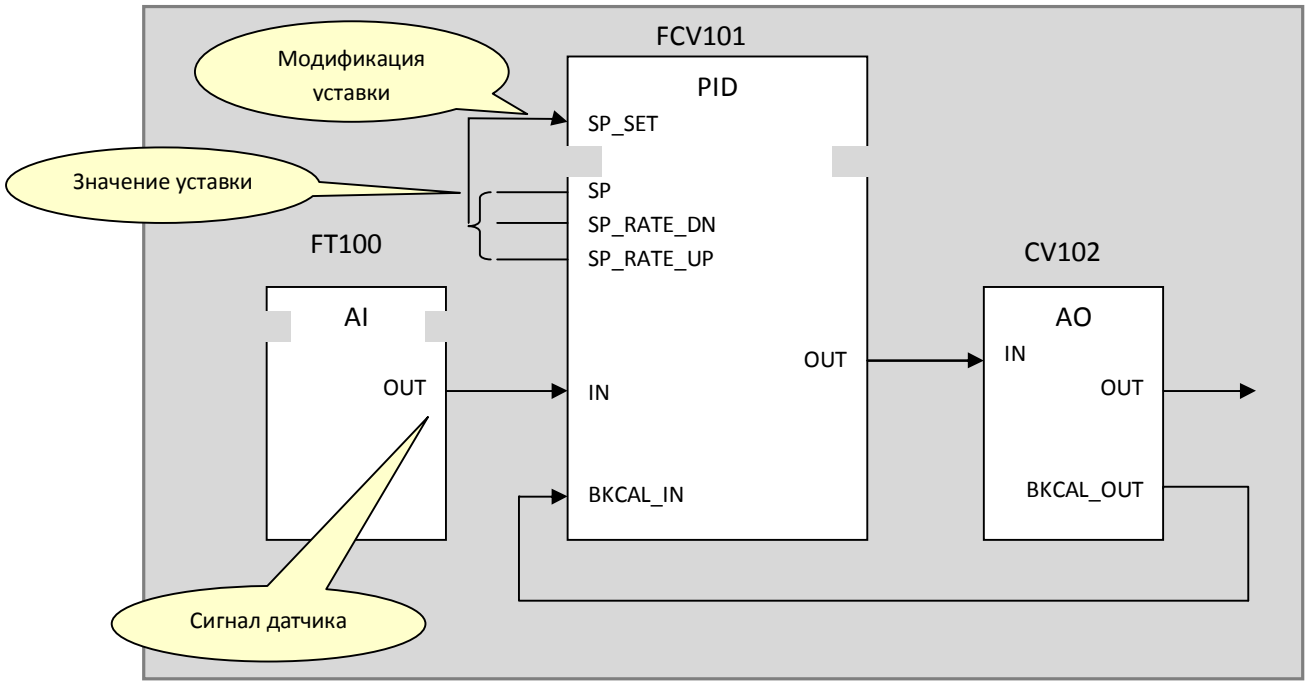


Рис.43 Схема системы управления расходом

На рисунке 43 приведена схема управления расходом из трех функциональных блоков: датчика расхода AI, регулятора PID, управления исполнительным механизмом AO.

Датчик AI FT100 означает, что в системе установлен функциональный блок типа AI в одном экземпляре с именем FT100. Выход AI.OUT выдает действительное число, равное значению показания датчика в физических единицах. Например, 5.2 т/ч. Регулятор сравнивает это с уставкой и формирует сигнал PID.OUT на блок управления AO. Считаем, что блоки AI, AO реализованы на текстовом языке. А регулятор PID реализован на визуальном языке Дракон (рис.44).

