Два пернатых в одной берлоге не уживутся?

Дмитрий Дагаев

Оглавление

Предисловие

С богом у него очень неопределенные отношения,
но иногда они напоминают мне отношения
«двух медведей в одной берлоге»,
Горький о Льве Толстом

Когда-то генерал Александр Лебедь, которому Ельцин хотел дать «в помощники» другого генерала, Павла Грачёва, возразил: – *Два перрнатых в одной берррлоге не уживутся*! Замечательный способ решения, однако.

Надо дать возможность каждому *драконоводу*, каждому творцу по скрипке, и предусмотреть, чтобы не было слышно из соседней берлоги. Только скрипки нужны такие, которые в руках Мастера *заиграют*. А для этого нужно их сделать. Вылезти из своих берлог и на стенах их, как на изумрудных скрижалях написать, какие они будут, эти инструменты, чтобы можно было на них *творить*. Потом создать инструменты, раздать желающим и продолжить каждому процесс *служения муз* в тиши его берлоги.

Если Оберон-технологии такие надежные, а Дракон-технологии такие перспективные, где же ПО?

Не пора ли сделать качественные графические средства для новых графических технологий?

***

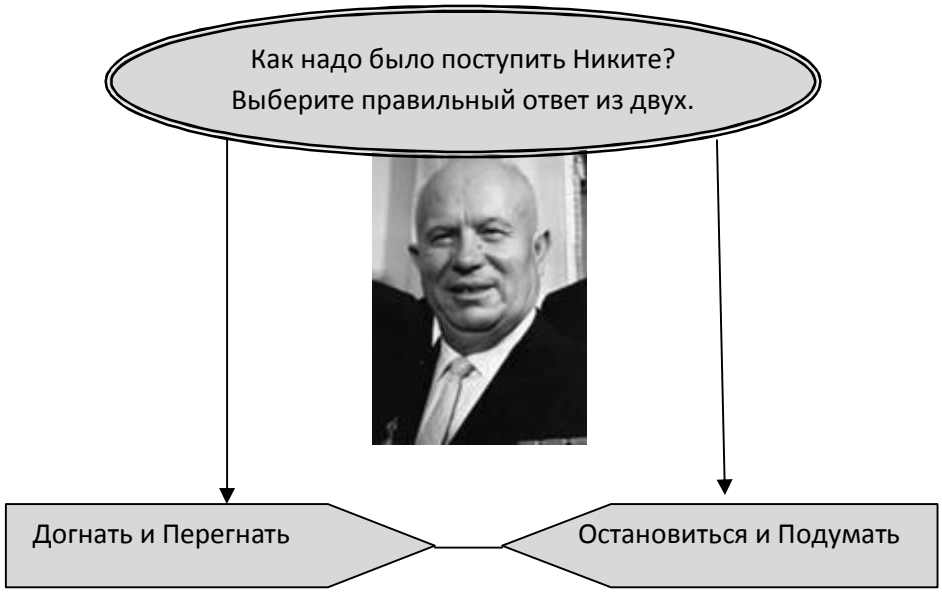
Дагаев Дмитрий Викторович dvdagaev@mail.ru с 1987 года в атомной энегетике занимается разработкой и внедрением систем управления, искусственного интеллекта и моделирования на основе графических форм представления информации. В разное время были разработаны модели систем управления для тренажеров на основе 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

«Древние предпочитали промолчать,
Стыдясь, что могут не поспеть за словом»
Конфуций

На всесоюзной выставке в Сокольниках у Н.С.Хрущева состоялась знаменитая словесная перепалка с Никсоном. Баталия достигла высшего накала, когда гости зашли на кухню «американского дома». Хрущев пнул стиральную машину и заявил, что «в жизни все это не нужно». Никсон ответил, что у гражданина СССР нет возможности самостоятельно решить, что ему нужно, потому что за него решают. Хрущев закричал, что советские люди добровольно отказываются от многих бытовых удобств ради того, чтобы у Родины было больше ракет. «Так, может, мы лучше начнем соревноваться в производстве стиральных машин, а не ракет?» – огрызнулся Никсон. Первый секретарь и вице-президент почти кричали, тыкали друг в друга пальцами и пихали ладонями в грудь. Свара происходила перед объективами телекамер, и посол США Тэд Томпсон стоял ни жив ни мертв. В этот момент наступила кульминация. Хрущев закричал: «Господин Никсон, ваши генералы хорохорятся, но мы им еще покажем, как говорят у нас в России, кузькину мать!» [1]

«Через 7 лет мы догоним Америку по всем показателям, – кричал Никита Сергеевич, – а потом обгоним и помашем вам ручкой!» Собственно вся страна потом занималась. Утомилась, зато расслабилась при следующем генсеке. Сейчас пока тоже не знает куда бежать.



Ответы: «Догнать и Перегнать» - 0 очков; «Остановиться и Подумать» - 20 очков.

Разумеется, нелегко начинать соревнования в отраслях, в которых уже дорожки протоптаны конкурентами. И если попытки не удались, попытаться понять. Может быть, есть ассиметричный ответ? Вопрос, в чем нужно догонять, а что нужно использовать не имеет простых ответов. Предлагаю свои, для обсуждения.

***

На сегодняшний день в сообществе имеет место *согласие* по двум пунктам:

1. Оберон – это хорошо;
2. Дракон – это хорошо.

Я *выношу на обсуждение* еще два пункта:

1. Нужен качественный графический пакет;
2. Нужен язык в пару к Оберону для построения больших систем.

1. Что мы хотим от графических пакетов?

Я как писатель-фантаст уверен, что этот чертов робот
Должен говорить на человеческом языке, а не наоборот
Спайдер Робинсон

Помню, решил я как-то воспользоваться для своей работы «стандартным подходом» в виде MS Visio. Четыре дня делал сложную схему с не менее сложной анимацией. В виде скриптов на Visual Basic for Applications. И почти все сделал, почти. Когда все рухнуло. И тексты программ все пропали, ибо записывались в бинарном виде по технологии Compound Document для MS Office. Я даже дошел до 16-ричного вьюера и обнаружил, что все мои 50 кбайт исходников случайно заполнились нулевыми байтами! И я понял, что это – знак. Не судьба с этими стандартными редакторами. Надо сделать хоть один хороший.

При оценке требований к графическим интерфейсам я буду основываться на фундаментальном труде Дж.Раскина «Интерфейс» [2]. Многие принципы взяты оттуда. Я их разделяю.

Определив задачу, для которой продукт предназначался, сначала спроектируйте интерфейс, а потом приступайте к реализации.

Я буду рассматривать некоторое множество задач, для которых Оберон-технологии будут иметь преимущество[3]: АСУТП и АСУ предприятием, тренажеры и системы моделирования, экспертные системы реального времени, информационные системы, системы слежения. И эти задачи как сфера приложений когнитивных технологий [4-7].

В конце концов, человеко-машинный интерфейс (ЧМИ) – лишь средство обмена данными от компьютера к человеку и наоборот. Данными точными и достоверными.

1.1. Интерфейсотехника Раскина

1. Робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинён вред.
2. Робот должен повиноваться всем приказам, которые дает человек, кроме тех случаев, когда эти приказы противоречат Первому Закону.
3. Робот должен заботиться о своей безопасности в той мере, в которой это не противоречит Первому и Второму Законам.
Айзек Азимов

Приведу удачное, на мой взгляд, перефразирование трех законов робототехники А.Азимова в интерпретации Раскина:

1-й закон. Компьютер не может причинить вред данным пользователя или своим бездействием допустить, чтобы данным был причинен вред.

2-й закон. Компьютер не должен впустую тратить Ваше время или вынуждать Вас выполнять действия сверх необходимых.

3-й закон. Интерфейс является ориентированным на человека, если он отвечает нуждам человека и учитывает его слабости.

Удовлетворяют ли интерфейсы тех систем, с которыми работает мы с Вами, этим требованиям? Устраивают они Вас? Ответ очевиден: нет.

В настоящее время графические интерфейсы ориентированы на приложения. Одни рассчитаны на просмотр текста, другие – на редактирование текста, третью – на просмотр графики… Разные приложения имеют разные наборы команд, и пользователь обычно не может использовать команды приложения А при работе с приложением В или наоборот. Представьте, что вдруг созданы отдельными людьми редакторы для языков ДРАКОН, МОЛНИЯ, ГНОМ, ГРАФ[7]. И еще для FBD, для технологического оборудования, для сильноточной электрики[3]. Не много ли редакторов? А как насчет анимирования этих схем в реальных задачах? У меня получается:

*Число программ-аниматоров: Число задач на одного программиста * Число программистов.*

При этом приложения расширяются до колоссальных размеров, поскольку каждое из них должно решать огромное множество задач, которые имеют второстепенной значение с точки зрения его основной задачи. Причем эти предоставляемые приложениями средства работают по-разному и имеют разные возможности в разных программах. Взаимодействие между приложениями не всегда возможно. При переходе от одного к другому Вы тратите драгоценное время на загрузку, на дурацкие открытия/закрытия файлов, на то, чтобы перестроиться на другой интерфейс. Может, стоит остановиться и подумать. А не догонять и перегонять?

Нужно ликвидировать приложения. Будет одна графическая программа.

Дж.Раскин считает, что человекоориентированное программное обеспечение не должно состоять из системы и набора приложений. Вместо этого, оно должно представлять собой «набор команд, некоторые из которых являются трансформаторами, автоматически вызываемыми в тех случаях, когда тип данных, предусмотренных командой, не соответствует типу данных выбранного объекта».

Приведу еще один пример к первому и третьему законам. Опубликовавшись [3] на форуме forum.oberoncore.ru, я зарегистрировался веду переписку по теме «Алаверды Дракону». Я человек новый на форуме, но с компьютером работал и раньше. Я набираю сообщение в форуме, затем нажимаю клавишу «Предпросмотр». Затем автоматически возвращаюсь к редактированию текста. Видимо, клавишей назад в браузере.

За 15 дней работы на форуме набранные мной данные пропали 2 раза. Один раз я почти страницу набрал. Несколько раз мне появлялось сообщение, что страничка просрочена. Такой вот *вражественный интерфейс*. Разумеется, этот *чертов робот заставил* меня говорить на его языке. И теперь я открываю notepad одновременно с форумом. И пишу текст сначала в notepad, потом копирую в форум.

1.2. На Вашем экране.

Бросая в воду камешки, смотри на круги, ими образуемые:
иначе такое бросание будет пустою забавой.
Козьма Прутков

Вот Вы получили экран на весь рабочий стол. Вы это себе представляете? Наверное, разработчики уже представляют, что *все это они уже сделали*? Вы выбираете мышью Дракон-схему и выполняете команду, относящуюся к этой схеме. Например, сконвертировать в программу. Или напечатать. Или делаете выборку из икон и выполняете команду, относящуюся к данному типу иконки. Или редактируете. *Вы уже работаете в одной графической среде, которая накрывает весь рабочий стол.* Это – обычная практика для рабочих станций тренажеров и АСУТП.

