

УДК
519
Е 70

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

А.П. ЕРЕМЕЕВ, П.В. ГРЕЧКИНА, Н.В. ЧИБИЗОВА

**КОНСТРУИРОВАНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ
ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ
НА ОСНОВЕ ИНСТРУМЕНТАЛЬНОГО КОМПЛЕКСА G2**

Учебное пособие
по курсам
«Экспертные системы» и «Интеллектуальные системы»
для студентов, обучающихся по направлениям
«Прикладная математика и информатика»,
«Информатика и вычислительная техника»,
«Информационные системы и технологии»

УДК
519
Е-70

*Утверждено учебным управлением МЭИ
в качестве учебного пособия для студентов*

Подготовлено на кафедре прикладной математики

Рецензенты: лауреат премии Президента РФ в области образования, д.т.н., действительный член РАЕН, профессор В.Н. Вагин, лауреат премии Президента РФ в области образования, д.т.н., действительный член РАЕН, зам. директора ГУ РосНИИИТиАП, профессор И.Б. Фоминых

Еремеев А.П., Гречкина П.В., Чибизова Н.В.

Е-70 Конструирование интеллектуальных систем поддержки принятия решений реального времени на основе инструментального комплекса G2: учебное пособие. – М.: Издательский дом МЭИ, 2012. – 89 с.
ISBN

Рассматриваются основные концепции конструирования интеллектуальных (экспертных) систем поддержки принятия решений реального времени (ИСППР РВ), предназначенных для мониторинга и управления сложными техническими и организационными системами. Приводится описание эффективного инструментального комплекса проектирования экспертных систем реального времени G2 (GENSYM Corp.), предназначенного для разработки интеллектуальных (экспертных) систем реального времени на основе современных технологий применения вычислительной техники и программирования. Возможности использования системы при выполнении различных функций иллюстрируются примерами.

Пособие предназначено для студентов, обучающихся по специальностям (профилям) «Прикладная математика и информатика» (010501), «Информационные системы и технологии» (230201) направлений подготовки «Прикладная математика и информатика» (010500), «Информационные системы и технологии» (020300), «Информатика и вычислительная техника» (230100) и изучающих дисциплины «Экспертные системы», «Интеллектуальные системы», а также выполняющих курсовые, научно-исследовательские и выпускные работы по тематике конструирования интеллектуальных (экспертных) систем реального времени. Пособие будет также полезно аспирантам, научным сотрудникам и специалистам, занимающимся вопросами создания ИСППР для мониторинга и управления сложными техническими и организационными системами в реальном времени.

ISBN 978-5-383-0029-8

© Московский энергетический институт
(технический университет), 2010

ВВЕДЕНИЕ

Интеллектуальные системы поддержки принятия решений реального времени (ИСППР РВ) – это программно-аппаратные комплексы, предназначенные для помощи лицам, принимающим решения (ЛПР) при мониторинге и управлении сложными объектами и процессами различной природы в условиях, как правило, достаточно жестких временных ограничений. При поиске решения используются экспертные модели, построенные на основе знаний специалистов-экспертов, и эвристические методы поиска решений. По современной классификации ИСППР РВ можно отнести к классу интегрированных динамических интеллектуальных (экспертных) систем семиотического типа, сочетающих строгие математические методы поиска решения, базирующиеся на формальных методах оптимизации, с нестрогими логико-лингвистическими моделями и эвристическими методами, базирующимися на экспертных знаниях [1,2].

Необходимость создания ИСППР РВ обуславливается непрерывно возрастающей сложностью управляемых объектов и процессов с одновременным сокращением времени, отводимого ЛПР на анализ проблемной ситуации и принятие необходимых управляющих воздействий. Так, например, оператору атомного энергоблока ВВЭР-1000 на принятие решения по коррекции аномального состояния объекта и переводу его в штатный режим часто отводится менее минуты.

Концептуально объединяя подходы и методы теории принятия решений, теории оптимизации, искусственного интеллекта (инженерии знаний), теории информационных систем, и используя объективную и субъективную информацию, ИСППР РВ обеспечивает ЛПР анализом решаемой проблемы, возможными прогнозами последствий принятия тех или иных управляющих воздействий и направляет его в процессе поиска решения с целью повышения эффективности принимаемых решений.

Одна из основных задач при конструировании ИСППР РВ – выбор подходящего формального аппарата для описания процесса принятия решений и построение на его базе адекватной (корректной) модели принятия решений (МПР). В качестве такого аппарата часто используются продукционные системы в сочетании с объектно-ориентированным подходом к представлению информации. Однако, большинство имеющихся на сегодня программных инструментальных средства проектирования экспертных систем ориентируются в основном на статические проблемные и предметные области, не требующие коррекции модели и стратегии принятия решений в процессе поиска решения, что не подходит для конструирования на их основе ИСППР РВ.

Необходимы инструментальные средства, позволяющие проектировать ИСППР РВ, способные функционировать в динамических и открытых проблемных/предметных областях при возможной

корректировке и пополнении МПР (базы знаний и стратегий поиска) в процессе поиска решения. К таким проблемным областям относится оперативно-диспетчерское управление сложными техническими и организационными объектами и процессами, в частности, объектами энергетики типа энергоблоков [3,4].

Особенностью задач, решаемых с помощью ИСППР РВ, является:

- необходимость учета временного фактора при описании проблемной ситуации и в процессе поиска решения;
- необходимость получения решения в условиях временных ограничений, определяемых реальным управляемым процессом;
- невозможность получения всей объективной информации, необходимой для решения, и, в связи с этим, использование субъективной, экспертной информации;
- комбинаторность поиска, необходимость активного участия в нем ЛПР;
- наличие недетерминизма, необходимость коррекции и введения дополнительной информации в процессе поиска решения.

Реализовать ИСППР РВ в полном объеме возможно с переходом на мощные вычислительные платформы типа рабочих станций и кластеров и соответствующие инструментальные среды, к наиболее известным из которых относятся G2 (Gensym Corp., США), RTworks (Talarian Corp., США), включая их различные проблемно/предметно ориентированные расширения [5]. Однако следует отметить, что непосредственное управление сложными и критическими процессами, особенно при использовании новых технологий, слишком рискованно. Переход к прямому управлению возможен только в случае хорошо изученных и освоенных технологий. Основное же назначение ИСППР РВ – помощь ЛПР при мониторинге, контроле, диагностике и управлении сложными объектами и процессами, выявление и предупреждение опасностей, разработка рекомендаций, т.е. помощь в разрешении проблемных ситуаций до того, как они станут необратимыми.

В работе дается описание основных возможностей новой версии инструментального комплекса конструирования интеллектуальных (экспертных) систем реального времени G2, базовая версия которого [5-7] совместно с расширением GDA (G2 Diagnostic Assistant) – визуальной средой программирования, предназначенной для разработки приложений, ориентированных на моделирование, диагностику и аварийное управление сложными производственными процессами, использовалась в учебном процессе и научных исследованиях на кафедре Прикладной математики [8-11].

1. ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ И АРХИТЕКТУРА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ

1.1. Архитектура и основные принципы построения ИСППР РВ

Как уже отмечалось, для реализации ИСППР РВ необходимо использования современных технологий конструирования интеллектуальных систем, основанных на концепциях распределенного искусственного интеллекта, адаптивных моделей знаний, параллельной обработки информации при поиске решения на основе экспертных моделей и методов правдоподобного вывода, а также ориентируясь на мощные вычислительные платформы и соответствующие инструментальные комплексы типа G2.

Базовая архитектура ИСППР РВ представлена на рис. 1.1.

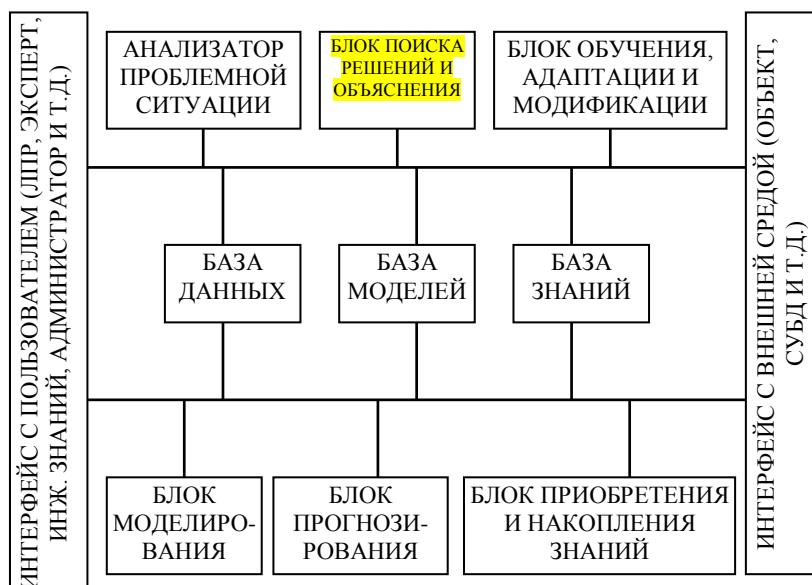


Рис. 1.1. Базовая архитектура ИСППР РВ

В отличие от традиционных экспертных систем [5] в ИСППР РВ необходимо включение дополнительных блоков моделирования и прогнозирования для возможности анализа и оценки последствий принимаемых решений и выбора наилучших рекомендаций.

ИСППР РВ является, по сути, системой распределенного интеллекта семиотического типа [12,13], включающей ряд взаимодействующих между собой интеллектуальных модулей (блоков), выполняющих соответствующие интеллектуальные функции. К числу таких блоков (помимо традиционных для статических экспертных систем [5] баз данных и знаний, решателя, блоков накопления и пополнения знаний, объяснения и организации взаимодействия с пользователем) относятся блоки моделирования (имитации) проблемной ситуации, прогнозирования, связи с внешними объектами (датчиками, контроллерами, концентраторами данных и т.д.), организации различных видов интерфейса (образного,

текстового, речевого, в виде различных графиков и диаграмм) с ЛПР. К интеллектуальным относятся также функции вывода (поиска) решения на базе моделей и методов представления и оперирования динамическими знаниями, характеризующимися неопределенностью, нечеткостью, неполнотой и противоречивостью. Поиск решения осуществляется с использованием механизмов неклассических, в том числе, псевдофизических (пространственно-временных, причинно-следственных) и немонотонных (абдуктивных, умолчания и т.д.) логик, а также механизмов обобщения и пополнения динамических знаний и методов на основе аналогий и прецедентов [11-17].

К основным принципам построения ИСППР относятся следующие:

- семиотичность;
- адаптивность и модифицируемость;
- распределенная и параллельная обработка информации;
- когнитивная графика и гипертекст в отображении информации.

Выбор инструментального комплекса G2 для реализации ИСППР РВ обусловлен интеграцией в нем современных высокоэффективных технологий разработки сложных программных продуктов:

- объектно-ориентированной технологии программирования, а также представления данных и знаний;
- технологии открытых систем и систем «клиент-сервер»;
- активной объектной графики;
- структурированного естественного языка и гипертекста для представления информации, обеспечивающих дружественный интерфейс с различными типами пользователей (ЛПР, администратор системы, эксперт, инженер знаний, программист);
- методов поиска (вывода) решения, основанных на различных механизмах достоверного и правдоподобного вывода: продукционных правилах, процедурах, динамических (имитационных) моделях, моделях на основе нечеткой логики;
- параллельного выполнения в реальном времени независимых процессов;
- сочетания технологии интеллектуальных (экспертных) систем, основанных на знаниях, с технологией традиционного программирования.

В состав базовых средств G2, необходимых для конструирования ИСППР РВ, входят:

- интерактивный редактор;
- средства графического интерфейса с пользователем, объектно-ориентированной графики и графических мониторинговых окон реального времени;
- средства анимации;

- средства отображения связей между объектами;
- средства взаимодействия с внешней средой, имитационного моделирования и обработки сложных правил и процедур;
- средства организации сообщений и объяснений.

Комплекс G2 разработан для различных вычислительных платформ (SUN, HP, IBM) и операционных систем (Unix/Solaris, Windows). В пособии рассматривается версия, ориентированная на ОС Windows NT/XP.

Рассмотрим основные принципы построения ИСППР РВ и средствами для их реализации, имеющимися в комплексе G2.

1.2. ИСППР РВ как динамическая система семиотического типа

ИСППР *семиотического типа* может быть задана набором [11]

$$SS = \langle M, R(M), F(M), F(SS) \rangle,$$

где $M = \{M_1, \dots, M_n\}$ – множество формальных или логико-лингвистических моделей, реализующих определенные интеллектуальные функции;

$R(M)$ – множество правил выбора необходимой модели или совокупности моделей в текущей ситуации, т.е. правил, реализующих отображение $R(M): S \rightarrow M$, где S – множество возможных ситуаций (состояний), которое может быть и открытым, или $S' \rightarrow M$, где S' – некоторое множество обобщенных ситуаций (состояний), например, нормальных (штатных), аномальных или аварийных, при попадании в которые происходит смена модели;

$F(M) = \{F(M_1), \dots, F(M_n)\}$ – множество правил модификации моделей $M_i, i = 1, \dots, n$. Каждое правило $F(M_i)$ реализует отображение $F(M_i): S'' \times M_i \rightarrow M'_i$, где $S'' \subseteq S$, M'_i – некоторая модификация модели M_i ;

$F(SS)$ – правило модификации собственно системы SS – ее базовых конструкций $M, R(M), F(M)$ и, возможно, самого правила $F(SS)$, т.е. $F(SS)$ реализует целый ряд отображений (или комплексное отображение) $F(SS): S''' \times M \rightarrow M', S''' \times R(M) \rightarrow R'(M), S''' \times F(M) \rightarrow F'(M), S''' \times F(SS) \rightarrow F'(SS)$, где $S''' \subseteq S, S''' \cap S' = \emptyset, S''' \cap S'' = \emptyset$, т.е. правила модификации данного типа используются в ситуациях, когда имеющихся множеств моделей, правил выбора и правил модификации недостаточно для поиска решения (решений) в сложившейся проблемной ситуации. Причем для модификации $F(SS)$ могут быть использованы как внутренние средства порождения моделей и правил (гипотез), так и внешние метазнания, отражающие прагматический аспект проблемной ситуации.