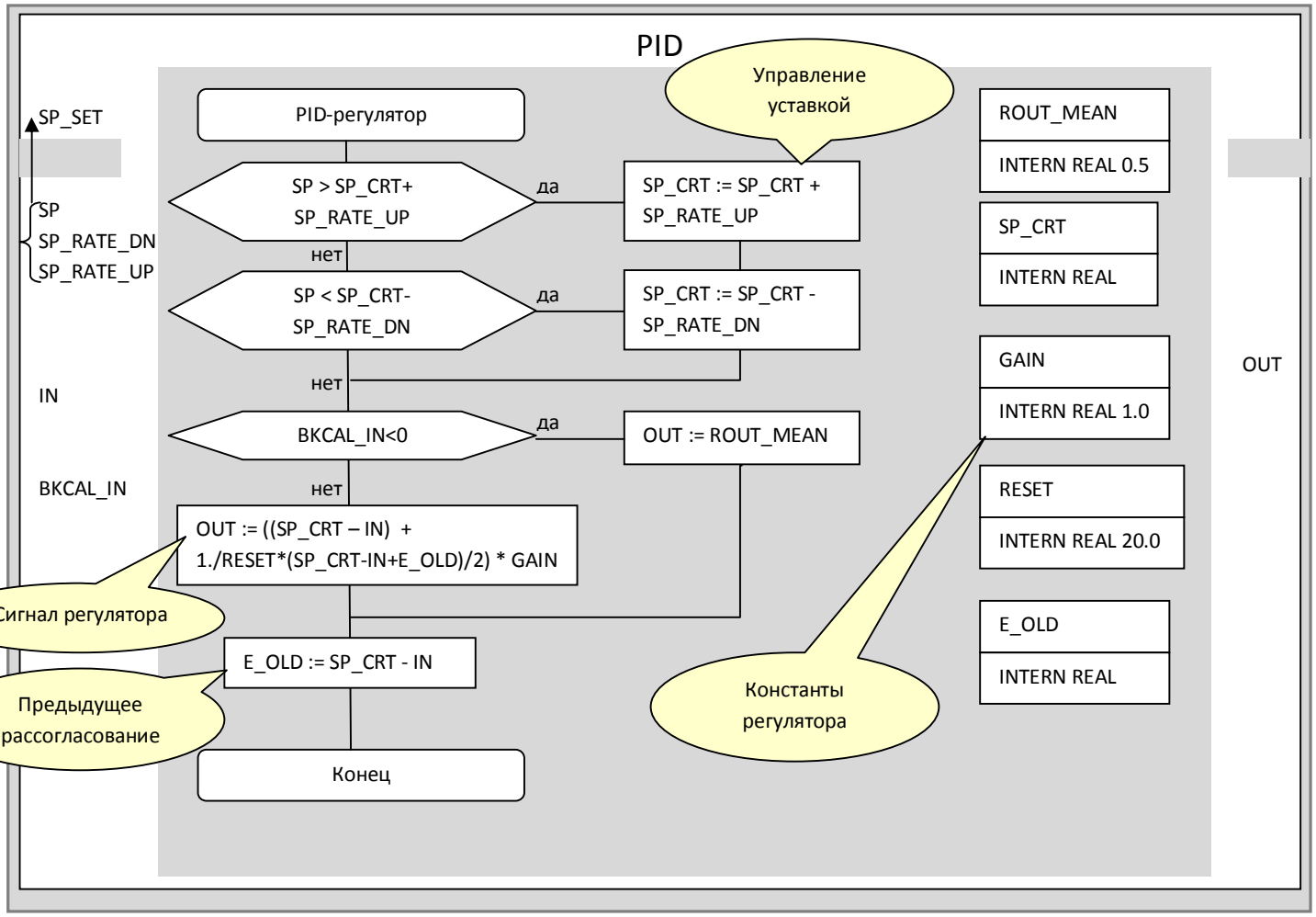


Рис.44 Реализация PID-регулятора на Драконе

Реализуем алгоритм PID-регулятора на Драконе аналогично описанному в [16]. Внутренняя переменная SP_CRT сохраняет текущее значение уставки. Если уставка SP увеличилась, то реализуется схема «безударного» включения. Текущее значение уставки увеличивается на величину SP_RATE_UP на каждом цикле. Или уменьшается на величину SP_RATE_DN.

Если сигнал обратной связи отрицательный, что показывает отключение исполнительного механизма, выходной сигнал устанавливается в половинное значение, задаваемое константой ROUT_MEAN.

Регулятор управляется сигналом рассогласования (SP_CRT-IN). При работающем исполнительном механизме выходной сигнал формируется по ПИ-закону. Характеристики регулятора задают константы GAIN, RESET. Предыдущее значение рассогласования сохраняется в переменной E_OLD.

Входные и выходные переменные соответствуют левым и правым контактам раскрытого элемента в масштабируемом интерфейсе. Элементы ПОЛКА для них не показаны.