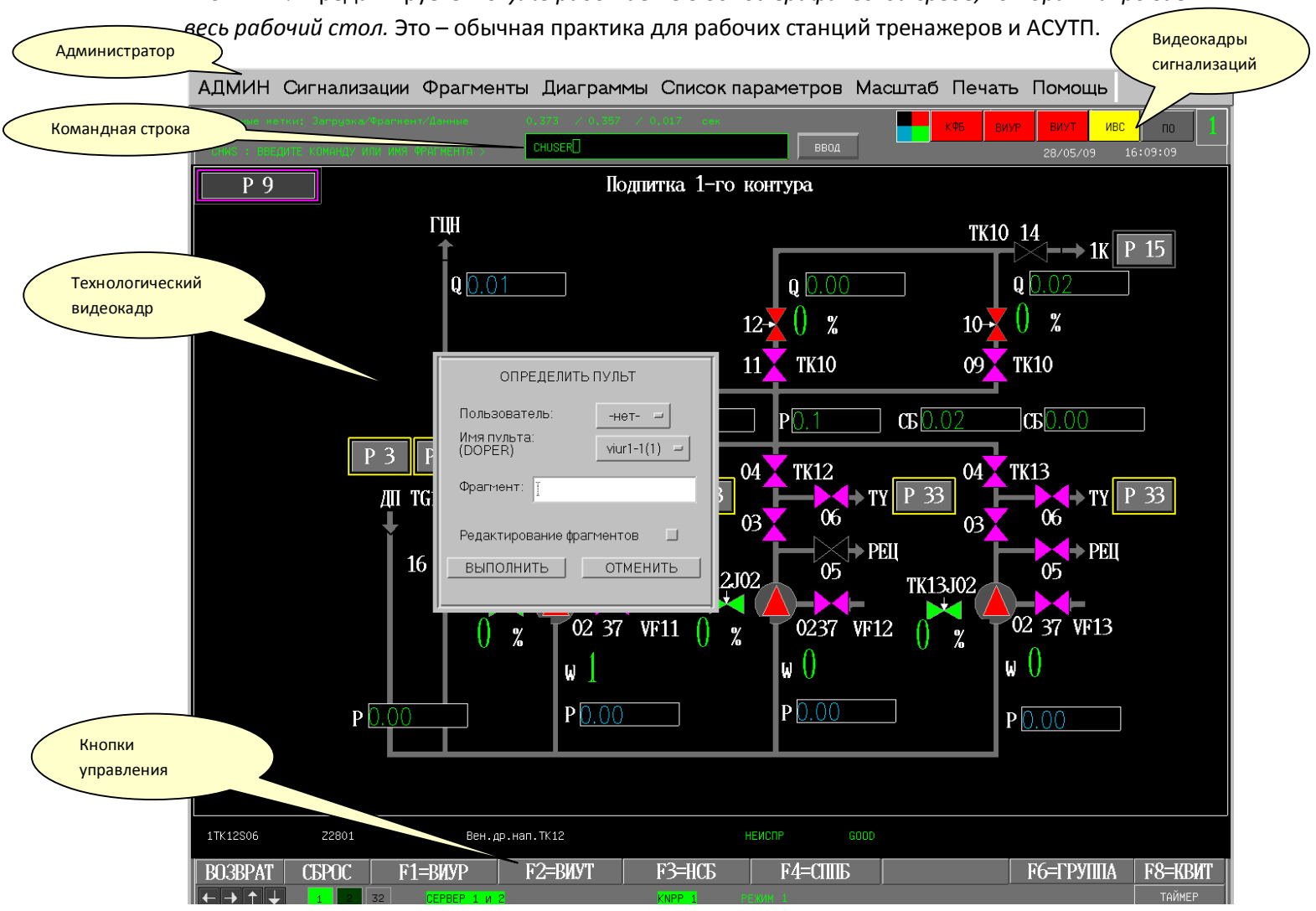


Рис. 1 Экран рабочей станции ИВС 1КАЭС

На рисунке 1 представлен экран рабочей станции системы информационно-вычислительной системы 1 блока Калининской АЭС. В середине – видеокадр с технологической схемой. На понятном оператору технологическом языке. Он меняется, если оператор выберет другой видеокадр. Всплывающее окно редактирует права по умолчанию на другую графическую станцию. Ибо Вы вошли «администратором». Остальное не меняется: внизу – кнопки управления, наверху – команды и сигнализации. И динамика привязана к реальным переменным процесса.

Ну как, все это разработчики уже сделали? А, Вы говорите о редакторе: так почему бы не отредактировать и технологическую схему, раз уж вошли администратором?

Наверное, не получится, ибо Ваше приложение для этого *не предназначено*. Так давайте *предназначим* новую графическую программу все это реализовывать! Вы возразите, нельзя сделать программу для решения *всех* задач. Правильно, и не нужно. Нужно сделать минимальную программу, но ясную и доступную. С возможностью подключения, обработчиков, возможностью понимания скриптов, возможностями расширения.

Начнем с простых вещей:

1. Графическая программа должна быть полностью конфигурируема в зависимости от задачи;
2. Графическая программа состоит из иерархической системы окон;
3. Окна графической программы могут редактировать и отображать текст, данные, Дракон-схемы, веб-страницы;
4. Меню в графической программе может и отсутствовать.

Я не уточняю, кто и как управляет конфигурацией графической программы. Важно, чтобы она была *полностью конфигурируема*. Программистом, который ее использует для своих конкретных задач. Пользователем, который подстраивает ее под себя. И окно с редактором Дракон-схемы разве сделать сложно? Окно с текстом или расширенным текстом все графические пакеты предоставляют.

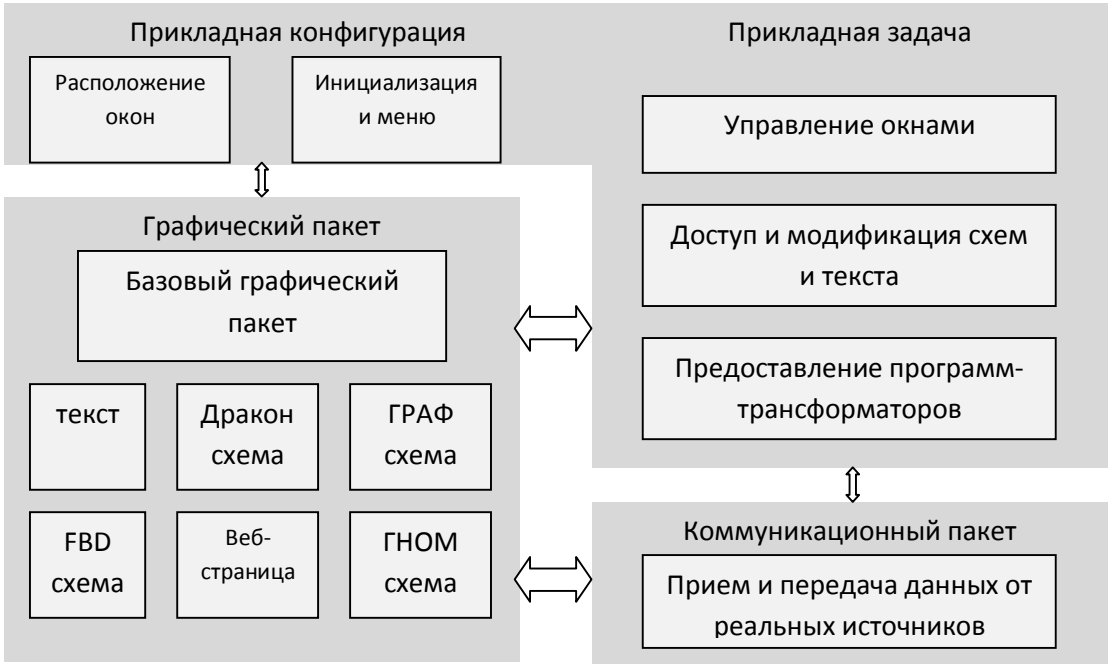


Рис. 2 Структура прикладной программы

На рис. 2 предлагается структура прикладной программы. Любой программы, входящей в класс рассматриваемых задач. Вашей программы, которую Вы написали бы, если бы имели такой графический пакет.

Графический пакет запущен. Если не все время, так достаточно долгое. Далее загружается Ваша задача и коммуникационный пакет. При загрузке сначала происходит формирование структуры расположения окон (layout), формирование меню и инициализация графических объектов начальными значениями. Начальные загружаемые схемы. Текст и страницы по умолчанию.

Далее Ваша прикладная программа начинает работать. Обновлять данные циклически. Взаимодействовать с оператором. Обратите внимание, что запуск прикладной программы происходит из той же графической оболочки. То же самое управление окнами. Даже, если этот запуск займет пару секунд, это будет незаметно для пользователя. Согласно Дж.Раскину, «программные продукты не должны вынуждать пользователя ждать без необходимости… Ритм взаимодействия должен устанавливаться самим пользователем».

Пользователь взаимодействует с графическими программами. Например, он может выбрать графический элемент на схеме для того, чтобы запросить по нему дополнительную информацию. Самое простое – выбрать правой кнопкой мыши «контекстное меню».