Каждая из моделей системы ориентирована на обработку некоторого («своего») типа неопределенности. В частности, методы и реализованные на их основе программные средства, использующие для оперирования неполной и противоречивой информации аппарат приближенных множеств, эффективны в ситуациях так называемой максимальной неопределенности, когда нет никакой дополнительной информации

(контекста) о проблеме. По мере поступления дополнительной информации, например, в виде вероятностей для правил, может быть применен интегрированный подход (модель), сочетающий методы теории приближенных множеств и теории вероятностей, что повышает степень правдоподобия (уверенности) выдаваемых системой рекомендаций (решений) ЛПР. При наличии полной информации о проблемной ситуации решения могут быть достоверными (при условии, что в базе знаний системы также имеется необходимая информация). Концепция использования интегрированной среды, включающей различные модели и методы поиска решения в условиях различного рода неопределенностей, реализуется в прототипе ИСППР РВ семиотического типа для управления сложными объектами и процессами типа энергоблока.

1.3. Адаптивность и модифицируемость

Рассмотрим ИСППР РВ, предназначенную для помощи ЛПР, управляющего сложным технологическим объектом или процессом, типа объекта энергетики [8,10]. Детальная структура ИСППР РВ представлена на рис. 1.2 и является конкретизацией обобщенной базовой архитектуры ИСППР РВ (см. рис. 1.1).

Основу ИСППР РВ семиотического типа, ориентированной на открытые и динамические проблемные/предметные области (ПО) составляет адаптивная, способная к модификации модель представления знаний и поиска решения.

Такую модель продукционного типа формально можно определить набором

$$\langle A, P, ST, P', ST', R_1, R_2, R_3, F \rangle,$$

где A – конечный алфавит, используемый для описания состояний или множеств состояний ПО;

P – начальное (или текущее) множество продукционных правил, используемых для преобразования состояний ПО (управляемого объекта);

ST – начальное множество стратегий поиска решения;

P', ST' – множества, используемые для пополнения P и ST ;

R_1 – правила выбора стратегии поиска решения из ST ;

R_2, R_3 – правила пополнения множеств P и ST ;

F – правила модификации модели (расширения алфавита, модификации множеств P' и ST' , правил выбора и пополнения).

Модель *адекватна* ПО, или *семантически корректна*, если любое состояние ПО (объекта) из множества допустимых начальных состояний переводится в состояние из множества конечных (целевых) состояний и процесс преобразования состояний конечен.

Проблема конструирования корректной модели включает в себя как задачи синтеза (системы продукционных правил, стратегий поиска, соответствующих решающих деревьев, правил пополнения и модификации

модели), так и задачи анализа (семантической корректности, возможностей оптимизации и редукции модели с целью минимизации времени поиска решения, методов обработки неопределенности и распараллеливания и т.д.). Важным фактором при реализации адаптивной модели является включение механизмов обучения на основе механизмов обобщения [14,18], аналогий и прецедентов [17], а также активно развиваемых в последнее время методов подкрепленного обучения [19].

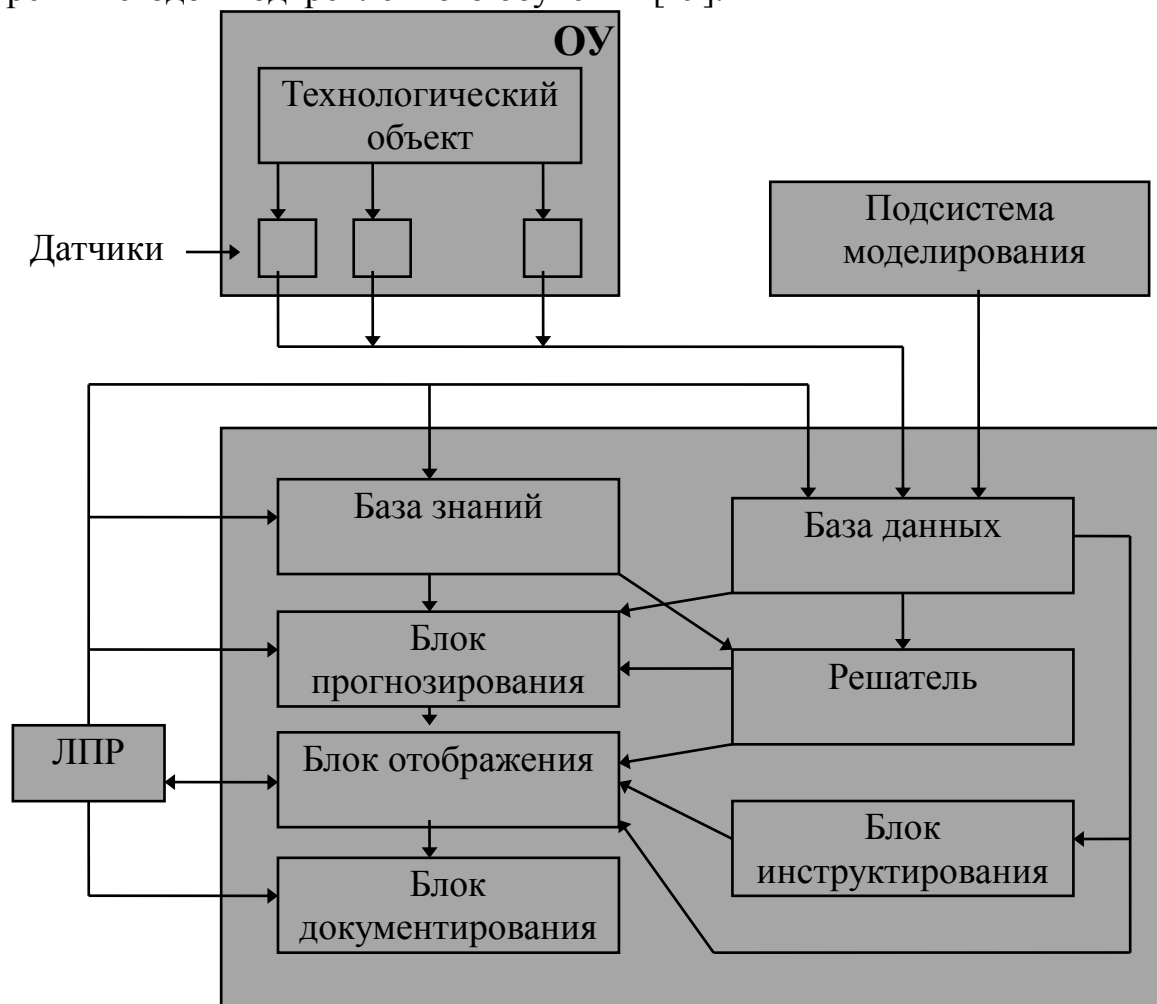


Рис. 1.2. Структура ИСППР РВ для управления технологическим объектом

Заметим, так как ИСППР РВ предназначены для помощи ЛПР в нештатных (**плохо определенных**) ситуациях, возникающих на сложных объектах, то они, как ранее отмечалось, как правило, работают в режиме советчика без непосредственного воздействия на объект, хотя при необходимости может быть организовано и непосредственное управление, например, на основе программируемых логических контроллеров (PLC), интерфейс с которыми имеется в G2. Как уже отмечалось, помимо традиционных для экспертных систем блоков в ИСППР РВ необходимо включение дополнительных блоков моделирования и прогнозирования для возможности анализа последствий принимаемых решений и выбора наилучших рекомендаций для ЛПР. Кратко остановимся на основных компонентах ИСППР РВ.

База данных (БД) – хранилище информации об объекте управления, поступающей с датчиков, от подсистемы моделирования и от ЛПР. Данные в БД представляют собой совокупность непрерывных и дискретных параметров объекта или его подсистем. Для представления элементов БД используется концепция иерархии классов в объектно-ориентированной технологии представления данных и знаний (рис. 1.3).

Определение классов дается посредством специальных таблиц, включающих такие атрибуты, как: имя класса (Class name), имя класса родителя (Superior class), описание специфических атрибутов класса (Class specific attributes). Базовым классом параметров объекта gcn (главного циркуляционного насоса системы охлаждения реактора) является класс gcn-parameter – наследник класса object. Каждый параметр имеет развернутое имя для представления ЛПР, что отражено атрибутом fullname. Для отображения непрерывных и дискретных параметров соответственно определяются два подкласса класса gcn-parameter – continuous-parameter и discrete-parameter.

Все параметры отображаются в виде иконок (некоторых простых образов). Цвет иконки параметра изменяется специальными производственными правилами в зависимости от положения текущего значения относительно диапазона допустимых значений (уставок). В нормальном состоянии – зеленый цвет. При выходе за диапазон – красный цвет. Изменение цвета происходит по производственным правилам типа *whenever* при получении значения атрибутом state, например:

whenever the state of any continuous-parameter CP receives a value and when not (the state of CP is normal) then change the body icon-color of CP to red.

Атрибуту state значение также присваивается по правилу *whenever* при получении непрерывным параметром текущего значения, например:

whenever the cur-value of any continuous-parameter CP receives a value and when the cur-value of CP > the top-set of CP then conclude that the state of CP is upper.

ЛПР может изменять текущее значение любого параметра, если по каким-либо причинам перестал функционировать источник данных параметра (например, датчик).

База знаний (БЗ) содержит экспертные знания, на основе которых проводится анализ состояния объекта. Вид, в котором представлены знания в БЗ, определяется выбранной моделью представления знаний. Большое распространение в интеллектуальных системах получила производственная модель с правилами типа:

Если <антецедент>, то <консеквент1> [, иначе <консеквент2>].

В ИСППР РВ антецедент обычно представляет собой логическое выражение относительно параметров состояния объекта. Консеквент описывает некоторое действие, которое следует предпринять в случае истинности (*консеквент1*) или ложности (*консеквент2*) антецедента.

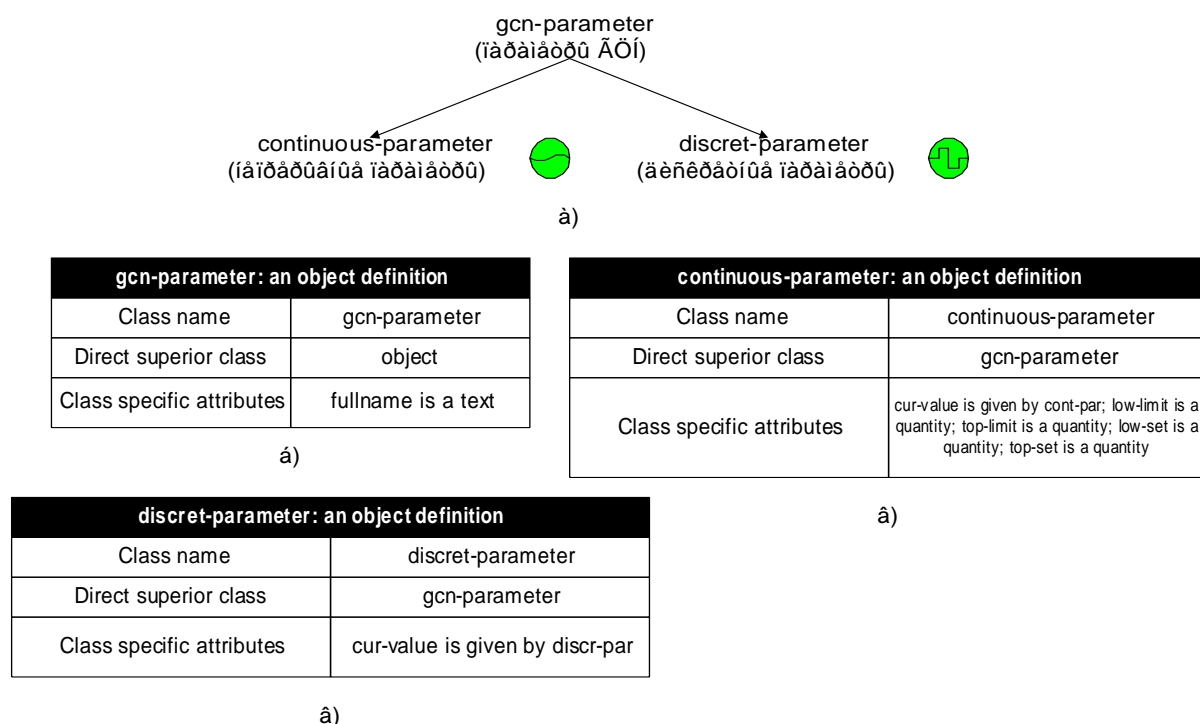


Рис. 1.3. Иерархическая организация базы данных (см. рисунок, подправить, убрать а!)

БЗ должна удовлетворять ряду требований, среди которых: наличие знаний, позволяющих проводить анализ состояния объекта при неполной информации; наличие для ЛПР возможности корректировки БЗ в рабочем режиме, и, следовательно, наличие развитых средств контроля вводимых данных на корректность, так как ошибочные изменения в БЗ при функционировании объекта недопустимы. Вышеперечисленные требования обуславливают необходимость представления информации в БЗ в наиболее удобном для восприятия ЛПР графическом виде, в частности, в виде граф-схем (решающих деревьев) и когнитивных образов.

Фрагмент внешнего представления БЗ приведен на рис. 1.4.