Псевдоязык, реализующий алгоритм регулятора, имеет аналогичный формат (Рис.45).

```
ЗАГОЛОВОК PID «Регулятор PID»
ПОЛКА x1 –vars SP_SET –assoc {SP SP_RATE_DN SP_RATE_UP} –type EVENT –descr
{«Модификация уставки»} –kind input_event
ПОЛКА x2 –vars SP –type REAL –kind input
ПОЛКА x3 –vars SP_RATE_DN –type REAL –kind input
ПОЛКА x4 –vars SP_RATE_UP –type REAL –kind input
ПОЛКА x5 –vars IN –type REAL –kind input –descr «Сигнал датчика»
ПОЛКА x6 –vars BKCAL_IN –type REAL –kind input –descr «Обратная связь»
ПОЛКА x7 –vars OUT –type REAL –kind output –descr «Выходной сигнал регулятора»
ПОЛКА x8 –vars ROUT_MEAN –type REAL –kind internal –val 0.5
ПОЛКА x9 –vars SP_CRT –type REAL –kind internal
ПОЛКА x10 –vars GAIN –type REAL –kind internal –val 1.0 –descr «Коэффициент усиления»
ПОЛКА x11 –vars RESET –type REAL –kind internal –val 20.0 –descr «Интегральный
коэффициент»
ПОЛКА x12 –vars E_OLD –type REAL –kind internal
ВОПРОС o1 –command {SP > SP_CRT + SP_RATE_UP}
o1 goto o2 o3
ДЕЙСТВИЕ o2 –command {SP_CRT := SP_CRT + SP_RATE_UP}
o2 goto o4
ВОПРОС o3 –command {SP < SP_CRT – SP_RATE_DN}
o3 goto o4 o5
ДЕЙСТВИЕ o4 –command {SP_CRT := SP_CRT – SP_RATE_DN}
o4 goto o5
ВОПРОС o5 –command {BKCAL_IN < 0}
o5 goto o6 o7
ДЕЙСТВИЕ o6 –command {OUT := ROUT_MEAN}
o6 goto o8
ДЕЙСТВИЕ o7 –command {OUT := ((SP_CRT–IN) + 1./RESET*(SP_CRT–IN+E_OLD)/2)*GAIN}
o6 goto o8
ДЕЙСТВИЕ o8 –command {E_OLD := SP_CRT–IN}
O8 goto o9
КОНЕЦ o9
```

Событийные входы

Константы регулятора

Сигнал регулятора

Рис.45 Псевдоязык для PID-регулятора

В тексте псевдоязыка содержатся описания интерфейсов связи функционального блока.

- Событийный вход SP_SET имеет вид event и ассоциирован с переменными SP SP_RATE_DN SP_RATE_UP;
- Входные переменные SP SP_RATE_DN SP_RATE_UP IN BKCAL_IN имеют вид input;
- Выходная переменная OUT имеет вид output;
- Внутренние переменные имеют вид internal. Переменным ROUT_MEAN GAIN RESET заданы начальные значения.

Программная реализация будет выглядеть как на рисунке 46.

```
(* Сообщения PID *)
PID_SP_SET_Message = RECORD(Message)
       REAL SP;
       REAL SP_RATE_DN;
       REAL SP_RATE_UP;
END;
(* Тип PID *)
TYPE PID_p POINTER TO PID;
PID = RECORD(FunctionalBlock)
       (* Inputs *)
       REAL SP;
       REAL SP_RATE_DN;
       REAL SP_RATE_UP;
       REAL IN;
       REAL BKCAL_IN;
       (* Outputs *)
       REAL OUT;
       (* Internal *)
       REAL ROUT_MEAN;
       REAL SP_CRT;
       REAL GAIN;
       REAL RESET;
       REAL E_OLD;
END;
PROCEDURE PID_init(f: FB_p);
       VAR r: PID_p;
BEGIN
       r:= f(PID_p);
       r.ROUT_MEAN := 0.5;
       r.GAIN := 1.0;
       r.RESET := 20.0;
END PID_init
PROCEDURE PID_handle(f: FB_p, VAR msg: Message);
       VAR r: PID_p;
BEGIN
       r:= f(PID_p);
       IF msg IS PID_SP_SET_Message THEN
              r.SP := msg(PID_SP_SET_Message).SP;
              r.SP_RATE_DN := msg(PID_SP_SET_Message).SP_RATE_DN;
              r.SP_RATE_UP := msg(PID_SP_SET_Message).SP_RATE_UP;
              r.PID_update(f);
       END
END PID_handle
PROCEDURE PID_update(f: FB_p);
       (*Регулятор PID*)
       VAR r: PID_p;
BEGIN
       r:= f(PID_p);
       IF r.SP > r.SP_CRT + r.SP_RATE_UP THEN
              r.SP_CRT := r.SP_CRT + r.SP_RATE_UP
       ELSE
              IF r.SP < r.SP_CRT - r.SP_RATE_DN THEN
                     r.SP_CRT := r.SP_CRT - r.SP_RATE_DN
              END
       END;
       IF r.BKCAL_IN < 0 THEN
              r.OUT := r.ROUT_MEAN
       ELSE
              r.OUT := ((r.SP_CRT-r.IN) + 1./r.RESET*(r.SP_CRT-r.IN+r.E_OLD)/2)*r.GAIN
```