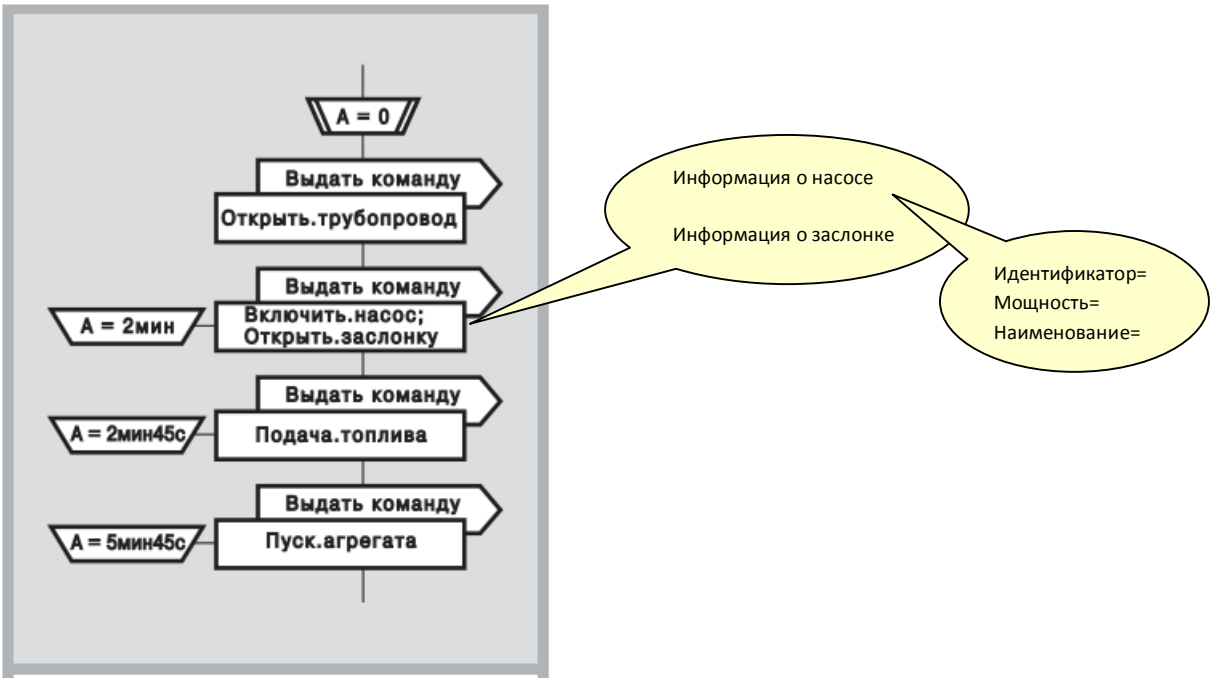


Рис. 3 Взаимодействие с элементом Дракон-схемы

Опять я обращусь к примеру алгоритма управления[4], который показан на рис.3. Вы правой кнопкой мыши выбрали иконку ДЕЙСТВИЕ, внутри которой 2 операции: включить насос и открыть заслонку. Появилось контекстное меню со строками: «информация о насосе» и «информация о заслонке». Вы выбрали насос и получили еще окно, в котором появились строки «Идентификатор=…», «Мощность=…», «Наименование=…».

Как это должно работать? *Графическая оболочка не уведомлена о характере Ваших прикладных задач*. Икона действие лишь содержит записи о насосе и заслонке. Формирует контекстное меню прикладная задача. От графической оболочки получается событие:

```
Контекст-меню: ДРАКОН.ДЕЙСТВИЕ.СОДЕРЖАНИЕ={НАСОС ЗАСЛОНКА}
```

А прикладная задача ищет обработчики для этого. И находит.

```
Обработчик(НАСОС)={«Информация о насосе» «команда-обработчик <имя насоса>»}
```

При нажатии вызывается команда-обработчик с параметром «имя насоса». Эта команда из прикладной задачи или подгружается. Она запрашивает базу данных на сервере через коммуникационный пакет. Ответ отображается: Идентификатор/Мощность/Наименование.

1.3. Элементарные принципы и действия

Сделайте мне красиво!

Мастера каллиграфии современной Японии, где это древнее искусство обрело новое дыхание, используют следующие шесть предметов:

Ситадзики: чёрная, мягкая циновка.

Бунтин: металл для прижатия бумаги во время письма.

Ханси: специальная, тонкая бумага для письма, которую традиционно изготавливают из рисовой соломы. Обычно это Васи - традиционная японская бумага ручного изготовления.

Фудэ: кисть. Существует большая кисть для письма больших символов и маленькая - для написания имени художника.

Судзури: тяжелый, черный сосуд для чернил.

Суми: "тушь для письма" - твёрдый черный материал, в который втирается вода для производства чернил, куда затем макают кисть.

Цветы сурепки вокруг.
На западе гаснет солнце,
Луна на востоке встаёт.

- Ёса Бусон

Концепции всех каллиграфических стилей Японии строились на представлении о том, что написанный текст должен доставлять *эстетическое наслаждение*. Зрительное восприятие текста по значению не уступало зрительному восприятию картины и играло важную роль в смысловом восприятии того же текста.

Как насчет того, чтобы изобразить такое в редакторе?

В [6] показано, как ДРАКОН позволяет изображать процессы, технологии и алгоритмы из различных областей деятельности. Например, алгоритм «изменение кровяного давления» осуществлял проверку: «*Диастолическое* давление превышает норму?». Я не обладаю медицинским образованием, и термин *диастолическое* в меня вселяет ужас. Ибо боюсь в этом слове сделать пару ошибок. Проверка орфографии в MS Word это слово не идентифицирует. Наверное, есть какие-то словари для медиков.

Что вы будете делать, когда настанет время вводить «*диастолическое*» в Дракон-схеме, в ГНОМ-рисунке, в текстовом редакторе? Допустим, вставлена проверка в программу, привязанную к графической системе. Так вот, если в графической системе есть проверка орфографии, то она должна применяться к любому видимому тексту, независимо от того, какую роль он играет в данный момент. Даже, если Вы имеете объект, написанный «тушью», с полем «текст».

Разработка графического пакета должна быть основана на идее, что *любые объекты, которые выглядят одинаково, одинаковые*. Любая иконка ВОПРОС должна обрабатываться как иконка ВОПРОС. Обработчики для текста должны применяться к любому тексту.

Пусть будет шесть предметов, *шесть красивых предметов*, которые не стыдно взять в руки Мастеру! Я знаю одного Мастера, который из обычной технологической картинки делал произведение искусства, достойное японской живописи. Его глаз настолько точен, что он видит с точностью до пикселя. И Он не признает векторную графику, только растровые изображения.

Ряд общих операций должны применяться к содержанию всех графических объектов. Ниже перечислены шесть операций.

- Указание. Пользователь может указать на то или иное содержание;
- Выделение. Пользователь может выделить какое-то содержание;
- Активизация. С помощью «клика» пользователь может активизировать содержание;
- Создание. Появление «пустого»;
- Удаление.
- Модификация или использование (с помощью команд):
    - Перемещение. Вставка содержания в одно место и одновременное его удаление из другого.
    - Трансформация. Преобразование в другой тип данных.
    - Копирование.

1.4. Программный доступ к данным

Я бы взял частями. Но мне нужно сразу.
Остап Бендер

Я в свое время делал графики по АСУТП для Волгодонской АЭС. И имел замечательный пакет DVDraw, разработанный еще до эры объектно-ориентированных систем. Очень быстрый и с минимальными затратами памяти. Только вот графики были в виде «черных ящиков». С не менее «черной» документацией. В графиках имелась возможность изменения цвета при пересечении границы. Например, с зеленого на желтый при превышении предупредительной уставки. И я легкомысленно показал это Заказчику. И это как-то попало в план работ. И только через некоторое время я обнаружил, что это делается «автоматически» по сравнению с заведенным числом. И никак иначе *управлять этим процессом я не могу*. А мне нужно было менять цвет: при недостоверности показания, при превышении и обратно, но с учетом гистерезиса. В итоге я потратил пару недель на то, чтобы отрисовывать 10 линий графиков (каждый со своим цветом) одновременно, 9 из которых в данный момент были невидимыми.

Из программы нужен доступ ко всем видимым данным графической оболочки, если эти данные явно не запрещены.

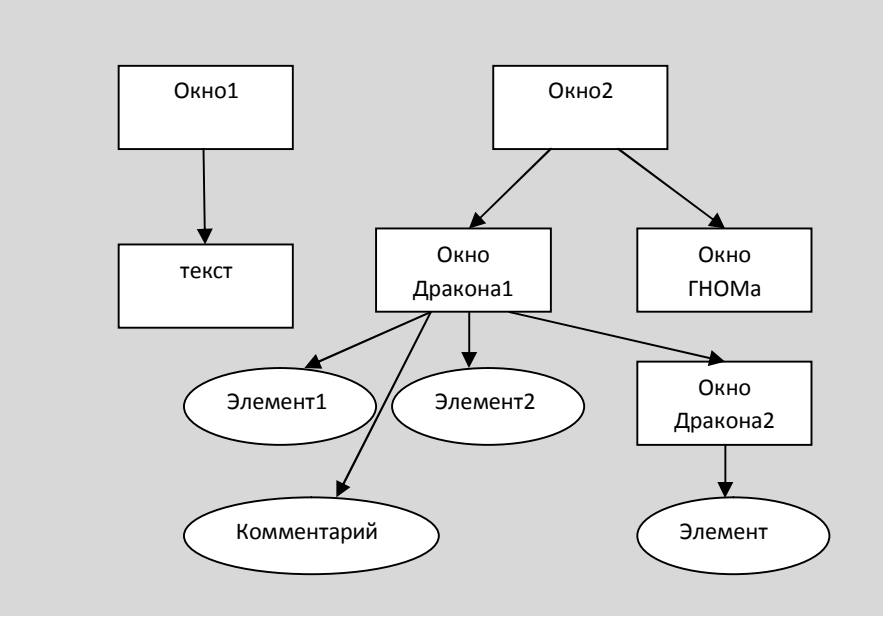


Рис. 4 Иерархия данных графической оболочки

Я предлагаю следующий подход.

1. Всем объектам графической оболочки присваиваются идентификаторы, учитывающие полный путь доступа. Например, нижний справа элемент будет иметь идентификатор «окно2.окно Дракона1.окно Дракона2.Элемент» . Разумеется, это лишь базовый принцип.
2. Все графические элементы принимают 2 типа сообщений. Сообщений в виде структур ElementTypeMsg (для элемента ElementType). Первое – это config – сконфигурировать. Второе – cget – запросить конфигурацию. Конфигурация осуществляется по тем полям ElementType, которые введены в структуру ElementTypeMsg. То есть по всем видимым данным элемента графической оболочки. И по некоторым невидимым. Например, число цифр после запятой при выводе действительного числа.

И наоборот, действие оператора приводит к посылке сообщения прикладной программе.

2. Какой язык нам нужен?

По М. Нечаеву кончик языка отвечает за тазовые органы,
середина слева - за печень, а справа – за селезенку.
Боковым краям соответствует состояние легких – правого и левого.
При хронических алкогольных интоксикациях
язык покрывается коричневым налетом практически у корня.

Нужен второй язык в пару к Оберону. Он должен привнести новые качества, т.е. быть *комплементарным*. Я предлагаю tcl [8] из своих сформировавшихся предпочтений. Он хорошо интегрирован с C и имеет интерпретатор и байтовый компилятор, разработанный почти теми же людьми, что и для Java. Для этой цели легко подойдут Lisp, Perl. Можно еще десяток придумать.

Что за качества второго языка?