Решатель – модуль, который реализует алгоритм (стратегию) поиска решения на основе применения знаний из БЗ к данным из БД. Решатель, как и БЗ, являются компонентами рассмотренной выше адаптивной модели. При реализации решателя целесообразно использовать преимущества объектно-ориентированной технологии. Решатель при этом представляет собой ряд виртуальных методов, определенных для главного класса (схемы объекта) и его подклассов, и используемых при поиске (выводе) решения. Стратегия (алгоритм) поиска решения формируется на основе анализа ситуации (состояния объекта или его подсистемы) с учетом специфики поступающей информации и информации, хранящейся в БЗ (детерминированная или недетерминированная, какова природа неопределенности и т.д.). Разработчик может пополнять и модифицировать методы, определяя новые стратегии поиска. Таким образом реализуется открытость и динамичность модели и в целом ИСППР РВ.

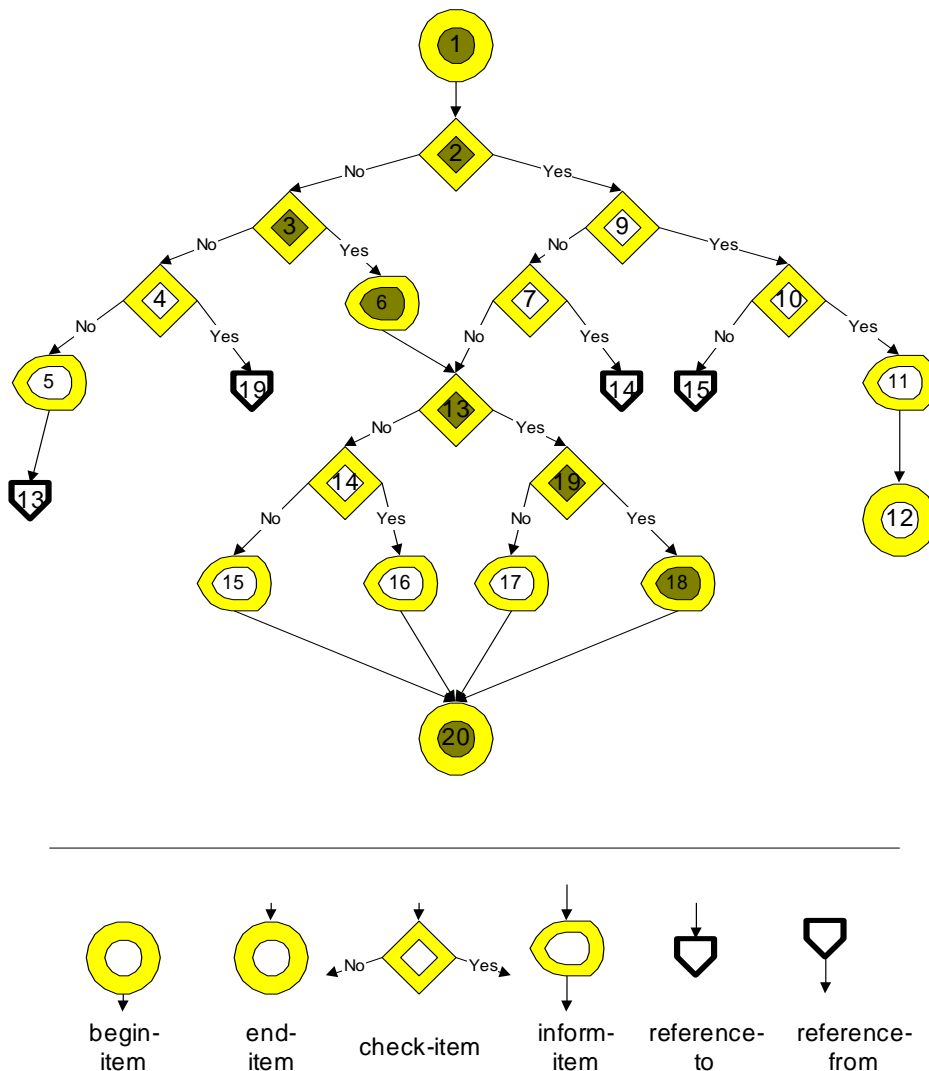


Рис. 1.4. Фрагмент представления БЗ в виде граф-схемы

Блок прогнозирования осуществляет функции прогнозирования аномальных ситуаций и последствий управляющих воздействий. Прогнозирование производится по команде ЛПР на основе данных о текущем состоянии управляемого объекта, поступающих из БД, и знаний, хранящихся в БЗ, с применением методов как достоверного, так и правдоподобного вывода.

Блок инструктирования направляет действия ЛПР в запланированных переходных режимах. Он подключается автоматически (по ситуации) при наступлении соответствующего режима функционирования объекта, информация о котором поступает из БД. Блок инструктирования организует при необходимости диалог с ЛПР (например, для уточнения исходной информации или пополнения БЗ) и выполняет совместно с блоком отображения информации функции взаимодействия с пользователем.

Блок отображения информации выполняет функции представления информации ЛПР. Исходными данными для него являются данные из БД, результаты оценки состояния объекта, полученные решателем, результаты прогнозов, сделанных блоком прогнозирования, и инструкции, выдаваемые блоком инструктирования.

Информация, с одной стороны, должна отображаться в удобном для быстрого восприятия ЛПР виде и, с другой стороны, должна быть как можно более полной. Эти требования противоречат друг другу, так как при увеличении объема знаковой информации уменьшается способность человека воспринимать ее. Проблема решается посредством многоуровневой схемы отображения информации с применением технологии гипертекста и когнитивной графики, о чем подробнее будет сказано ниже.

Устройствами отображения информации в ИСППР РВ обычно являются дисплей рабочей станции и дисплеи рабочих мест. Неподвижное изображение на экране соответствует статическому состоянию объекта, а движение отображает переход объекта в новое состояние.

Для отображения информации вводится ряд рабочих пространств.

Рабочее пространство актуальных сообщений. При нормальном состоянии объекта в рабочем пространстве актуальных сообщений находится только одно сообщение "Объект (система) в норме". Это сообщение имеет зеленый фон. При возникновении аномальных или аварийных (критических) ситуаций в рабочем пространстве появляются сообщения на желтом или красном фоне.

Рабочее пространство с изображением мнемосхемы объекта. Датчики представляются по группам графическими образами фиксируемых ими параметров объекта. Динамика процессов, происходящих в объекте, отображается изменением цвета графических образов параметров. Глядя на схему, ЛПР может качественно оценить состояние объекта или его подсистемы и определить, какие параметры находятся вне диапазона допустимых значений (выделяются красным цветом).

Рабочее пространство БЗ. В нем расположена граф-схема (дерево решений), отображающее динамику функционирования объекта. ЛПР использует это пространство для коррекции граф-схемы в рабочем режиме. В основном же это пространство предназначено для эксперта и инженера по знаниям при создании и тестировании БЗ (АМПР).

Рабочее пространство параметров. Предназначено для представления графиков изменения наиболее важных для ЛПР параметров, отображающих динамику их изменения в реальном масштабе времени.

На рис. 1.5 представлен пример полиэкрана для ЛПР, содержащего рассмотренные рабочие пространства, для прототипа ИСППР РВ, созданного в среде G2 и предназначенного для оперативно-диспетчерского

персонала атомного энергоблока (объектом является одна из подсистем энергоблока – эжекторная установка системы охлаждения) [11].

Блок моделирования моделирует (имитирует) поведение объекта. Он может выступать как агент данных о состоянии объекта на этапах тестирования системы, собственно принятия решений для сравнения с данными, поступающими от датчиков, а также для тренировки и обучения оперативно-диспетчерского персонала (ЛПР). Блок моделирования также может быть использован совместно с блоком прогнозирования для прогноза аномальных ситуаций и последствий управляющих воздействий.

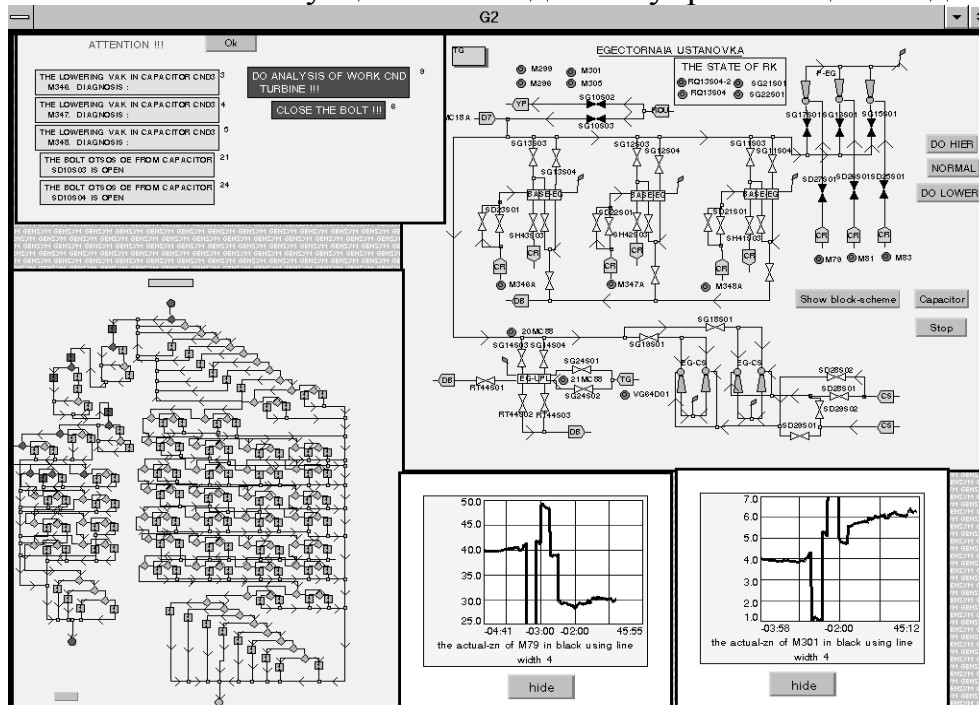


Рис. 1.5. Пример полиэкрана для ЛПР

1.4. Распределенная и параллельная обработка информации

При реализации ИСППР РВ, ориентированных на помощь оперативно-диспетчерскому персоналу (ЛПР) при управлении сложными объектами и процессами, одной из основных проблем является необходимость поиска приемлемого решения в условиях жестких временных ограничений. Число анализируемых параметров для объекта типа энергоблока может достигать многих тысяч, что не позволяет провести достаточно качественный анализ информации на основе последовательной ее обработки. Поэтому естественным образом возникает задача распараллеливания процесса обработки информации в ИСППР РВ. Теоретические вопросы параллельной обработки информации в интеллектуальных системах, базирующихся на продукционно-сетевом механизме представления знаний, рассмотрены, в частности в работах [1,11,20]. Остановимся на возможности реализации средств параллельной обработки информации в ИСППР РВ, создаваемых на основе инструментального комплекса G2.

Рассмотрим возможности параллельной обработки информации в ИСППР РВ. Параллелизм, присущий самой природе решаемой задачи или исследуемого процесса, интерпретируется как внешний параллелизм, в отличие от внутреннего параллелизма, характеризующего собственно возможности реализации алгоритма (программы) решения задачи.

К внутреннему параллелизму относится мелкоструктурный (*fine-grain*) параллелизм на уровне программных (машинных) инструкций, к внешнему параллелизму – крупноструктурный (*large-grain*) параллелизм на уровне задач (программных модулей). Для реализации различных уровней параллелизма в ИСППР РВ необходимо наличие соответствующих программных средств поддержки, достаточно мощный набор которых имеется в комплексе G2.

Инструментальный комплекс G2 – полная (комплексная) среда (*complete environment*) для разработки, внедрения и сопровождения приложений в широком диапазоне ПО, интегрирующая в себе современные технологии программирования и обработки информации. Комплекс поддерживает реализацию следующих уровней внешнего и внутреннего параллелизма.

Параллелизм на уровне разработки. Возможен многопользовательский подход к созданию приложения. Каждый разработчик имеет доступ к БЗ, находящейся на сервере, причем приложение может быть реализовано не только на различных ЭВМ, но и с использованием нескольких G2, взаимодействующих между собой.

Параллелизм на уровне задач, программных модулей и процедур. В рамках одного приложения может одновременно работать несколько модулей, выполняющих определенные действия. В ИСППР РВ целесообразно разбиение всего программного комплекса на модули, которые соответствовали бы различным подсистемам управляемого объекта. Данный уровень можно трактовать как следующую (более низкую) степень внешнего параллелизма.

При параллельном выполнении ряда задач может возникнуть необходимость обмена какими-либо данными и соответственно проблема синхронизации данных. Для организации синхронного взаимодействия между блоками различных подсистем используются продукционные правила типа *whenever* и *when*.

Для распараллеливания процесса поиска решения внутри процедур и методов применяется конструкция *do in parallel*, которая служит для задания параллельного выполнения нескольких независимых операторов.

Параллелизм на уровне правил. В G2 существует несколько типов правил: *initially*, *unconditionally*, *whenever*, *when*, *if*.

Все правила начальной инициализации *Initially* выполняются параллельно при запуске приложения. По этим правилам можно запустить какую-либо процедуру, присвоить параметрам или переменным начальные

значения, открыть нужные для работы пространства. Правила типа *unconditionally* безусловно производят определенные действия при активизации и выполняются параллельно, как и правила *initially*.

Продукционные правила типа *when* и *whenever* похожи. Правило *when* активизируется, когда выполняется некоторое логическое условие. Правило *whenever* активизируется, когда совершается некоторое событие. Если в данный момент времени активны несколько правил *when* или *whenever*, то параллельно будут выполняться заключения каждого такого правила. Аналогично и для продукционного правила типа *if*, которое отличается от правила *when* тем, что на его основе можно строить сложные схемы рассуждений типа деревьев решений.

Параллелизм внутри правил. Помимо распараллеливания на уровне правил в G2 существует также возможность распараллелить процесс обработки и внутри самих правил, т.е. реализовать несколько уровней мелкоструктурного параллелизма. Для организации параллельного выполнения определенных действий для целого множества объектов одного и того же типа, применяются обобщенные правила.