Сообщения для событийных входов

Тип объекта PID

Инициализация начальных значений

Обработчик события SP_SET

Алгоритм Дракон-схемы

```
      END;
      r.E_OLD := r.SP_CRT-r.IN
END PID_update
(* Инсталляция экземпляра PID *)
VAR FCV101: PID;
FCV101.update := PID_update;
FCV101.init := PID_init;
FCV101.handle := PID_handle;
```

Нужно инсталлировать объект FCV101

Рис.46 Программная реализация функционального блока PID

Начальные значения будут установлены, когда вызовется функция FCV101.init(FCV101).

Данные приходят от коммуникационной системы в виде посылок определенной структуры. Это – либо сообщения message, либо входные для функциональных блоков данные, передаваемые по подписке.

При приходе сообщения message от коммуникационной системы, адресованного FCV101, вызывается FCV101.handle(FCV101, message). Пока обрабатывается только событие SP_SET. Формат сообщения при этом должен содержать поля SP SP_RATE_UP SP_RATE_DN. При добавлении других событийных входов будут обрабатываться другие сообщения.

При поступлении входных данных производятся обновления на уровне схемы:

```
FCV101.IN := FT100.OUT;

FCV101.BKCAL_IN := CV102.BKCAL_OUT;
```

Этот код обрабатывается на уровне схемы привязки функциональных блоков. Он получен из этой схемы рис.43. При поступлении данных происходит обновление значений входов блока FCV101. Далее, вызывается процедура FCV101.update(FCV101) в цикле реального времени.

Динамически изменяемые значения должны отображаться прямо на схемах.