Предлагаемый язык обладает качествами:

- Интерпретатор и байт-компилятор
- Все данные - строки
- Функциональный язык
- Хороший пример оконного интерфейса

Интерпретатор и байт-компилятор. Допустим, вы загрузили какой-то видеокадр с сервера. Или Веб-страничку, если хотите. И в ней есть окна ввода, кнопки и прочие элементы интерфейса. Вы будете *загружать с сервера программу на Обероне* вместе с ней? Вот и я не буду. Мне нужен интерпретатор для этого. Как JavaScript, VBScript. Даже для упрощенного C есть интерпретаторы, кстати. Или байтовый компилятор, превращающий в байт-код строки моей программы при загрузке.

Все данные – строки. Я согласен, что данные без типа в одной куче как void* в языке C – это зло. В предлагаемом интерпретаторе проблема типизации решается просто: Все данные – строки.

Присваивания set a 2; set b a; set c 3.14; - одно и то же. Значения будут «2», «a», «3,14». И команды элементов интерфейса на моем видеокадре будут сохраняться на видеокадре же. Как строки. И если мне понадобится поменять цвет на синий, я выполню *строку* config 123 –color #0000ff; где 123 есть номер объекта, которому нужно поменять цвет.

Функциональный язык. Обратите внимание. *Вы интерпретаторе большие куски писать не будете. Оберон для этого. А маленькие кусочки будете.* Вроде «config 123 –color #0000ff», который *всегда* возвратит Вам строку. И еще Вы будете знать код ошибки.

Хороший пример оконного интерфейса. Есть tk - старая добрая графическая оболочка для tcl. Я не призываю повторять. Я призываю взять подходы. Минимум из них.

Основные понятия языка приведены ниже. Все – это строки: программы, данные, переменные.

Язык tcl.

`$var` – означает строку, равную значению переменной `var. set var  xx; puts $var;` - устанавливает и печатает переменную `var`, равную "`xx`". Значение переменной может устанавливаться императивным языком.

`[func arg1 arg2]` – означает выполнение встроенной команды или заданной программистом процедуры `func`. Которая принимает 2 аргумента. Результат – тоже строка. `puts [string toupper xx]` – печатает результат функции string. Первый аргумент `toupper` – тип преобразования – в верхний регистр. Результат функции – XX.

`expr ($n+2)*3` – означает выполнение выражения, в частности, `(n+2)*3`. Заметьте, все имена и значения строковые, поэтому требуется команда expr и взятие значения переменной.

`proc func {arg1 arg2} { …}` – означает задание процедуры, принимающей аргументы `arg1`, `arg2`. Пример рекурсивной процедуры вычисления числа Фибоначчи:

```
proc fib {n} {
      if {$n <= 2} {
            return 1
      }
      expr [fib [expr $n-1]]+[fib [expr $n-2]]
}
```

Фигурные скобки, как и двойные кавычки, объединяют текст в строку с пробелами. Отличаются кавычки тем, что в таких строках выполняются функции и подстановки значения переменных.

«`puts $var`» - есть строка `puts xx`;

`{puts $var}`- есть строка `puts $var`;

Графический пакет tk.

Графический пакет tk очень простой. Он состоит из операторов трех типов:

- Создание окон и объектов;
- Запрос и модификация конфигурации;
- Операции над окнами и объектами (удаление, расположение …)

Окна имеют иерархическую структуру. Например, окно .x.y.z имеет родительское окно .x.y, а окно .f не имеет родительского. Окно типа canvas представляет собой двумерное поле для разного рода графических и текстовых элементов. Любых. В том числе и для окон, любое из которых может быть в свою очередь canvas.

Следует отметить, что окна могут разрабатываться разными людьми. И подключаться к работающей системе как сменные модули – динамические библиотеки.

На рис. 5 показан пример простой программы на tcl/tk, вводимый окно командного интерпретатора.

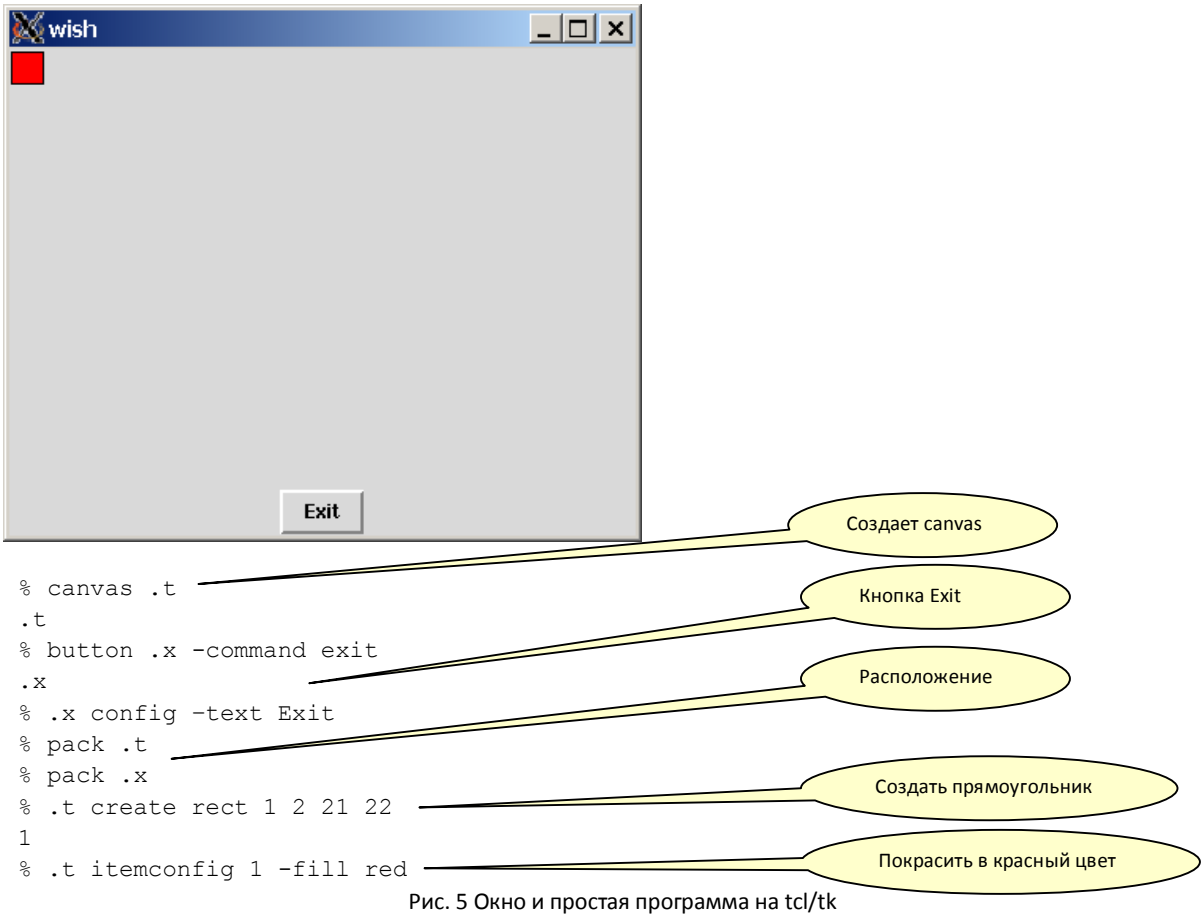


Рис. 5 Окно и простая программа на tcl/tk

К каждому окну можно применить config, cget. К каждому элементу окна canvas можно применить itemconfig, itemcget.

Выполнение скрипта из императивного языка осуществляется (реализован на C):

`Tcl_Eval(interp, " puts [string toupper xx]");` - где interp есть структура интерпретатора, возвращающая код завершения как возвращаемое значение и результат interp->result как строку.

К любой переменной var может быть привязан callback. Это программа, вызываемая, чтобы установить значение переменной перед чтением в интерпретаторе. Или чтоб использовать значение переменной после записи в интерпретатор.

Tcl_TraceVar(interp, “var”, on_var_read); - устанавливает программу on_var_read на чтение.

Выполнение оператора императивного языка с параметрами возможно из интерпретатора.

my_prog par1; - так вызывается программа из интерпретатора.

Tcl_CreateCommand(interp, “my_prog”, my_prog_impl); - так программа my_prog_impl привязывается к имени my_prog.

И, наконец, командой может создаваться объект и заполняться его поля:

Tcl_CreateCommand(interp, “my_type”, my_type_impl); - может создавать объект

my_type object; - создать объект

object name My-Name; - выполнить метод, заполнить имя

3. Все вместе

3.1. Концепт Вашей системы

Всякая работа требует больше времени, чем вы думаете.
Следствие Закона Мэрфи

Вот Вы соединили все и решили сделать *систему* на этой графической оболочке. Систему, обучающую начинающих программированию. Вы берете 2 примера из [4]. Ваш редактор достаточно хорош, чтобы отображаемые им картинки *выглядят не хуже, чем в книге*(Рис. 6).