Для построения обобщенных правил применяются конструкции *for* и *every*. Конструкция *for* позволяет выполнять определенные действия для объектов одного типа, например, параллельно выводить заключение о состоянии (исправен или нет) любого из них, а *every* позволяет выполнять определенные действия для некоторых атрибутов (параметров, переменных) объектов одного типа.

Возможны ситуации, когда параллельная обработка недопустима, так как действия правил могут зависеть друг от друга и выполнение одного из них должно предшествовать (инициировать) выполнение другого. Для таких случаев используется конструкция *in order*, с помощью которой устанавливается порядок выполнения определенных действий.

Для организации логических и арифметических операций и их параллельной обработки используются блоки *AND* (И), *OR* (ИЛИ), *N TRUE* (голосование), синхронизации и сдерживания.

На рис. 1.6 приведен пример полиэкрана для просмотра графиков изменения параметров в блоке прогнозирования прототипа ИСППР РВ.

ИСППР РВ должны обладать удобными для ЛПР средствами общения и отображения текущей ситуации на основе технологии когнитивной графики и гипертекста, которые позволяют ЛПР активно использовать механизмы не только поверхностного, но и глубинного уровней мышления [12].

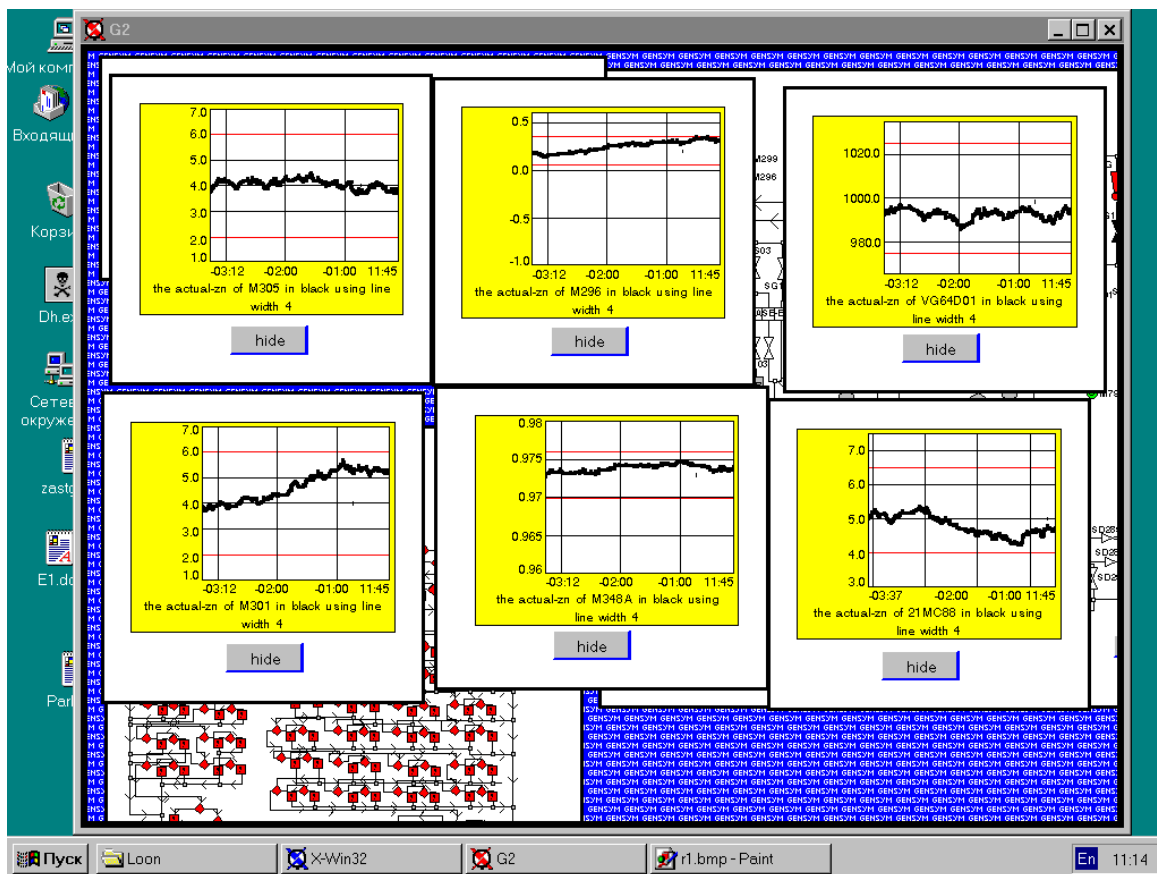


Рис. 1.6. Пример полиэкрана для отображения динамики изменения параметров

1.5. Когнитивная графика и гипертекст в отображении информации

Образное, графическое представление информации о решаемой задаче или управляемом объекте (процессе) удобнее текстового и по выразительности и по времени восприятия. Особенно это преимущество важно для ИСППР РВ, используемых при управлении сложными и критическими по времени процессами. Так, на диспетчерский пульт крупного энергоблока выводится информация, поступающая от нескольких тысяч датчиков. Оператор за очень ограниченное время должен осмыслить поступающую информацию и при необходимости принять решение о коррекции функционирования соответствующих подсистем. Поэтому оперативно-диспетчерский персонал часто испытывает стрессовые перегрузки. Однако известно [3], что нагрузка на оператора существенно снижается, если информация об отклонениях и аномальных ситуациях дается в графическом виде и в виде анимационных диаграмм. Технология гипертекста позволяет при этом раскрывать информацию до нужного ЛПР уровня детализации.

Поэтому целесообразно дополнить данные экраны еще и экранами с образным представлением информации средствами когнитивной графики. Расширенные графические средства позволяют, с одной стороны, добиться максимальной выразительности и восприимчивости информации ЛПР, и, с

другой стороны, обеспечить тот уровень информированности (понимания) ЛПР, который необходим для принятия решений.

Анализ деятельности оперативно-диспетчерского персонала сложного объекта типа энергоблока показал [3,4,8], что предпочтительнее использовать трехуровневую систему представления информации об управляемом объекте или процессе, включающую: уровень объекта (системы) в целом, на котором сообщается в каком (нормальном, аномальном или критическом) состоянии он находится, и в каких подсистемах возникли отклонения; уровень подсистемы, на котором выявляется состояние конкретной подсистемы; уровень непосредственно измеряемых параметров с указанием значений параметров и динамики их изменения.

Основной тип изображения соответствует уровню системы (объекта) или процесса в целом и имеет вид ядра с возможно отходящими от него лучами, соответствующими подсистемам (или обобщенным параметрам процесса) верхнего уровня. Число лучей определяется числом подсистем. Ядро представляет собой образ-лицо, которое может улыбаться, морщиться или кричать в зависимости от состояния объекта в целом или его подсистем.

Цвет ядра и лучей может быть красным, желтым или зеленым. Зеленый цвет ядра соответствует нормальному состоянию всей системы (процесса) в целом, желтый – аномальному состоянию, красный – критическому состоянию. Основной тип изображения всегда хранится на экране.

Изображение второго типа соответствует уровню подсистемы и характеризует ее состояние. Если подсистема (обобщенный параметр), в свою очередь, является сложной системой, то процесс «раскрытия» образа может быть рекурсивно продолжен.

Если наблюдаемая подсистема характеризуется совокупностью измеряемых параметров, то возникает кольцевое изображение третьего типа, соответствующее уровню параметров и динамике их изменения. Пример полиэкрана с использованием средств когнитивной графики, приведен на рис. 1.7.

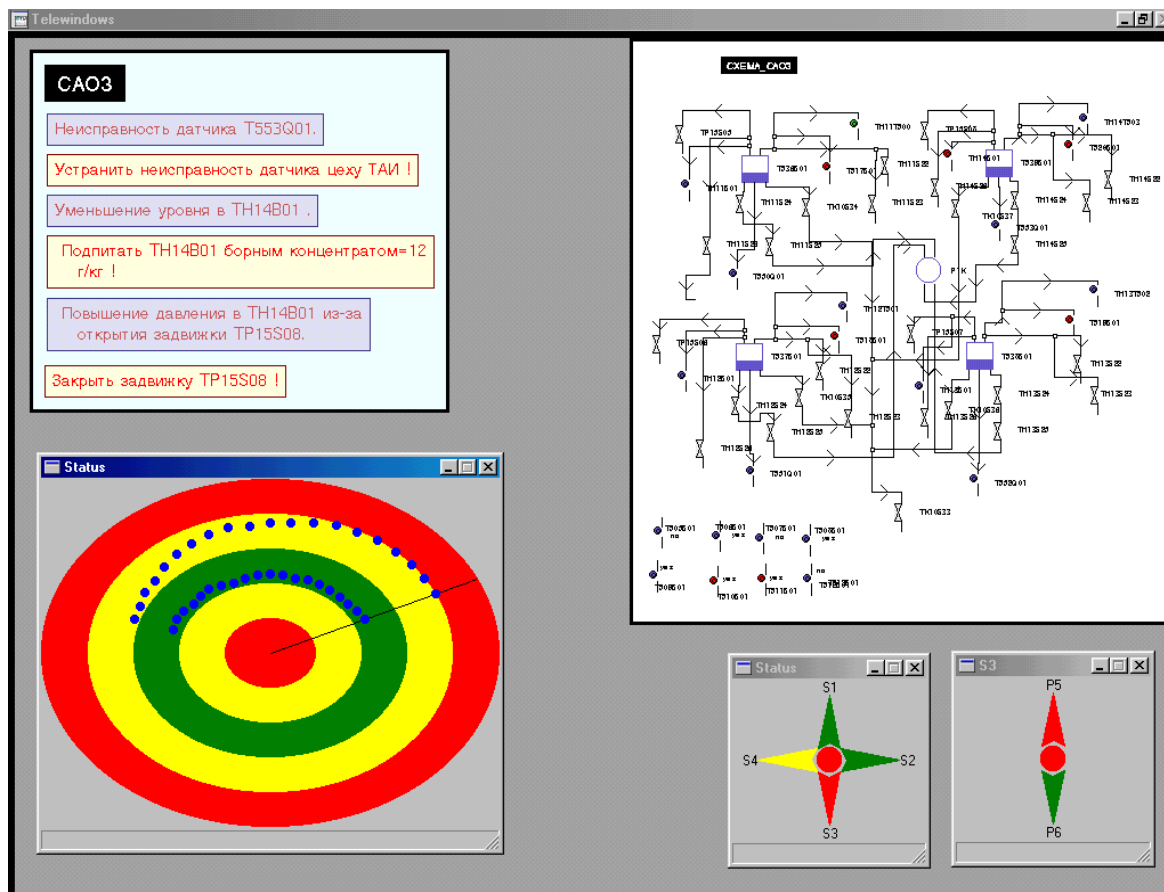


Рис. 1.7. Пример полиэкрана с использованием когнитивной графики (образное представление состояния подсистемы и параметров)

1.6. Контрольные вопросы к разделу 1

1. Дайте определение ИСППР РВ, назначение.
2. Приведите базовую архитектуру ИСППР РВ.
3. Перечислите основные принципы построения ИСППР РВ и обоснуйте выбор инструментального комплекса G2.
4. Определите ИСППР как систему семиотического типа.
5. Поясните принцип адаптируемости и модифицируемости.
6. Приведите и поясните структуру ИСППР РВ для управления технологическим объектом.
7. Поясните распределенную и параллельную обработку информации средствами G2.
8. Поясните использование когнитивной графики и гипертекста при отображении информации.
9. Поясните структуру полиэкрана с использованием когнитивной графики.

2. ОСНОВНЫЕ КОМПОНЕНТЫ КОМПЛЕКСА G2

2.1. Главное меню и меню рабочих пространств

Создаваемые в среде G2 приложения называются **базами знаний** (БЗ). БЗ содержит два основных типа знаний:

- данные: объекты (включая переменные, параметры, списки, массивы), атрибуты, отношения, связи;
- выполняемые утверждения: правила, процедуры, функции, формулы.

При разработке приложения используются **рабочие пространства** – *workspaces* (РП).

Для манипуляций в среде G2 используется клавиатура и мышь. Щелчок мышью используется для: вызова меню; выбора пункта меню; редактирования текста. Удержание клавиши мыши применяется для перемещения рабочих пространств, меню и элементов создаваемого приложения. Двойной щелчок мышью используется для обрыва «повисших» связей. Все клавиши мыши равноправны.

Работа в среде G2 начинается с вызова **главного меню** системы, изображенного с необходимыми пояснениями на рис. 2.1. Для создания РП верхнего уровня используется пункт *New Workspace* главного меню. Меню РП изображено на рис. 2.2. Разработчиком могут быть созданы дополнительные подпространства (*subworkspaces*), которые полностью аналогичны РП верхнего уровня. Для создания подпространств и перехода к ним из сущностей, с которыми они ассоциированы, используются соответственно команды *create subworkspace* и *go to subworkspace*. Таким образом реализуется иерархия РП.

2.2. Сущности

Для определения классов **сущностей** (*items*), используемых в БЗ, служат описания сущностей (рис. 2.3). Описания могут быть встроенными и определяемыми пользователем. Сущности подразделяются на **сущности для хранения данных** (рис. 2.4) и **выполняемые сущности/утверждения** (рис. 2.5). Для создания и манипулирования сущностями используется меню сущностей (рис. 2.6).

2.3. Редактирование выполняемых сущностей

Для редактирования выполняемых сущностей (правил, процедур, формул и т.п.) в G2 используется интерактивный текстовый редактор. Процесс редактирования все время направляется **процедурой грамматического разбора**, что гарантирует введение только синтаксически правильных конструкций. В окне редактирования появляется динамически изменяемая подсказка, указывающая, какие конструкции можно вводить, начиная с текущей позиции курсора. Можно выбирать подходящие

шаблоны из подсказки или набирать текст на клавиатуре. На рис. 2.7 представлены некоторые этапы редактирования общего правила.