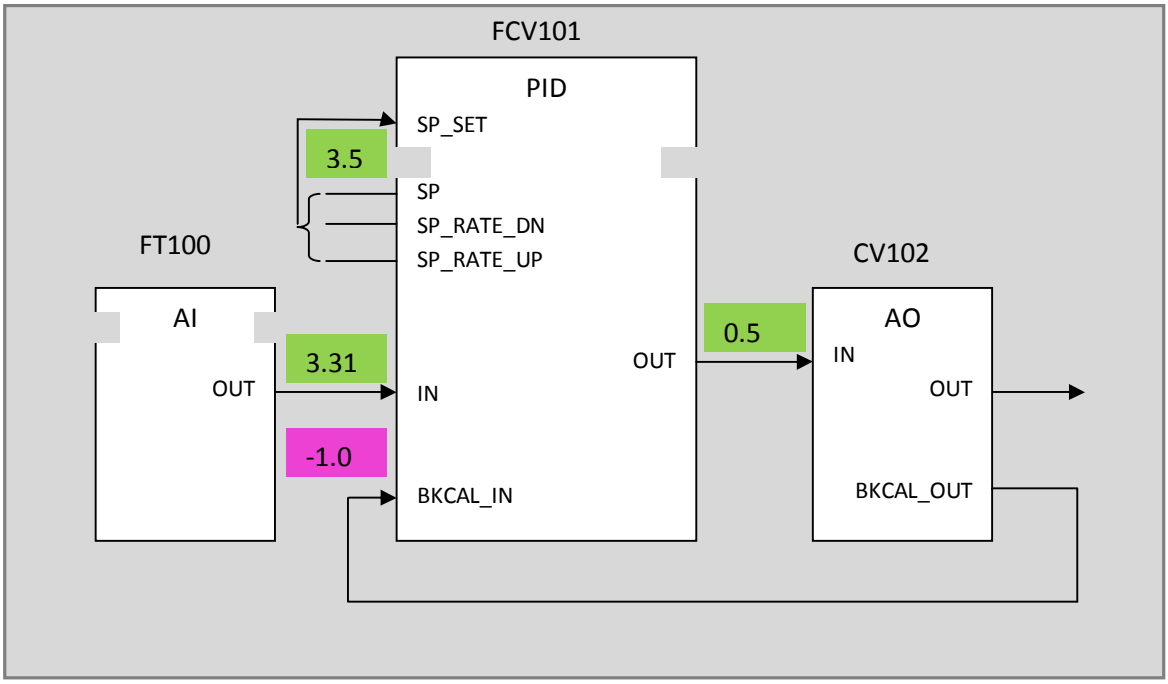


Рис.47 Данные на схеме управления расходом

На рисунке 47 приведено отображение данных непосредственно на схеме управления расходом. Зеленым подсвечены данные, имеющие нормальный статус. Такие, как величина датчика и уставка IN, SP. Фиолетово-малиновым (magenta) подсвечены недостоверные данные (BKCAL_IN). Для реализации отображения данных требуется включать в состав схемы SCADA-систему. Такие системы имеют графический интерфейс и предназначены для взаимодействия системы управления с оператором. Задача отображения аналогична ранее рассмотренным: нужно загрузить схему и осуществить привязку приходящих по подписке данных к графическим объектам на схеме.

Система разработки получает данные от удаленных узлов распределенной системы по сетям реального времени. Разработчик алгоритмов (Дракон-схем) на своем рабочем месте редко имеет возможность отладки прямо на распределенной системе. Для этого делаются эмуляторы работы функциональных блоков, запускаемые прямо на машине разработчика. Эти эмуляторы не обязательно должны генерить код на Обероне. Наоборот, удобнее скрипт в виде интерпретатора или байтового компилятора. Тот же самый скрипт. Есть аналогичные разработки эмуляторов функциональных блоков на Java за рубежом[17].

Принцип 6. Создайте ему рабочее место как в распределенной сети.

Выводы

Итак, было продемонстрированы пять классов реальных систем, которые можно построить на визуальных языках: Дракон и FBD. Структура таких реальных систем 5-ти уровневая. Нужно иметь фундамент в виде системного и архитектурного уровней. Необходимо иметь графическую оболочку и скрипт с псевдоязыком для работы с визуальными данными как основу для дальнейшего усовершенствования. Можно разработать 3-й, прикладной уровень этих систем. И тогда когнитивные технологии – Ваши!

И не забудьте принципы: что Вы предложете Пользователю.

1. Предоставьте всю необходимую информацию в поле зрения пользователя.
2. Выполните за человека всю рутинную и непосильную для него работу.
3. Структурируйте схемы в соответствии с категориями, которыми мыслит человек.
4. Предоставьте хорошие средства создания твердых копий.
5. Сделайте ему масштабируемый интерфейс доступа к данным на всех уровнях.
6. Создайте ему рабочее место как в распределенной сети.

Список литературы

1. В.Д.Паронджанов. Как улучшить работу ума? М.: 2001
2. В.Д.Паронджанов. Язык Дракон. Краткое описание.
3. В.Д.Паронджанов. Дружелюбные алгоритмы, понятные каждому. М: 2010
4. В.Д.Паронджанов. Почему мудрец похож на обезьяну. М: 2007
5. Д.В.Дагаев. Алаверды Дракону. http://forum.oberoncore.ru/download/file.php?id=2298
6. Д.В.Дагаев. Два пернатых в одной берлоге не уживутся?. http://forum.oberoncore.ru/download/file.php?id=2336
7. Geocoding with PHP and the Google Maps API. http://habrahabr.ru/blogs/php/31424/
8. N.Wirth. Programming in Oberon. A derivative programming in Modula-2 (2004)
9. Ф.Хейес-Рот, Д.Уотерман, Д.Ленат. Построение экспертных систем. М: 1987
10. Джордж Ф.Люгер. Искусственный интеллект. М: 2003.
11. И.В.Петров. Программируемые контроллеры. Стандартные языки и инструменты. М.: 2003
12. Валерий Вяткин. IEC 61499. Function Blocks for Embedded and Distributed Control Systems Design
13. J.H.Christensen. International standarts for open distributed automation. 2000.
14. Полевая шина FOUNDATION Fieldbus, Технический обзор. FD-043, версия 2.0.
15. ОСРВ XOberon. http://www.ifr.mavt.ethz.ch/research/xoberon/
16. Guideline: FOUNDATION Fieldbus Function Blocks. Version 1.01.
17. Function Block Development Kit. http://www.holobloc.com