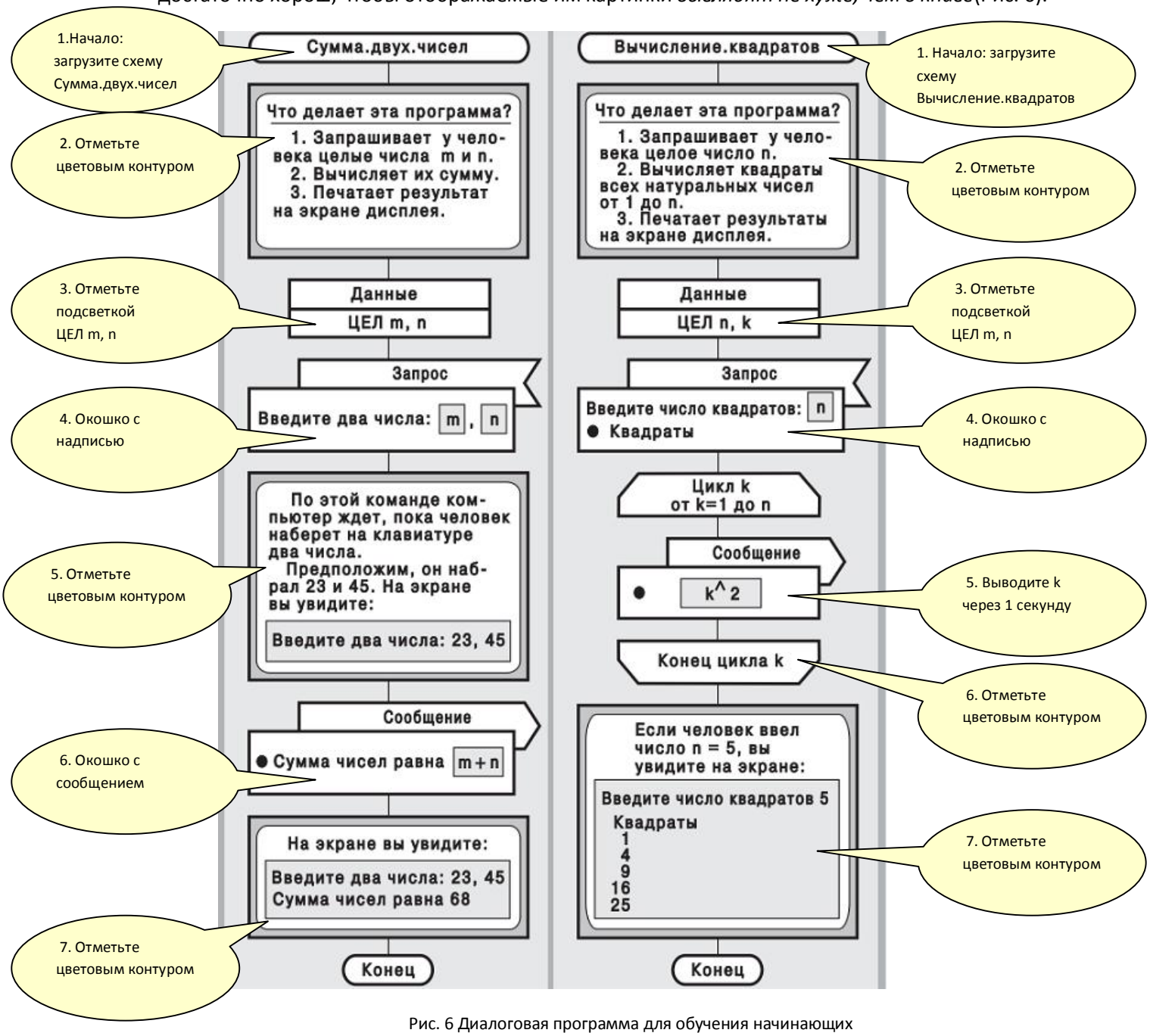


Рис. 6 Диалоговая программа для обучения начинающих

И поэтому Вы *набиваете* по новой в редакторе эти схемы. Ибо .png рисунок в Дракон-схему Вы сконвертировать не сможете. Берете программу-редактор и … задумываетесь.

Перед Вами доверчивые 9-классники, которых Вы обучаете программированию. По этим схемам Вам делать презентацию. И мы поглядим, как это все должно работать.

1. Схема слева загружена, и Вы приступаете к рассказу о том, как работает показанная на ней программа.

2.Графический пакет отметил цветовым контуром икону «Комментарий»: Что делает эта программа? Вы медленно рассказываете. А программа ждет вашего клика. Вы кликаете.

3. Графический пакет подсветил декларацию переменных ЦЕЛ m, n в иконе «Полка». И ждет Вашего клика.

4. Графический пакет отметил цветом иконку и вывел окошко с надписью «Введите два числа:». Вы пробуете ввести текстовые символы – и не получается! Вводите целые числа, и окошко исчезает.

5. Снова комментарий. Вы кликаете.

6. Вам вывели окошко с сообщением: Сумма чисел равна … Вы показываете, просите учеников проверить. И кликаете дальше.

7. Последний комментарий.

Для правой схемы имеем аналогичное.

1. Схема справа загружена, и Вы приступаете к рассказу о том, как работает показанная на ней программа.

2.Графический пакет отметил цветовым контуром икону «Комментарий»: Что делает эта программа? Вы медленно рассказываете. А программа ждет вашего клика. Вы кликаете.

3. Графический пакет подсветил декларацию переменных ЦЕЛ n, k в иконе «Полка». И ждет Вашего клика.

4. Графический пакет отметил цветом иконку и вывел окошко с надписью «Введите число квадратов:». Вы пробуете ввести текстовый символ – и не получается! Вводите целое число=5, и окошко исчезает.

5. Вы видите, как в цикле с интервалом в 1 секунду выводится число квадратов: 1, 4, 9, 16, 25.

6. Цикл окончен. Вы кликаете дальше.

7. Последний комментарий.

Допустим, Вы имеете среду Дракон и программы конвертации во все Ваши императивные языки. Вы преобразовали этот алгоритм в программу на Обероне. Но что-то *не получается с программой*. Ибо Вы не можете остановить эту программу. На комментарии Вы не можете остановиться по определению. На операторах? Если был отладчик MS Visual Studio, вы бы, конечно, смогли хотя бы остановиться. Вы в шоке.

Вы, конечно, знаете, что по академику А.Ершову «Человек неизмеримо усилит свой интеллект, если … сделается программистом» [4]. И Вы приступаете к *практической реализации* в стиле фантазии на тему Пароннджанова «Как улучшить работу ума».

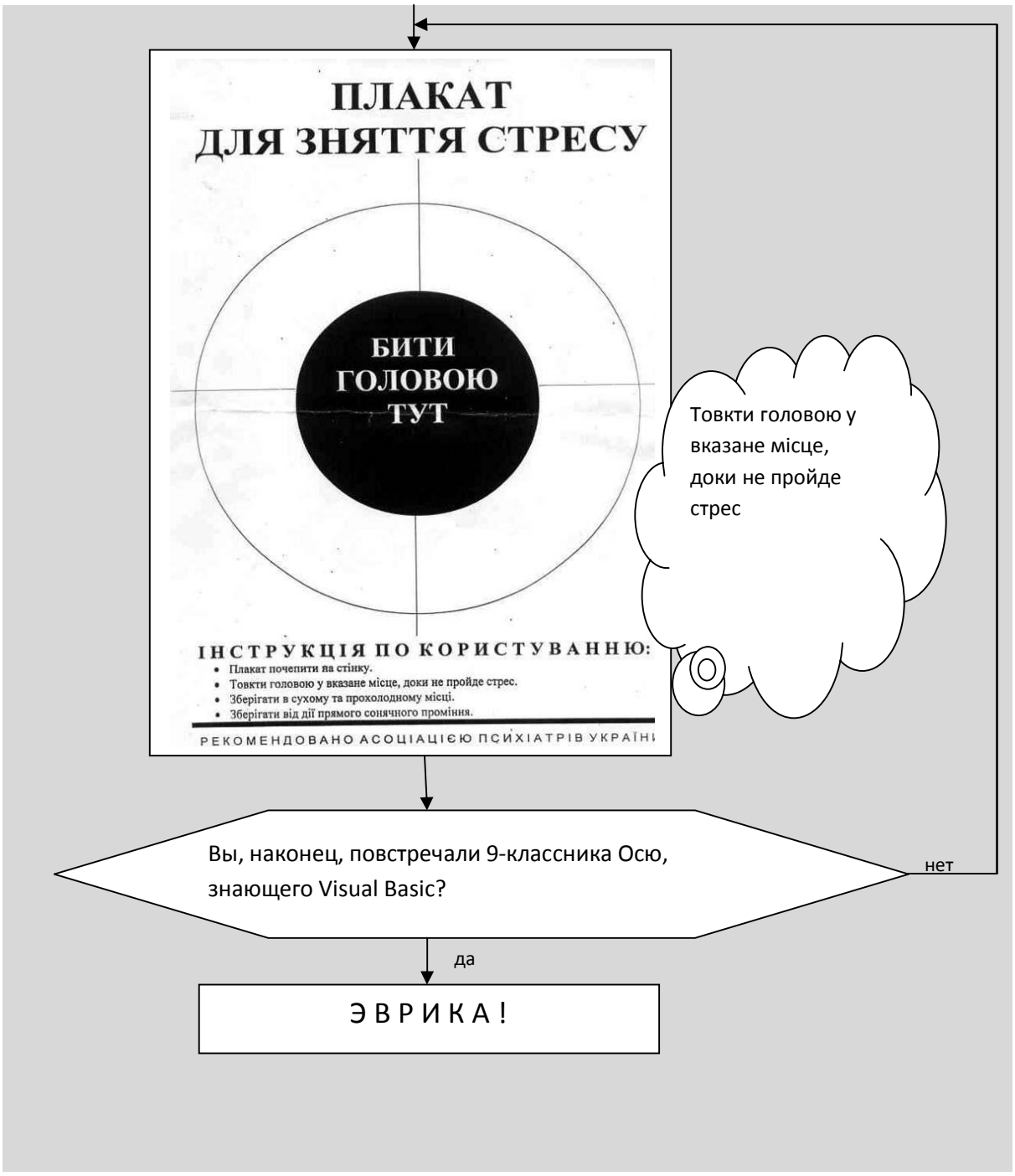


Рис. 7 Алгоритм улучшения работы ума программиста

И будете снимать стресс до тех пор, пока Вам не встретится умный 9-классник Ося, который уже изучил Visual Basic и знает как с помощью его и MS PowerPoint сделать *презентацию.* И Вы поймете две вещи:

- Нужен интерпретатор;
- Ваша Дракон-схема это одно, а алгоритм программы – совсем другое.

Далее Вы бросаетесь к редактору и создаете совсем другую схему: алгоритм работы программы Вашей презентации (На Рис.8 приведен фрагмент алгоритма для одной иконки).