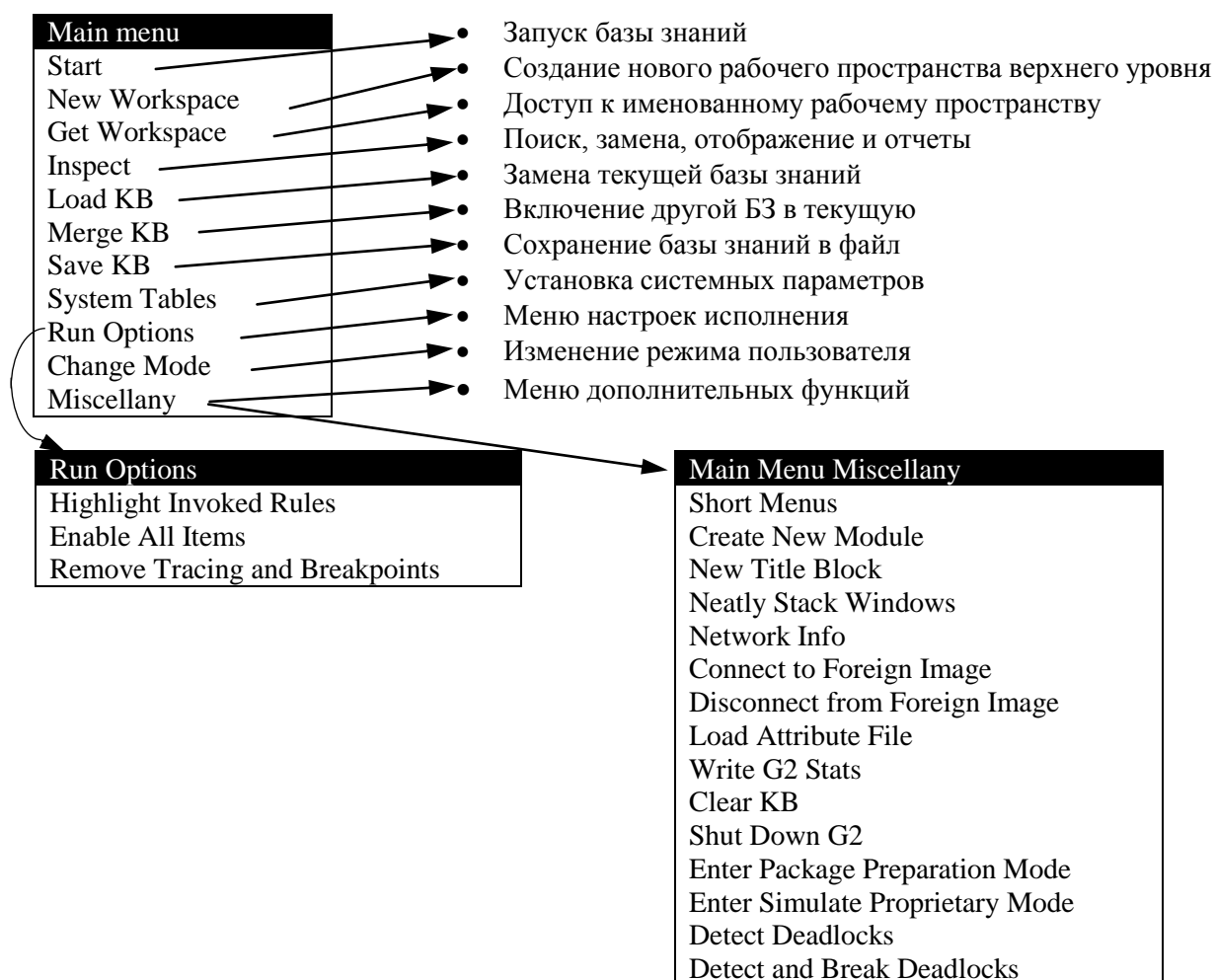


Рис. 2.1. Главное меню

KB Workspace	
New Object	Создание определенных ранее сущностей
New Rule	
New Display	
New Definition	
New Free Text	
New Button	
Name	Имя рабочего пространства
Clone Workspace	Копия рабочего пространства с содержимым
Table	Показать атрибуты
Color	Цвет фона, рамки и текста
Move	Переместить
Hide Workspace	Скрыть (пространство остается активным)
Lift to Top	Вверх стека видимых пространств
Drop to Bottom	Вниз стека видимых пространств
Shrink Wrap	Минимизировать размер рамки
Delete Workspace	Удалить из БЗ навсегда!
Disable	Временно игнорировать вместе с содержимым
Main Menu	Перейти в Главное Меню
Describe	Показать связь с другими сущностями
Table of Hidden Attributes	Показать скрытые атрибуты пространства
Describe Configuration	Показать конфигурацию
Print to Server	Создать файл для печати

Рис. 2.2. Меню рабочего пространства



Рис. 2.3. Описания сущностей

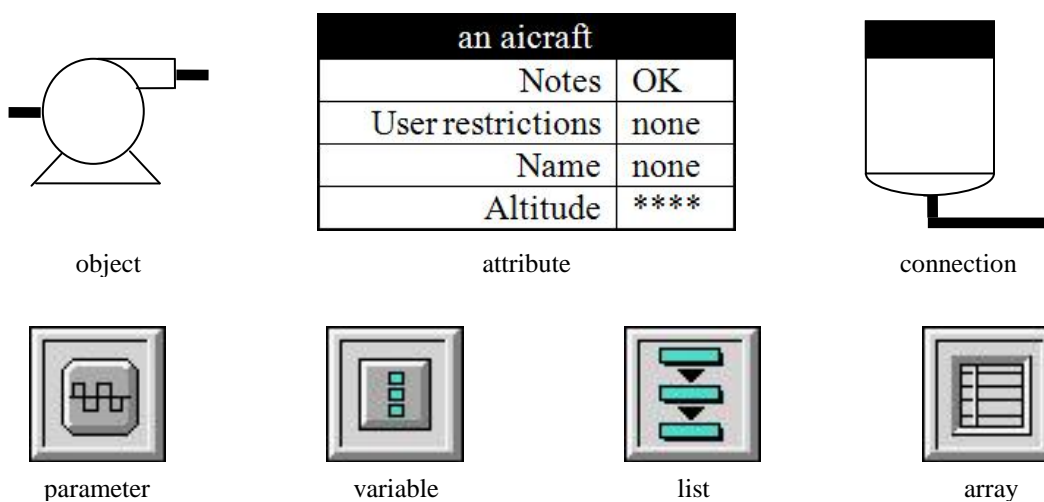


Рис. 2.4. Примеры сущностей для хранения данных

Правила

```
for any tank T
  if the rate of change per second of the volume of T during the last 1 second > 0
    then conclude that the level of T is increasing
```

Процедуры

```
update-volume(N: integer)
T: class tank;
begin
  for T = each tank do
    conclude that the volume of T = the volume of T + 1;
  end;
end
```

Формулы

```
let the current-level of any tank
= the volume of the tank / the area of the tank
```

Функции

```
cube(x) = x * x * x
```

Формулы моделирования

```
state variable: d/dt (the volume of any tank) =
  the inflow of the tank - the outflow of the tank, with initial value 0
```

Рис. 2.5. Примеры выполняемых сущностей/утверждений

object	object definition	generic formula
table	table	table
transfer	create instance	edit
name	edit icon	transfer
clone	transfer	clone
rotate/reflect	clone	align text
change size	rotate/reflect	color
delete	color	font
disable	delete	delete
describe	lift to top	lift to top
describe restrictions	drop to bottom	drop to bottom
create subworkspace	disable	disable
	describe	describe
	table of hidden attributes	table of hidden attributes
	show unsaved attributes	show unsaved attributes
	describe configuration	describe configuration
	create subworkspace	

Рис. 2.6. Примеры меню сущностей

2.4. Выполнение приложения

Для управления работой приложения существуют следующие команды, доступные из главного меню: *Start*, *Restart*, *Reset*, *Pause* и *Resume*. Во время работы приложения или во время паузы можно пополнять БЗ.

Редактирование кнопок допустимо только во время паузы. Временные значения сущностей теряются при *Reset* и *Restart*.

2.5. Представление конкретных знаний

База знаний в G2 представляет собой совокупность следующих элементов:

- объектов и атрибутов для представления данных;
- команд для присвоения значений и выполнения других действий;
- правил продукционного типа для представления событий, выработки заключений и для сканирования заданных областей;
- процедур для последовательной обработки;
- связей для отображения взаимоотношений между объектами.

2.5.1. Типы данных и атрибуты

В G2 предусмотрены следующие типы конкретных (specific) данных:

- целое;
- вещественное;
- символ;
- текст;
- истинностное значение.

Данные обычно представляются в виде атрибутов объектов, которые хранятся в таблице атрибутов. Для ссылки на соответствующий атрибут необходимо применение артикля *the*, например:

the flow of pump-3;
the power of pump-3;
if the flow of pump-3 = 0 and the power of pump-3 is on
then inform the operator that "Насос-3 неисправен".

Атрибуты могут быть встроенными и введенными пользователем. Присваивание значений атрибутов может производиться вручную или по команде *conclude that*:

- для символов и истинностных значений используется **IS**: *conclude that the power of pump-3 is off*;
- для чисел и текста используется знак равенства: *conclude that the flow of pump-3 = 5*;
- при употреблении выражения в правой части также используется знак равенства: *conclude that the power of pump-3 = the power of pump-2*.

Проверка типов данных выполняется только для параметров и переменных.

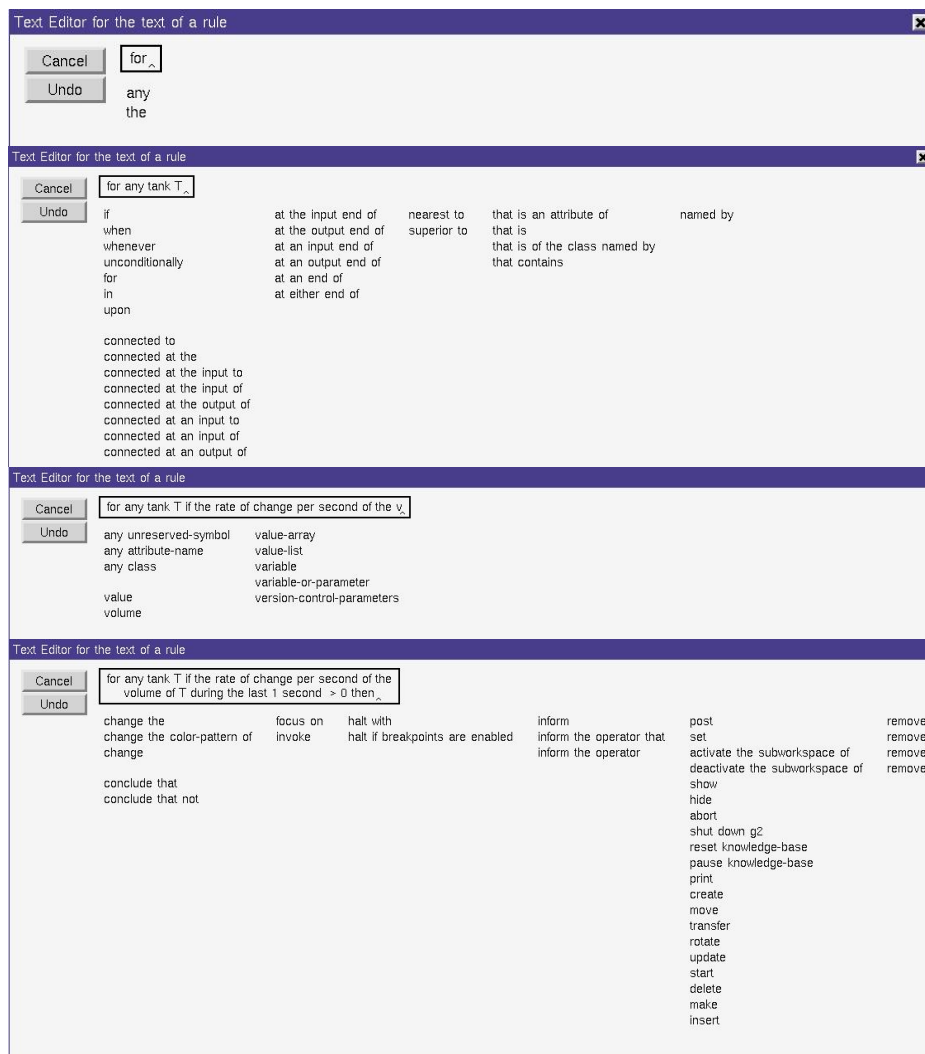


Рис. 2.7. Некоторые этапы редактирования общего правила
(ссылка на рис. 2.7 на стр. 21, а сам рис. 2.7 – на стр. 25!)

2.5.2. Команды

Команды исполняются во время работы приложения. Использовать команды можно в правилах, процедурах, пользовательских командах меню и в командных кнопках. Команды подразделяются на следующие функциональные группы:

1. присвоения значений и динамического представления сущностей: *conclude* – заключить, *change* – изменить, *set* – установить, *insert* – вставить, *remote* – удалить, *create* – создать, *delete* – удалить, *make permanent* – сделать постоянным, *make transient* – сделать временным;
2. интерфейса с пользователем: *inform* – информировать, *show* – показать, *hide* – скрыть, *print* – печатать, *move* – переместить, *transfer* – передать, *rotate* – повернуть, *update* – обновить;
3. управления исполнением приложения: *start* – запустить, *abort* – сбросить, *focus* – фокусироваться (на правилах, которые касаются определенного объекта), *invoke* – возбудить (правила определенной категории), *activate* – активировать, *deactivate* – деактивировать, *reset knowledge-base* –

ВОССТАНОВИТЬ начальное состояние БЗ, *pause knowledge-base* – приостановить работу БЗ, *halt* – останов, *shut down G2* – закончить работу G2.

2.5.3. Командные кнопки

Для создания **командной кнопки** используется меню РП: *KB Workspace → New Button → action button*.

Таблица атрибутов командной кнопки доступна в режимах *Pause* и *Reset*. В процессе исполнения приложения щелчок мышкой вызывает исполнение действия, приписанного кнопке. Конфигурирование командной кнопки осуществляется посредством редактирования ее атрибутов. Посредством связки *and* в командных кнопках, правилах и пользовательских командах меню можно сформировать список команд. По умолчанию команды из списка выполняются параллельно. Установка по умолчанию может быть изменена директивой *in order*.

2.5.4. Правила

В G2 существует пять типов правил: *whenever*, *if*, *when*, *initially* и *unconditionally*.

1. Правило *whenever*

- G2 **возбуждает (активирует)** правила *whenever* в ответ на наступление событий.
- Правило *whenever* состоит из двух частей:
 - антецедента: часть “*whenever*”;
 - консеквента: часть “*then*”.
- Часть “*whenever*” описывает событие:
 - перемещение объекта;
 - вступление объекта в отношение с другим объектом;
 - неудача при получении значения для переменной.
- “*Whenever*” может включать дополнительные условия.
- Часть “*then*” содержит список команд, исполняемых G2 в ответ на событие: присвоение значение, запуск процедур, информирование пользователей.

Пример правила *whenever*:

*whenever car-21 is moved and when the item-x-position of car-21 < 300
then move car-21 by (15,0)*

2. Правило *if*

- Правила *if* используются при обработке, управляемой потоком данных, и **активируются** при присваивании значений атрибутам.
- Вызывают «прямой вывод» (*forward chaining*): от данных к цели.
- Правила *if* содержат неявную ссылку на событие.

- Правила *if* могут активироваться в результате: прямого вывода, обратного вывода (*backward chaining*), сканирования и т.д.

Пример правила *if*:

*if the power of pump-45 is on and the flow of pump-45 = 0
then conclude that the status of pump-45 is broken*

3. Правило **when** аналогично правилу *if*, но оно, в отличие от правила *if*, не вовлекается в прямой и обратный вывод.

4. Правило **initially** используется для выполнения некоторых начальных действий, например:

initially

*show this workspace with its top left corner at the top left corner of screen
and change the background-color of this workspace to yellow*

5. Правило **unconditionally** используется для безусловного выполнения некоторых действий.

Правило в G2 может активироваться одним из 9 способов:

1. данные, входящие в условие правила, изменились (используется с прямым выводом);
2. правило определяет значение переменной, которое требуется другому правилу или процедуре (используется в обратном выводе);
3. активирование правила каждые n секунд, где n – число, определенное для данного правила (используется в механизме сканирования);
4. явное или неявное активирование другим правилом – путем применения операций *focus* и *invoke*;
5. переменной, входящей в условие правила, присвоено значение независимо от того, изменилось оно или нет;
6. активирование правила каждый раз при запуске приложения;
7. определенный объект на экране перемещен пользователем или другим правилом;
8. определенное отношение между объектами установлено или уничтожено;
9. переменная не получила значения в результате обращения к своему источнику данных.

2.5.5. Процедуры

В G2 имеется PASCAL-подобный язык для записи процедур. Для определения процедуры используется меню РП: *KB Workspace* → *New Definition* → *procedure*.

Пример записи процедуры приведен на рис. 2.8.

Типы данных в G2 объединяются в иерархию:

item-or-value

- value
 - quantity

- integer
- float
- symbol
- text
- truth value
- time stamp
- class (включает все встроенные и определенные пользователем сущности).

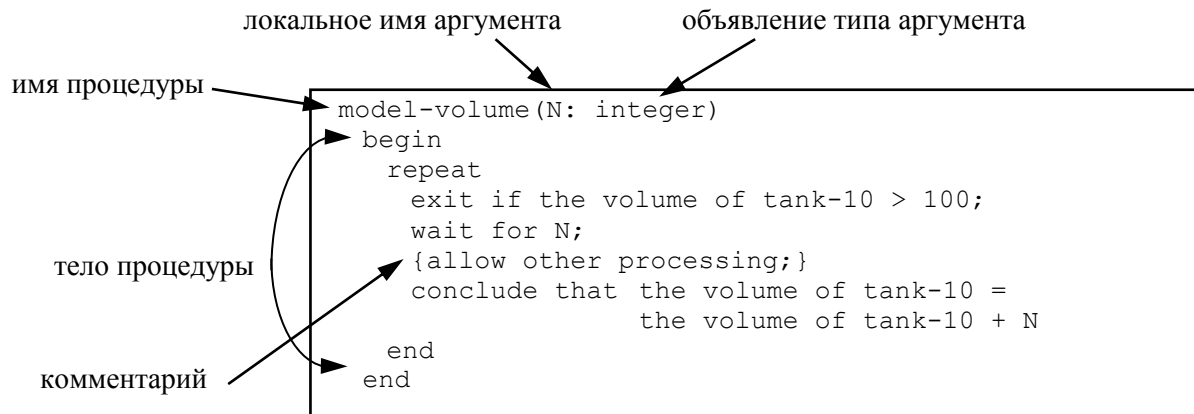


Рис. 2.8. Пример процедуры

Локальное имя аргумента может ссылаться на любой тип данных. Если локальное имя аргумента ссылается на сущность, то необходимо указать класс, экземпляром которого является данная сущность (см. пример **ниже**). В списке аргументов процедуры определения локальных имен аргументов разделяются запятыми.

Пример определения локальных имен аргументов в процедуре:

```

model-volume-31 (N:integer, MSG: text, T: class new-tank)
begin
  conclude that the volume of T = the volume of T + N;
  inform the operator for the next 2 seconds that "[MSG][the name of T]"
end
  
```

В процедурах могут использоваться следующие утверждения: *repeat* – повтор, *allow other processing* – допуск другой обработки, *exit* – выход из цикла; атрибут *Uninterrupted-procedure-execution-limit* (указывает лимит времени непрерываемого выполнения процедуры); конструкция *for* с использованием счетчиков. Например:

```

model-volume-t5 (N: integer)
begin
  repeat
    conclude that the volume of tank-t5 = the volume of tank-t5 + N;
    allow other processing;
    exit if the volume of tank-t5 > 60.0
  end
end
end
  
```

Оператор “exit if <логическое выражение>” может использоваться в любом цикле, а не только в цикле *repeat*.

```
display-messages (N1: integer)
N2: integer;
begin
  for N2 = N1 down to 1 do
    inform the operator for the next 4 seconds that “[N2]”
  end
end
```

Для организации ветвлений и переходов используются операторы: *if* (в том числе вложенные *if*), *go to*, *case* и блоки *begin...end* в операторе *if*.

Операторы могут помечаться символами или числами. Пример процедуры с использованием оператора *case*:

```
example-case()
begin
  case (the current hour) of
    9, 10, 11: inform the operator for the next 5 seconds
               that “Сейчас утро”;
    12: begin
          inform the operator for the next 5 seconds that “Время обеда”;
          pause knowledge-base
        end
    13, 14, 15, 16: inform the operator for the next 5 seconds
                   that “Сейчас день”;
    otherwise: inform the operator for the next 5 seconds
              that “Ты не заработался, дружок?”;
  end
end
```

В процедурах могут быть организованы задержки:

- на указанный временной интервал;
- на время ожидания истинности логического выражения;
- на время ожидания наступления заданного события.

Для этого используется оператор *wait*, который имеет следующий синтаксис:

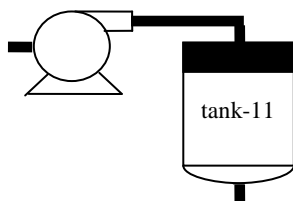
```
wait { for <интервал> |
      until <логическое выражение> checking every <интервал> |
      until <предикат события> }
```

Например: *wait until the volume of tank-1 receives a value.*

Для вызова процедур используются операторы *call* и *start*. *Call* отличается от *start* тем, что главная процедура не продолжится, **пока не завершится вызванная процедура**. *Start* запускает самостоятельную процедуру, а главная процедура сразу же продолжает свою работу.

Оператор *return* возвращает значение в вызвавшую процедуру.

Для представления взаимосвязей между объектами используются связи. На сущности можно ссылаться посредством связей. Это особенно удобно при записи общих утверждений. Пример использования связей приведен на рис. 2.9.



```

volume-thru-connection(N: integer)
begin
  repeat
    exit if the volume of tank-11 > 100;
    wait for N;
    conclude that the volume of tank-11 =
      the volume of tank-11 + the flow
        of the new-pump connected to tank-11 * N
  end
end

```

Рис. 2.9. Пример использования связей

2.6. Представление общих знаний

2.6.1. Общие правила

Общие правила позволяют снизить трудоемкость разработки и сопровождения созданного приложения, сократить избыточность информации и возможность ошибки, а также за счет создания библиотек использовать их и для других приложений, т.е. тиражировать общие знания.

Общее правило начинается с утверждения *for any* <имя класса>.

for any new-pump

*if the power of the new-pump is off and the flow of the new-pump > 0.001
then conclude that the status of the new-pump is dripping and
change the body icon-color of the new-pump to red*

При поиске решения посредством прямого вывода общее правило возбуждается для каждого конкретного экземпляра класса new-pump. Допустимо использование в общих правилах вложенных конструкций *for*, а также конструкции *connected to* для учета наличия связей между объектами. Например:

for any control-valve C

for any storage-tank S connected to C

if the status of C is broken

then conclude that the status of S is shut-down

Для снятия неоднозначности используются локальные имена.

for any object O1 connected at an input of any object O2

if the flow of O1 is on

then conclude that the power of O2 is on

2.6.2. Утверждение every

Утверждение *every* используется для ссылки на каждый элемент класса в команде, например:

move every car upon this workspace by (25,0).

2.6.3. Порты

Когда к объекту подсоединено несколько объектов одного класса, ссылка типа “the <object> connected to <object>” двусмысленна. Связи могут

иметь направленность, которая может использоваться для снятия неоднозначности, например:

the pump connected at the input of tank-1
the pump connected at the output of tank-1

Если объект имеет несколько входных или выходных портов, то при определении объекта можно именовать порты. Указание имен портов также позволяет снять неоднозначность при записи утверждений:

the pump connected at the left-input-port of the tank.

Чтобы узнать имена портов, используется пункт меню *Describe*.

2.6.4. Общие процедуры

При записи **общих процедур**, как и общих правил, используется конструкция *for* для организации циклов. Пример общей процедуры приведен на рис. 2.10.

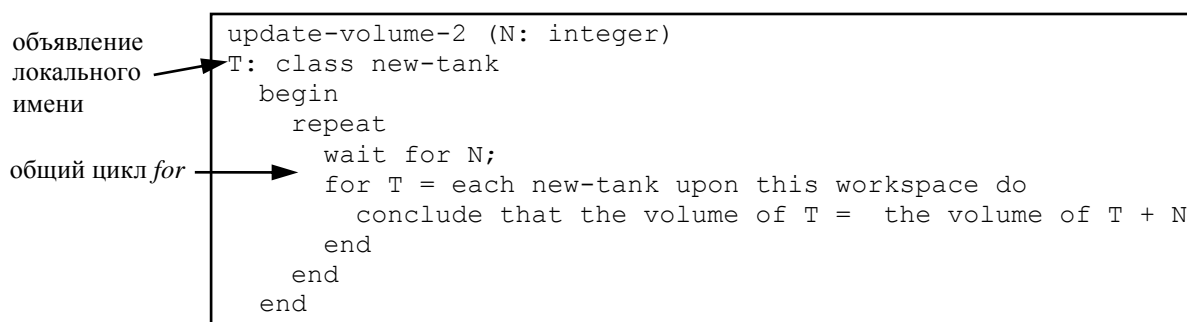


Рис. 2.10. Пример общей процедуры

Общий цикл *for* требует объявления локального имени.

Передача параметров в процедуру при ее вызове аналогична возбуждению общего правила для конкретного экземпляра. Для передачи каждого экземпляра класса используется конструкция *every*. Несколько реализаций процедуры могут выполняться одновременно (параллельно).

2.6.5. Проверка на существование

При ссылке на несуществующий элемент (сущность) генерируется сообщение об ошибке. Общие утверждения будут более надежными, если в них включать проверку на существование сущности перед ссылкой на нее.

if the station S connected at the output of station-45 exists
and the processing-time of S = 1
then change the body icon-color of S to yellow

2.6.6. Ссылки на арифметические конструкции и счетчик

В утверждениях можно ссылаться на сумму – *sum*, произведение – *product*, среднее – *average*, минимум – *minimum* и максимум – *maximum* по всем экземплярам – *over each*.

conclude that the volume of this-tank =
the volume of this tank - the sum over each new-pump
connected to this-tank of (the flow of the new-pump)

Также при записи утверждений можно использовать счетчики.

*conclude that the counter of tk =
the count of each tank upon this-workspace
{such that (the status of the tank is broken)}*

2.6.7. Классы объектов

Все классы существуют в рамках **иерархии классов**. Для отображения иерархии наследования класса используется пункт меню *Inspect* главного меню. Можно редактировать классы прямо из РП, созданного посредством команды *Inspect*. Атрибуты и пиктограммы наследуются по иерархии. Использование наследования позволяет реализовать более качественное приложение за счет:

- уменьшения избыточности и упрощение определения классов;
- использования общих высказываний в правилах, формулах (включая формулы моделирования) и процедурах;
- естественного способа описания мира (проблемной области).