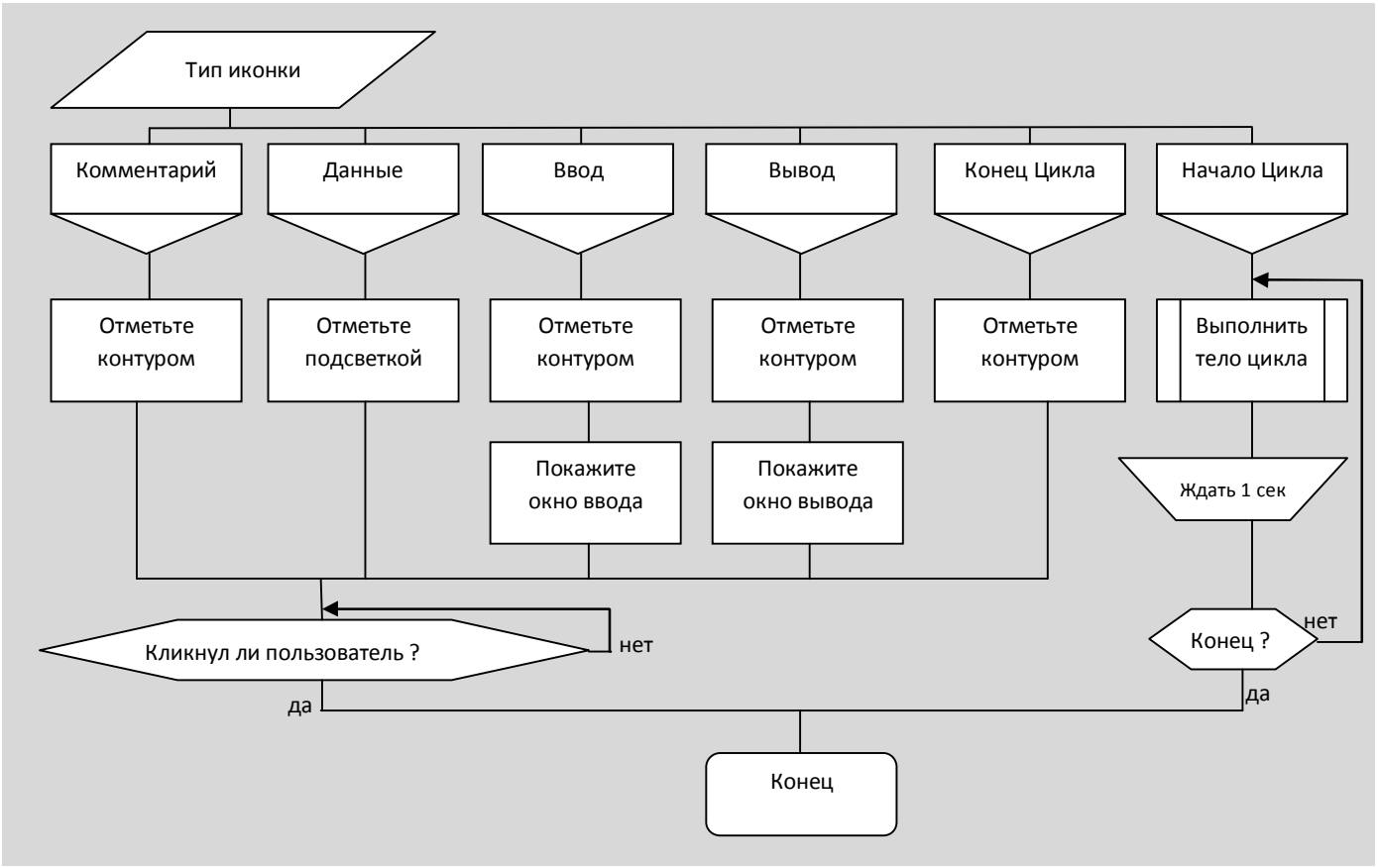


Рис. 8 Алгоритм презентации для данной иконы

Все, теперь алгоритм презентации есть. Вы на него посмотрели, полюбовались и поняли: *никакого когнитивного смысла в нем нет*. Ибо показывать его некому. Не нужны стихи, которых никто не читает, картины, которыми никто не любуется. В результате очевидно следующее: ДРАКОН-схема, предоставляемая пользователю это совсем не то, что алгоритм программы, которая работает на компьютере. Пусть и в контексте данной ДРАКОН-схемы.

3.2 Структура данных схемы для Вашей системы.

«Дао рождает одно, одно рождает два,
два рождают три, а три рождают все существа»
Дао Дэ Цзин

Наконец, Вы приступаете к непосредственной работе программиста. Вы начинаете определять структуры данных. И понимаете, что и структуры редактора и формат данных не подходят. Вам нужен формат данных, отражающий Дракон-схему. Ибо Вы это делали для императивных алгоритмов, а данный алгоритм не императивный (Автор тоже давал предложения в [3] для императивных алгоритмов). Вот что получается (Рис. 9).

```
ТИП_СХЕМЫ Презентация
ЗАГОЛОВОК S1 –name Сумма.двух.чисел –coords 50+20+40X10
S1 goto S2
КОММЕНТАРИЙ S2 –coords 50+45+40X30
S2 hypertext {
<u>Что делает эта программа?</u>
   1. Запрашивает у человека целые числа m и n.
   2. Вычисляет их сумму.
   3. Печатает результат на экране дисплея.
}
S2 goto S3
ПОЛКА S3 –text {ЦЕЛ m, n} –coords 50+70+30X20
S3 datatype ЦЕЛ {m n}
S3 goto S4
ВВОД S4 –hypertext {Введите два числа <border>m, n</border>} –coords 50+100+40X30
S4 input {Введите два числа <input type=”ЦЕЛ”>m</input>, <input type=”ЦЕЛ”>n</input>}
S4 goto S5
КОММЕНТАРИЙ S5 –coords 50+140+40+50
S5 hypertext {
   По этой команде ком-
пьютер ждет, пока человек
наберет на клавиатуре
два числа.
   Предположим, он наб-
рал 23 и 45. На экране
Вы увидите:
<border>Введите два числа: 23, 45</border>
}
S5 goto S6
ВЫВОД S6 –hypertext {Сумма чисел равна <border>m+n</border>} –coords 50+200+40X30
S6 output {Сумма чисел равна <output type=”ЦЕЛ”>[expr $m+$n]</output>}
S6 goto S7
КОММЕНТАРИЙ S7 –coords 50+240+40X50
S7 hypertext {
   На экране вы увидите:
<border>Введите два числа: 23, 45
Сумма числе равна 68</border>
}
S7 goto S8
КОНЕЦ S8 –coords 40+300+30X20
СВЯЗЬ S1 S2 50+25 50+30
…
```

Тип схемы определяет состав всех остальных полей

Координаты X+Y+длинаХширина

<u> - подчеркивание
<border> - рамка

Ввод целого числа m в окне

[Арифметическое выражение m+n] в виде строки

Вывод результата m+n в окне

Рис. 9 Формат данных Дракон-схемы

Формат данных представляет собой набор структур и связей между ними. Запись, конечно же, напоминает объектно-ориентированную. Но я не предлагаю ни полиморфизма, ни инкапсуляции. Ограничимся структурами. Этот формат есть текст, текст на интерпретаторе. Все данные – строки.

Каждая структура состоит из *полей*. Состав полей – различный для различных прикладных задач. Поэтому первый оператор определяет ТИП_СХЕМЫ=Презентация.

Оператор	Описание
ТИП_СХЕМЫ=Презентация	Некий файл ресурсов «Презентация» описывает состав полей структур, используемых ниже
ЗАГОЛОВОК <имя>	Экземпляр структуры <имя> для иконы ЗАГОЛОВОК
КОММЕНТАРИЙ <имя>	Экземпляр структуры <имя> для иконы КОММЕНТАРИЙ
ПОЛКА <имя>	Экземпляр структуры <имя> для иконы ПОЛКА
ВВОД <имя>	Экземпляр структуры <имя> для иконы ВВОД
ВЫВОД <имя>	Экземпляр структуры <имя> для иконы ВЫВОД
КОНЕЦ <имя>	Экземпляр структуры <имя> для иконы КОНЕЦ
СВЯЗЬ <имя1> <имя2> <тчк1> <тчк2>	Описание линии. Чтобы этого файла было достаточно, чтобы отрисовать ДРАКОН-схему.
Остальные	Просто заполняют значения полей

Вы спросите, зачем редактору нужен текст, интерпретатор, когда можно *просто загрузить бинарный файл* с Дракон-схемой. И будете правы. Однако заметьте, что я не предлагаю этот формат данных *вместо* бинарного. Я его предлагаю *вместе* с бинарным. При работе грузите бинарный. Но если вы хотите откуда-то файл сконвертировать, то Вам нужно иметь возможность *загружать и сохранять текстовый*. Да и значения полей для Вашего бинарного файла будут текстовыми.

Остальные операторы просто заполняют строками значения полей структур. При этом возможны две формы записи: икона имя -опция значение –опция значение … и имя опция значение .

```
ВЫВОД S6 –hypertext {Сумма чисел равна <border>m+n</border>}

S7 hypertext {…
```

И то, и другое заполняет поле «hypertext» структур.

Среди строк полей есть пара интересных. Поле input в строке содержит *тэги ввода данных*. В поле input вводятся два числа. Это означает, что будет сформировано окно, состоящее из текста и *текстовых символов для двух окон ввода*: Введите два числа m, n. Графической оболочке будет сказано:

- создай окно с гипертекстом;
- добавь текст «Введите два числа»;
- добавь окно ввода для m, тип данных – ЦЕЛ;
- добавь текст «, »;
- добавь окно ввода для m, тип данных – ЦЕЛ;
- добавь текст «.».

Поле output *выводит* строку с результатом в окне вывода аналогично. Но прежде выполняется арифметическое вычисление [expr $m+$n], вычисляющее сумму из *строк* m и n.

3.3. Исполняющая программа для Вашей системы.

«Программа стать счастливым, к которой нас принуждает принцип удовольствия, неисполнима, и все же мы не должны – нет, мы не можем – отказаться от стараний хоть как-нибудь ее исполнить»
Зигмунд Фрейд

Ваша программа должна исполнять алгоритм рисунка 8 для каждого типа иконки. Программа на Обероне будет иметь приблизительно такой вид (Рис. 10).

```
CONST IconComment = 21; IconData = 10; IconInput = 14; IconOutput = 15;
      IconEndCycle = 13; IconCycle = 12;
CONST MarkCountour = 22; MarkHighlight = 23;
PROCEDURE ProcessIcon(sch: Schema; ic: IconData)
      VAR ici: InputIconData; ico: OutputIconData;
BEGIN
      CASE ic.iconType OF
            IconComment: ItemConfig(sch, ic.iconId, MarkCountour, 1)
          | IconData: ItemConfig(sch, ic.iconId, MarkHighlight, 1)
          | IconInput: BEGIN
                        ItemConfig(sch, ic.iconId, MarkCountour, 1);
                        ici := ic(InputIconData);
                        ItemInput(sch, ici.inputCmd);
                  END
          | IconOutput: BEGIN
                        ItemConfig(sch, ic.iconId, MarkCountour, 1);
                        ico := ic(OutputIconData);
                        ItemOutput(sch, ico.outputCmd);
                  END
          | IconEndCycle: ItemConfig(sch, ic.iconId, MarkCountour, 1)
          | IconCycle: BEGIN
                        icc := ic(CycleIconData);
                        REPEAT
                              ProcessIcon(sch, ic.gotoIconData);
                              Delay(Timeout1Second)
                        UNTIL ItemExpression(icc.expr) # 0
                  END
      END
END
PROCEDURE ItemConfig(sch: Schema; iconId, option, value: INTEGER);
PROCEDURE ItemInput(sch: Schema; inputCmd: ARRAY 256 OF CHAR);
PROCEDURE ItemOutput(sch: Schema; inputCmd: ARRAY 256 OF CHAR)
      VAR rc: INTEGER;
BEGIN
      rc := Tcl_Eval(sch.interp, inputCmd);
      IF rc = 0 THEN BEGIN
            globalInputWindow := GraphicPkg.CreateWindow(sch);
            GraphicPkg.Write(sch, sch.interp.result);
      END
END
PROCEDURE ItemExpression(sch: Schema; inputCmd: ARRAY 256 OF CHAR);
```

Типы иконок и типы полей

Покажите окно ввода

Покажите окно вывода

Выполнить тело цикла и ждать 1 сек

Выполнить input_data получить гипертекст

Открыть окно и показать строку вывода

Рис. 10 Прикладная задача обработки

Для каждого элемента выполняется эта процедура. По клику элемент возвращается к «серому цвету» и далее переходим к следующему в списке: ic.gotoIconData.