При этом рекомендуется:

- определять правила и формулы как можно более общими;
- определять пиктограммы, связи и атрибуты как можно выше в иерархии;
- определять пиктограммы, связи и атрибуты, общие более чем для одного класса, в суперклассе;
- определять исключения и специальные случаи в подклассах.

Поскольку изменения в описании класса не всегда распространяется на его экземпляры, то рекомендуется использовать атрибут *change* для форсирования обновления экземпляров класса.

Для создания пиктограмм используется редактор пиктограмм. Пиктограмма может содержать один или более слоев. Каждый слой содержит один или более графических элементов одного и того же цвета. Слои могут быть сгруппированы в поименованные области (*regions*). Цвет любой поименованной области можно изменить вручную или динамически.

Некоторые полезные выражения

- any <класс>
- every <класс>
- the class of <сущность>
- <сущность> is a <класс>
- <сущность> is an instance of the class named by <символьное выражение>
- the symbol that is a superior-class of <символьное выражение> {это ссылка на имя класса-родителя для сущности}
- the symbol that is an inferior-class of <символьное выражение>
- for any symbol that is an inferior-class of <символьное выражение>

2.6.8. Классы связей

Подобно объектам, связи имеют описание класса. Для определения связей используется меню РП: *KB Workspace* → *New Definition* → *connection-definition*.

Рис. 2.11 иллюстрирует определение **класса связей**. Связям соответствуют поименованные области, цвета которых можно изменять динамически. Если имеется **разветвленная** связь, то используя пункт меню *Describe* для верхней связи, можно узнать имя соответствующего порта. Нельзя соединить связи разных классов и связи, направленные противоположно. Для предотвращения ошибок конфигурации можно использовать направление потоков и задание значения по-manual-connections атрибута *Capability and Restrictions* в описании объекта.

Точки связи используются для соединения связей из разных РП и являются классом объектов. Различные блоки соединяются посредством точек связи. Использование одинаковых имен для двух точек связи устанавливает связь между ними. Можно определить подкласс точек связи с уникальной пиктограммой, атрибутами и т.д.

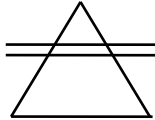
PIPE-WITH-BORDER, an connection-definition		 PIPE-WITH-BORDER
Notes	OK	
User restriction	none	
Class name	pipe-with-border	
Superior class	connection	← Начало иерархии
Attributes specific to class	width is 20	← Может иметь атрибуты подобно объекту
Capabilities and restrictions	none	
Class restrictions	none	
Change	none	
Inherited attributes	none	
Cross section pattern	2 red, 2 white, 2 red	← Вместо пиктограммы
Stub length	20	← Длина в пикселях для STUB
Junction block	junction-block-for-pipe-with-border	← Создается автоматически

Рис. 2.11. Пример определения класса связей

2.6.9. Рекомендации

1. Представляйте знания в общей форме для возможности повторного использования.
2. Уменьшайте избыточность в общих утверждениях и описаниях классов.
3. Создавайте общие утверждения как можно более надежными посредством проверки существования объектов в общих ссылках.
4. Не используйте конкретные значения в общих утверждениях, а храните их в атрибутах объектов, ссылаясь на эти атрибуты в общих утверждениях.
5. Ограничивайте область действия общих утверждений, когда это возможно.

2.7. Ограничения базы знаний

2.7.1. Возможности и синтаксис ограничений

G2 позволяет ограничивать следующие возможности:

- использование пунктов меню, включая пункты меню, определяемые пользователем;
- команды, выполняемые без использования меню, включая перемещения сущностей – *move*, редактирование выбранного текста – *click-to-edit* и т.д.;
- отображение атрибутов для различных групп пользователей;
- реакцию на выбор мышкой любой сущности (так могут создаваться новые элементы интерфейса);
- клавиатурные команды, доступные пользователю.

Допустима иерархия ограничений, что позволяет:

- задавать ограничения для РП в их таблицах;
- наследование каждой сущностью из этого РП заданных ограничений;
- перезаписывать (перекрывать) наследуемые ограничения.

Синтаксис ограничений включает следующие конструкции:

- ключевые слова: *include* и *exclude*, *when* и *unless*, *additionally* и *absolutely*;
- *when* устанавливает ограничения для группы пользователей, *unless* устанавливает ограничения для всех групп кроме указанной;
- *include* и *exclude* без *additionally* изменяют унаследованные ограничения, перекрывая их;
- *additionally* добавляет ограничения к наследуемым;
- *exclude absolutely* определяет ограничения, которые не могут быть перекрыты.

Системная таблица KB Restriction приведена на рис. 2.12. Ограничения, определенные в ней, применяются к каждой сущности в БЗ.

KB-RESTRICTIONS	
Notes	OK
User restrictions	unless in administrator mode: selecting any gensym-raised-button implies activate; when in presenter mode: menu choices for presentation-click include: nothing; non-menu choices for presentation-block exclude: move-object, click-to-edit; non-menu choices for free-text, borderless-free-text, number-message, presentation-message or presentation-message-no-background exclude: move-object, click-to-edit; selecting any presentation-bullet implies show-detail; selecting any free-text, borderless-free-text, number-message, presentation-message or presentation-message-no-background does nothing
Main menu user restrictions	none
Keyboard command restrictions	none
Initial G2 user mode for this kb	presenter

Рис. 2.12. Системная таблица KB Restriction

2.7.2. Изменение статуса пользователя

Для изменения статуса пользователя нужно вызвать пункт меню *Change Mode* в главном меню (заметим, что этот пункт меню может быть скрыт для определенных групп пользователей). Затем вводится имя группы пользователей в атрибуте *G2-user-mode*. Может потребоваться также имя пользователя (атрибут *User-name*) и пароль (атрибут *Password*).

2.7.3. Ограничение класса

В описании классов объектов, связей и сообщений можно ограничивать все экземпляры класса и все экземпляры всех его подклассов.

2.7.4. Пункты меню, определяемые пользователем

PUMP, an object	POWER-ON, a user-menu-choice
table	Notes OK
transfer	User restrictions None
name	Name POWER-ON
clone	Label power-on
rotate/reflect	Applicable class pump
delete	Condition the power of the item is off
disable	Action conclude that the power of the item is on
describe	Action priority 2
create subworkspace	
power-on	

2.8. Команда *Inspect*

Команда *Inspect* применяется для:

- ввода, отображения и редактирования вводимой команды;
- поиска сущностей (команда *go to*):
go to new-car,
go to rule-xxx-1,
go to message-board;
- перехода к РП для поиска и просмотра найденных сущностей, отображения иерархий объектов или сущностей, РП, модулей (команда *show on a workspace*):
show on a workspace every new-car containing the symbol RED,
show on a workspace every item with notes,
show on a workspace the class hierarchy of car;
- получения доступа/редактирования атрибутов сущности и для просмотра множества сущностей, удовлетворяющих заданным критериям (команда *display*):
display a table of every car,
display a table column wise of every car,
display a table of the class-name
and attribute-displays of every object-definition,
display a table column wise of every car such that the lot-time of the car > 10;

- организации подсветки заданной информации (команда *highlight*):
highlight the word car in every rule;
- редактирования (команда *replace*):
replace the word exit with exit in every borderless-free-text;
- создания простых отчетов по любым компонентам БЗ (команда *write*), например, создания файла отчета, содержащего таблицы атрибутов каждого правила:
write to the file report-1 every rule.

2.9. Параметры и истории

Отличие **параметров** от **простых атрибутов** заключается в следующем:

- параметры предусматривают проверку типов, а простые атрибуты – нет;
- параметры позволяют сохранять историю;
- параметры имеют начальные значения – *initial value* (это не то же самое, что значения по умолчанию – *default value*); простые атрибуты имеют значения по умолчанию, но не имеют начальных значений, начальные значения простым атрибутам нужно устанавливать принудительно в команде *conclude*;
- параметры должны всегда иметь значение; простые атрибуты могут не иметь значения, что может привести к ошибкам на стадии исполнения.

Для объявления атрибута параметром используется таблица атрибутов (рис. 2.13). Атрибут, который содержит параметр, получает значение из параметра. При вычислении, например, выражения “the volume of tank-x”, G2 получает значение параметра, содержащегося в атрибуте *volume*. Для ссылки на параметр в выражении можно сослаться на его атрибут-родитель или представить параметр в виде пиктограммы и сослаться на него как на любой другой объект.

OLD-TANK, an object definition	
Class name	old-tank
Superior class	tank
Attributes specific to class	volume is given by quantitative-parameter; level is steady.....
Capabilities and restrictions	none
Class restrictions	none
Change	maximum-volume is 30;
Inherited attributes	state is ok
Default settings	none
Attribute displays	inherited
Stubs	inherited
Color	inherited
Icon description	inherited

an old-tank	
Notes	OK
User restrictions	none
Names	none
Volume	0
Maximum volume	30
State	ok
Level	steady

Рис. 2.13. Пример объявления атрибута параметром

Рис. 2.14 иллюстрирует встроенные классы параметров. Для того чтобы посмотреть эту иерархию, используйте возможности команды *Inspect: show on a workspace the class hierarchy of parameter*.

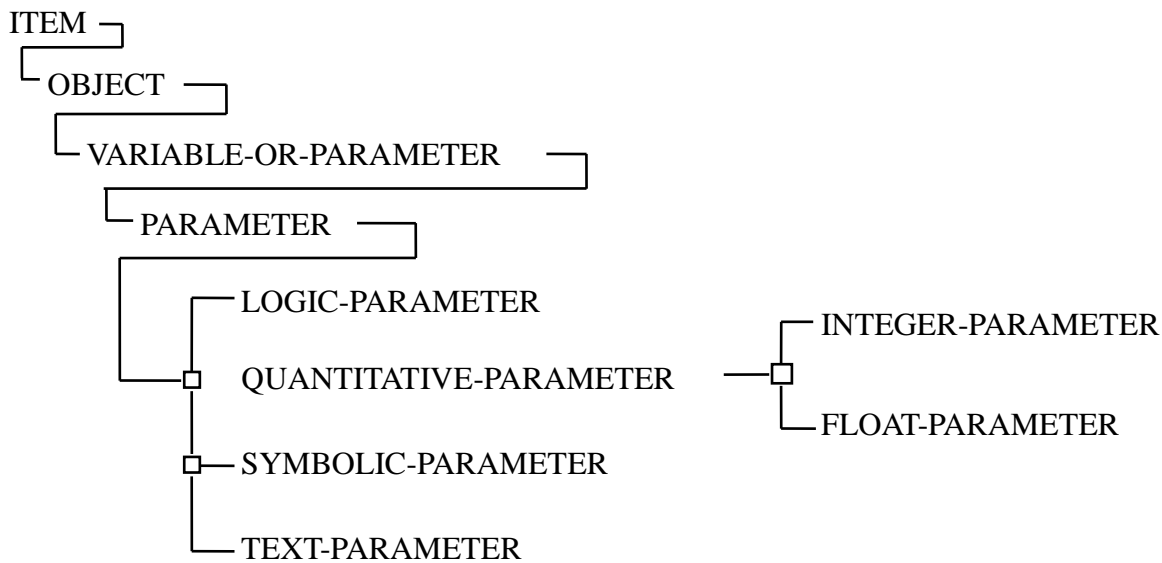


Рис. 2.14. Встроенные классы параметров

G2 позволяет представлять истории для сущностей и классов. Рис. 2.15 иллюстрирует пример представления истории.

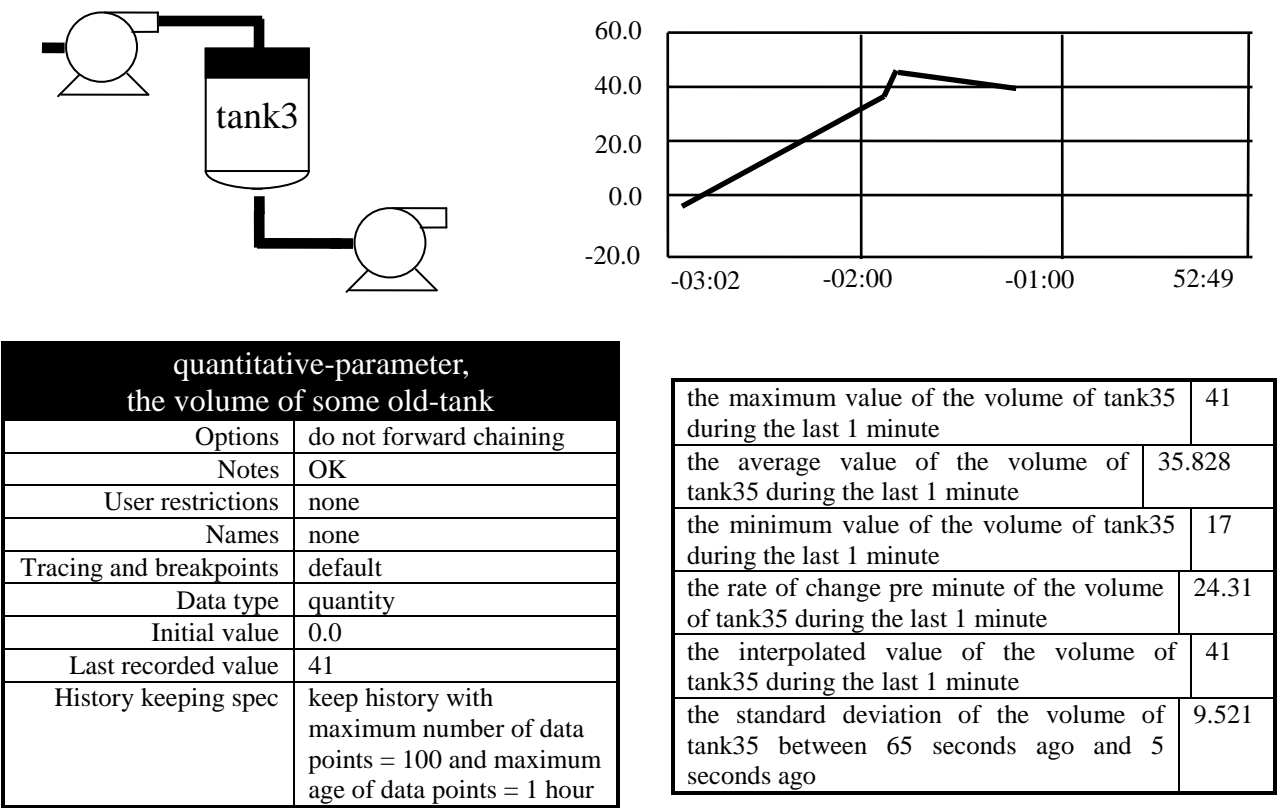


Рис. 2.15. Пример представления истории

2.10. Поиск данных и реальное время

2.10.1. Переменные, поиск данных, обратный вывод

Переменные аналогично параметрам имеют свою таблицу атрибутов (рис. 2.16). Основные отличия между переменными и параметрами заключаются в следующем:

- значение переменной имеет интервал корректности – *validity interval* (по истечении этого отрезка времени значение считается некорректным);
- переменные могут запрашивать и получать новые значения из источника данных – *data server*.