Связь с объектом осуществляется через процедуры ItemConfig, ItemInput, ItemOutput, ItemExpression. В схеме должен быть реализован индексный доступ через iconId.

- Процедура ItemConfig изменяет значение поля для данного элемента, например, цвет контура.
- Процедуры ItemInput|ItemOutput выводят новые окна ввода и вывода. Во время их выполнения вызывается подстановка строки. Она преобразует выражение [expr $m+$n] в строку в виде суммы.
- Процедура ItemExpression вычисляет выражение аналогично, например, конец цикла в примере рис. 6 справа.

Обратите внимание, что значения переменных m и n должны быть установлены, прежде чем мы придем к иконе вывода. Это осуществляется графической оболочной. Гипертекстовая строка <input type=”ЦЕЛ”>m</input> не только рисует окна ввода, но и устанавливает целые значения m и n в интерпретаторе.

В результате Вы имеете систему с двумя языками. Единственное, о чем я хочу предостеречь: *не используйте интерпретатор там, где с той же легкостью может использоваться Оберон*. Делайте программы эффективными.

И еще. Программа написана и схему на Рис. 8 можете отправить в мусорную корзину за ненадобностью. Ибо схема эта когнитивной нагрузки не несет. А вот схемы на Рис. 6 – для Пользователя, они нужны. Я высказался, можете соглашаться, можете спорить.

3.4. Запуск программы для Вашей системы.

"It's a Unix system, I know this!"
Девочка в фильме Парк Юрского Периода,
найдя нужную иконку в графической системе.

Как Ваша прикладная программа должна запускаться? Ибо она пока лежит в файле на диске. Правильно, из графической оболочки. Самое простое – выбрать правой кнопкой мыши Дракон-схему и создать *контекстное меню*. Как создать контекстное меню? Спросим что-нибудь попроще. Что мы знаем о Дракон-схеме?

- Эта схема – Дракон-схема;
- Тип схемы – Презентация.

Вот из этих данных и надо создать контекстное меню. Пусть у Вас заведена информация о прикладных программах: Презентация.Редактировать <другая программа> и Презентация.Выполнить <Ваша программа>. Вы выбираете схему, создаете контекстное меню из двух выборов: Редактировать и Выполнить. И нажимаете Выполнить. Тогда Ваша прикладная задача загрузится.

Отлично, а если Вы до этого редактировали туже самую схему. И была запущена другая программа – Редактор? Возможно, предыдущая работавшая с данной схемой задача выгрузится. Здесь должен работать *алгоритм сборки мусора*, основанные на счетчике ссылок.

- Если программа вызывается графическим интерфейсом, то ей присваивается счетчик ссылок 1.
- Если программа повторно используется графическим интерфейсом, счетчик ссылок увеличивается на 1.
- Если графический интерфейс вызывает другую программу, счетчик ссылок уменьшается на 1.
- И, наконец, если счетчик ссылок равен 0, программа завершается.

4. С высоты птичьего полета

«Замедляя ход, Маргариту догнала Наташа. Она, совершенно нагая с летящими по воздуху растрепанными волосами, летела верхом на толстом борове»
Булгаков

Опять я возвращаюсь к идеям Дж.Раскина в плане подходов к построению интерфейсов. «Человек всегда очень плохо ориентировался в лабиринтах. При работе с какой-то сложной программой часто случается, что для решения возникшей проблемы необходимую команду или флажок можно найти где-нибудь глубоко в подменю. Мы плохо запоминаем длинные последовательности поворотов, - именно поэтому лабиринты являются хорошими головоломками.

Противоположностью лабиринтов является ситуация, в которой Вы можете видеть цель и путь к ней, который позволяет сохранить чувство ориентации во время перемещения и при необходимости вернуться назад. Изящным решением является принцип масштабируемого интерфейса».

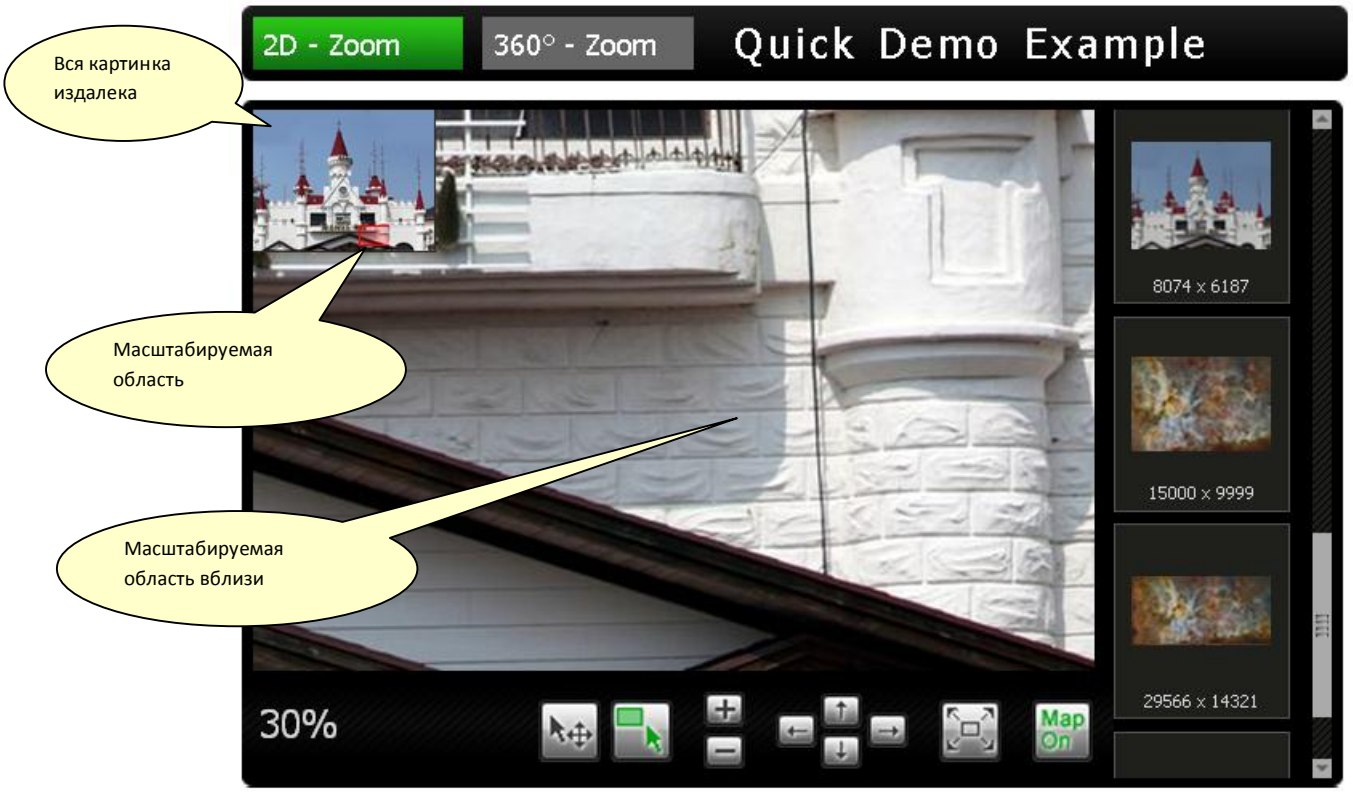


Рис. 11 Масштабируемый интерфейс AJAX-ZOOM

В масштабируемой области по мере приближения появляются новые детали, которые были невидимы раньше. Как будто стены в комнате начинают покрываться заметками, листочками, новыми деталями изображения. И вы уже ориентируетесь и знаете, в каком углу находится интересующая Вас информация.

Применительно к нашему случаю мы подчас имеем схемы, вложенные одна в другую (Рис. 12).

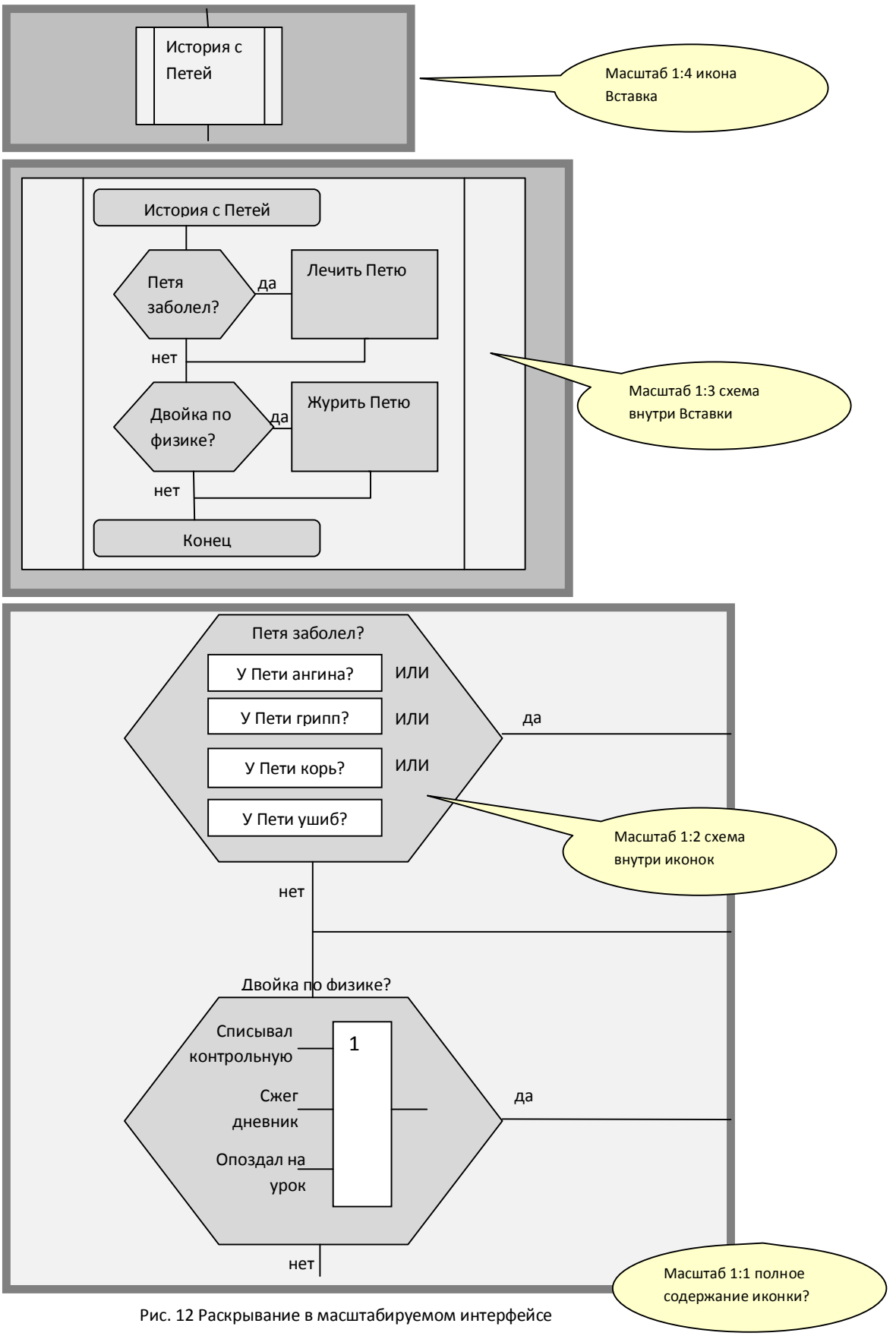


Рис. 12 Раскрывание в масштабируемом интерфейсе

Опишем, как выйти из «истории с Петей», рассмотренной в [4, рис.60]. При масштабе 1:4 Вы видите иконку Вставка, внутри которой расположен алгоритм «история с Петей». Увеличиваете масштаб (например, колесиком на мышке) и получаете 1:3. Вы видите контуры иконы Вставка по бокам и алгоритм Дракон-схемы внутри. Приближаете еще до 1:2, и появились деталь внутри иконок Вопрос. Там логические выражения, и эти выражения реализованы на комплементарных языках[3]. Сверху – на текстовом языке. Снизу – на FBD. Вы можете приблизиться еще, и тогда должны увидеть *содержание всех полей* на фоне иконки во весь экран.

Данный подход к интерфейсу учитывает любые возможные степени вложенности. Ибо система может быть сколь угодно разветвленной.

Я не согласен с подходом, предложенным В.Д.Пароджановым[4], согласно которому «инструментальные программы языка Дракон должны обеспечить автоматический перевод рамочного алгоритма и наоборот» (Рис. 13). Предложенный мной подход лучше. Ибо *Вы не меняете схему, а лишь масштаб детализации*.

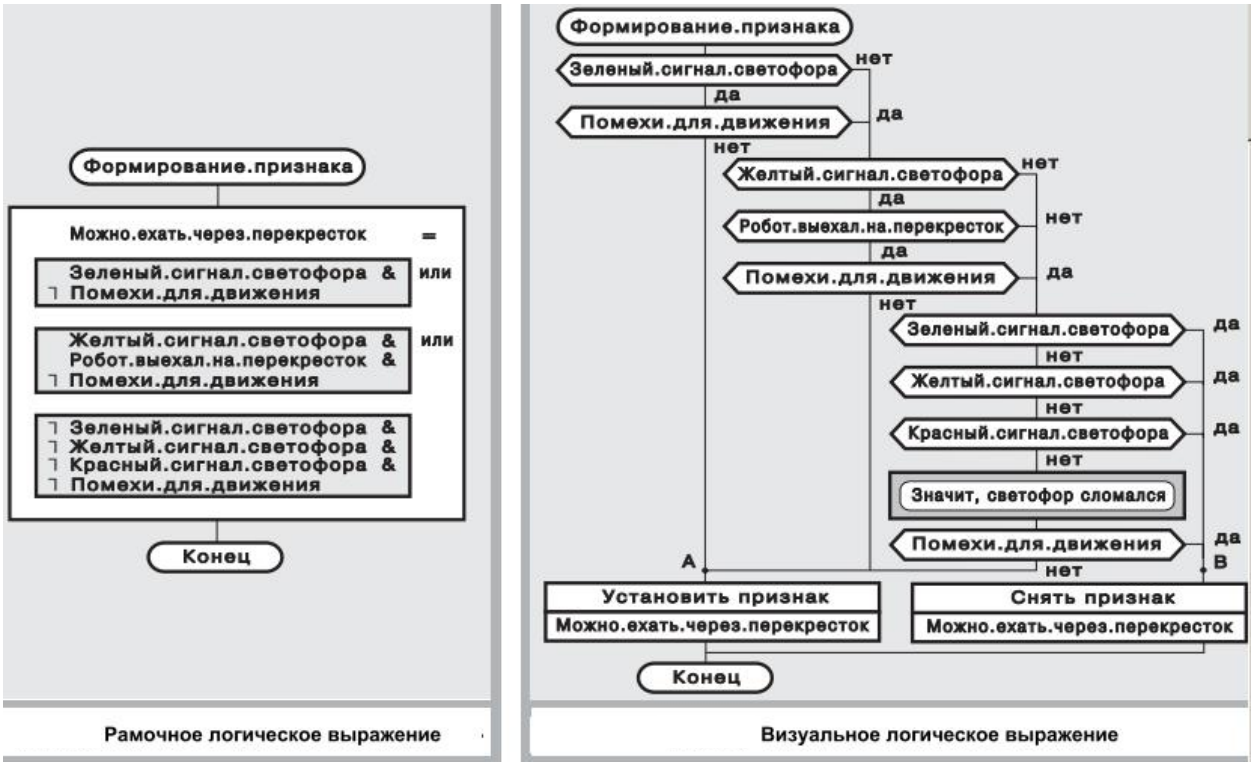


Рис. 13 Визуальное и рамочное представление

К сообществу

«А двести рублей не спасут гиганта мысли?
Я думаю, торг здесь неуместен!»
Из разговора Кислярского с Воробьяниновым

Дорогие Драконоводы! Пишу все это не для того, чтобы я сделал свой графический пакет *вместо* ваших. А затем, чтобы Вы прониклись и *вместе* сделали графический пакет со сменными модулями. *Вместе* друг с другом, *вместе* с моими идейками. Я предлагаю некую модель координации усилий. Учитываю, что у каждого есть в первую очередь собственные корпоративные интересы. Нужно иметь доступный базовый графический пакет и набор типов окон: текст, гипертекст, Дракон, ГРАФ, FBD, МОЛНИЯ, …

Каждому доступно: Базовый пакет + Число типов окон.

Здесь важна даже не координация усилий, а координация интерфейсов. А также согласие разработчиков публиковать базовый графический пакет и сменные окна.

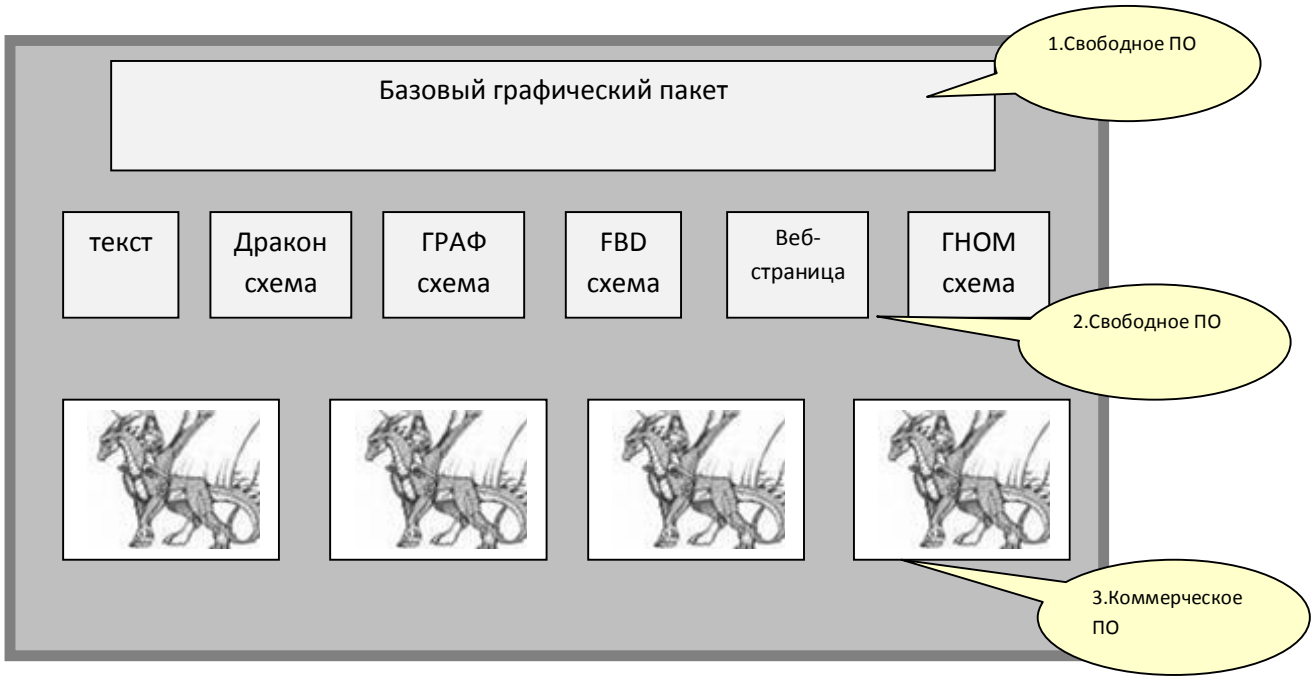


Рис. 14 Состав программных систем

На рис. 14 предложена такая схема. Графический пакет и окна к нему разрабатываются как свободное ПО под лицензией LGPL. Разработчики выигрывают лишь от взаимного обмена. Лицензия LGPL позволяет использовать уровни 1 и 2 разработчикам коммерческого ПО. Тут уже каждый драконовод разрабатывает *для своей берлоги* и ничего не публикует.

Заключение

Может быть, Вы найдете лучший функциональный язык в пару к Оберону. Может, Вы предложите лучшую графическую оболочку. Пусть даже несколько таких. Или, может быть, захотите проголосовать. Возможно.

Остановитесь. Подумайте.

И сделайте Ваши ставки, Господа!

Список литературы

1. Кухонная свара в Сокольниках. http://www.gzt.ru/topnews/world/40074.html?from=copiedlink
2. Дж.Раскин. Интерфейс. Новые направления в проектировании компьютерных систем.- Пер. с англ. – СПб, 2005.
3. Д.В.Дагаев. Алаверды Дракону. http://forum.oberoncore.ru/download/file.php?id=2298
4. В.Д.Паронджанов. Как улучшить работу ума? М.: 2001
5. В.Д.Паронджанов. Язык Дракон. Краткое описание.
6. В.Д.Паронджанов. Дружелюбные алгоритмы, понятные каждому. М: 2010
7. В.Д.Паронджанов. Почему мудрец похож на обезьяну. М: 2007
8. Tcl. http://www.activestate.com/ActiveTcl
9. Масштабируемый интерфейс AJAX-ZOOM. http://www.ajax-zoom.com