Переменные могут объединяться в классы. Иерархия встроенных классов переменных аналогична иерархии параметров. Можно определить собственный подкласс, используя атрибут *Default setting* для назначения интервала корректности по умолчанию (*default validity interval*) и др.

a float-variable, the flow of some new-pump	
Options	do not forward chain, breadth first backward chain
Notes	ok
User restrictions	none
Names	none
Tracing and breakpoints	default
Data type	float
Initial value	2.0
Last recorded value	2.0, expired 17 Nov 93 12:11:33 p.m.
History keeping spec	do not keep history
Validity interval	2 seconds
Formula	none
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	none

← Начальное значение (необязательное)
← Время «умирания»
← История
← “supplied” (вычисляется машиной вывода из интервалов корректности переменных, используемых для получения значения), “indefinite” или временной интервал
← Источник данных
← Интервал для поиска нового значения

Рис. 2.16. Пример таблицы атрибутов

Поиск данных инициируется в случае, когда какая-либо компонента G2 запрашивает значение для переменной, а последнее записанное значение перестало быть корректным. Рис. 2.17 иллюстрирует процесс поиска данных.

Атрибут *Timeout-for-variables* в системной таблице *Inference-Engine-Parameters* контролирует период времени, отведенный на получение значения, для всех переменных и составляет по умолчанию 30 секунд.

Процесс поиска данных заканчивается неудачей в следующих случаях:

- если переменная превысила этот период или нет источника, который может предоставить значение;
- обратный вывод и другие формы поиска данных завершаются неудачей;

При неудаче возбуждается правило *whenever* в форме *whenever <переменная> fails to receive a value then...* Пока запросы на получение значения остаются, G2 возбуждает поиск нового значения через интервал

повтора (*retry interval*). Атрибут *Retry-interval-after-timeout* устанавливается в системной таблице *Inference-Engine-Parameters* (1 минута по умолчанию).

Запрещение поиска данных:

- “the current value of <переменная или выражение>”, например: “the current value of the status of tank-1”, в этом случае G2 не осуществляет поиск данных для вычисления выражения;
- “<переменная или выражение> has a current value”, например: “the status of tank-1 has a current value” истинно, если G2 может получить значение без поиска данных, иначе – ложно;
- в атрибуте *Option* переменной введено “do not seek data”.

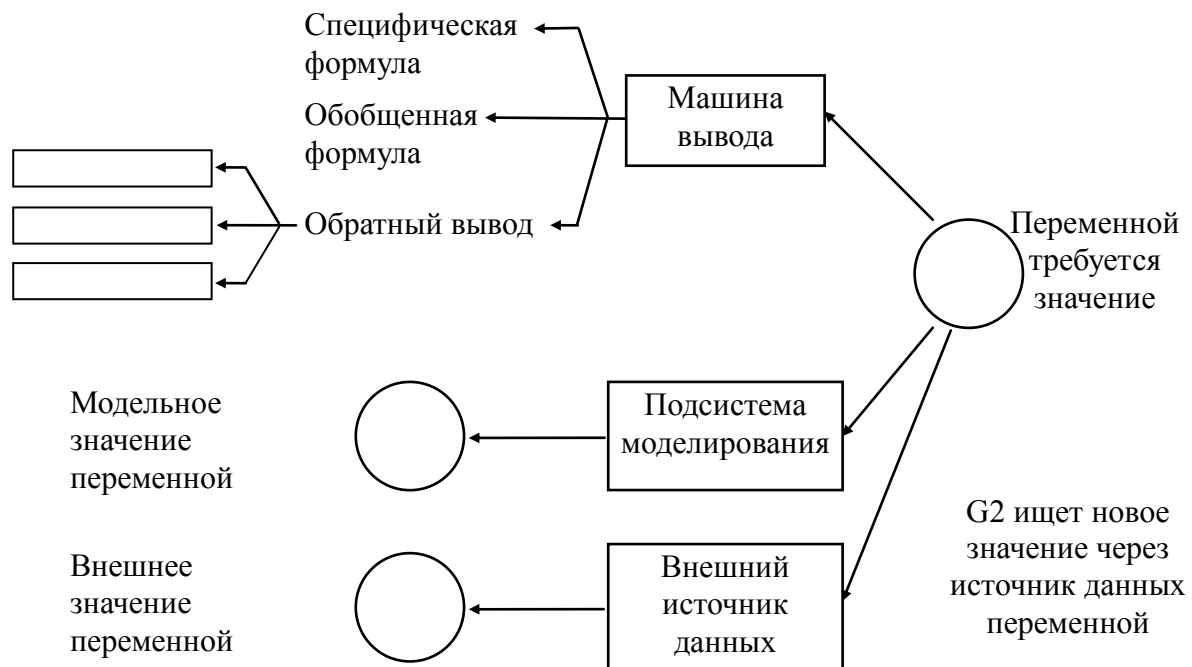


Рис. 2.17. Организация процесса поиска данных

G2 использует обратный вывод (*backward chaining*) как форму поиска данных. Поиск значения переменной осуществляется посредством правил, в которых делаются заключения о ее значении. Источником данных (*data server*) при этом является машина вывода (*inference engine*). G2 возбуждает все правила, которые делают заключение о значении требуемых переменных. Чтобы найти значение переменной, на которую ссылается данное правило, G2 может построить обратную цепочку вывода к другим правилам. Когда значение найдено, все другие запросы на поиск значения отменяются. По умолчанию G2 производит поиск в ширину, просмотр всех правил запускается одновременно. Как только любое правило находит значение, все другие правила сбрасываются.

Можно организовать обратный вывод в глубину для поиска для любой переменной (в атрибуте *Options* соответствующей переменной указать *depth-*

first backward chaining и задать наиболее правдоподобным правилам высший приоритет – меньшие числа из интервала от 1 до 10). G2 возбуждает правила в соответствии с их приоритетом. Последующее правило возбуждается только в случае, если предыдущее правило заканчивается неудачей или исчерпало время (*time-out*). Поиск в глубину требует меньших накладных расходов, чем поиск в ширину, но может продлиться дольше.

Отметим, что единственным основанием для запрещения обратного вывода может являться необходимость минимизации избыточной обработки. Для запрещения обратного вывода используется атрибут *Options* в правиле (для отмены использования этого правила в обратной цепочке вывода) или в переменной (для запрета обратного вывода при получении значения). Правило типа *when* по умолчанию не возбуждается в процессе прямого или обратного вывода.

Помимо рассмотренного способа получения значения переменными посредством поиска данных, возможно получение значения независимо от поиска данных и источника данных **для?** переменной:

- с помощью команды *conclude that* (forward chaining и др.);
- через пользовательский ввод (поля ввода и др.);
- через GSI;
- через атрибут *Initial value*.

Переменная также может получать значения посредством формул. При этом используется машина вывода в качестве источника данных. Формулы подразделяются на специфические и обобщенные. Специфическая формула определяется в атрибуте переменной *Formula*. Обобщенная формула определяется в РП подобно правилам: *KB-Workspace* → *New Definition* → *generic formula*.

При вычислениях специфические формулы перекрывают обобщенные, которые перекрывают еще более общие. Формула вычисляется только тогда, когда переменной требуется новое значение. Пример общей формулы:

*let the lot-time of any car =
the current month + 12 * (the current year - the year of the car)*

2.10.2. Конструкции, используемые при работе со временем

При работе в реальном времени могут быть полезны следующие синтаксические конструкции:

- the collection time of <переменная или параметр>
[as of <целое> data points ago];
- the expiration time of <переменная>;
- conclude that <переменная> {= | is} <выражение>
[with expiration <целочисленное выражение>]
[with collection time <целочисленное выражение>].

2.10.3. Процедуры в контексте реального времени

В процедурах допустимы непосредственные ссылки на простые атрибуты и параметры. Непосредственная ссылка на переменную невозможна. Чтобы использовать в процедуре значение переменной, необходимо передать значение в процедуру или использовать оператор *collect data* для получения значения. Возможна передача переменной в процедуру, но любая ссылка на переменную ссылается на то значение, которое имела переменная при ее передаче. Интервал корректности (*validity interval*) внутри процедур не проверяется.

Для получения значения из внешнего окружения процедуры можно использовать оператор *collect data*. Полученное значение не имеет интервала корректности. Оператор *collect data* должен завершиться перед выполнением следующего оператора, что может привести к ожиданию значения. Можно посредством инструкции *timeout* включить предельное время ожидания и указать операторы, выполняемые при истечении предельного времени ожидания. Можно также осуществить сбор данных для нескольких локальных переменных. Например:

```
data-environment ()
T5, T6: float;
begin
  collect data (timing out after 5 seconds)
    T5 = the temperature of room-5;
    T6 = the temperature of room-6;
  if timeout then goto last-line end;
  inform this workspace on this workspace below room-5
    for the next 60 seconds that "[T5]";
last-line: end
```

Для организации параллельной обработки служит оператор *do in parallel*. Ниже приведены примеры процедур с использованием этого оператора и организации циклов на его основе.

```
race()
begin
  do in parallel
    call race-car(racer-1);
    call race-car(racer-2);
    call race-car(racer-3);
  end
end

race-until()
begin
  do in parallel until one completes
    call race-car(racer-1);
    call race-car(racer-2);
  end

  call race-car(racer-3);
end

call race-car(racer-3);
end

race-car(C: class car)
speed: integer = random(1,5);
begin
  repeat
    move C by (speed,0);
    exit if the item-x-position
      of C >= the item-x-position of
      the finish-line nearest to C;
    allow other processing
  end
end
```

В случае аварийного завершения процедуры для обработки ошибки можно использовать оператор *on error*. Данный оператор возвращает имя и текст ошибки (для большинства ошибок возвращается имя "error"). Необходимо объявить локальные имена: *error-name* (символьное) – для

имени ошибки; *error-text* (текстовое) – для текста ошибки. Можно вкладывать блоки *begin-end* внутри тела процедуры и включать оператор *on error* после каждого блока. Пример использования оператора *on-error*:

```
example-error-handler()
error-name: symbol;
error-text: text;
begin
  call nobody();
  begin call none() end
  on error(error-name, error-text) end
end
on error(error-name, error-text)
  inform the operator that "[error-text]" end
```

Для сброса процедуры и всех ее работающих копий или только конкретных копий служит команда *abort*. Чтобы сослаться на конкретную копию, необходимо изменить значение атрибута *Class-of-procedure-invocation* на *procedure-invocation* у процедуры. По умолчанию его значение *none*, что более эффективно.

2.11. Представление динамических знаний

2.11.1. Динамические сущности

Любую сущность можно создать и именовать динамически. Объекты, созданные динамически, по умолчанию являются временными (*transient*). На рис. 2.18 приведены примеры создания объектов динамически, объявления их постоянными и временными. Временные объекты также можно динамически удалить (командой *Delete*). Чтобы удалить постоянный объект динамически, сначала необходимо сделать его временным. G2 также поддерживает динамически созданные связи.

Transfer	<table> <tr> <th colspan="2">an action-button</th></tr> <tr> <td>Label</td><td>"Transfer"</td></tr> <tr> <td>Action</td><td>in order create a bottle B and transfer B to this workspace at (the item-x-position of the bottle- station upon this workspace - 25, the item-y-position of the bottle-station upon this workspace - 25)</td></tr> </table>	an action-button		Label	"Transfer"	Action	in order create a bottle B and transfer B to this workspace at (the item-x-position of the bottle- station upon this workspace - 25, the item-y-position of the bottle-station upon this workspace - 25)
an action-button							
Label	"Transfer"						
Action	in order create a bottle B and transfer B to this workspace at (the item-x-position of the bottle- station upon this workspace - 25, the item-y-position of the bottle-station upon this workspace - 25)						
Make Permanent	<table> <tr> <th colspan="2">an action-button</th></tr> <tr> <td>Label</td><td>"Make Permanent"</td></tr> <tr> <td>Action</td><td>make every bottle upon this workspace permanent</td></tr> </table>	an action-button		Label	"Make Permanent"	Action	make every bottle upon this workspace permanent
an action-button							
Label	"Make Permanent"						
Action	make every bottle upon this workspace permanent						
Make Transient	<table> <tr> <th colspan="2">an action-button</th></tr> <tr> <td>Label</td><td>"Make Transient"</td></tr> <tr> <td>Action</td><td>make every bottle upon this workspace transient</td></tr> </table>	an action-button		Label	"Make Transient"	Action	make every bottle upon this workspace transient
an action-button							
Label	"Make Transient"						
Action	make every bottle upon this workspace transient						

Рис. 2.18. Пример объявления объектов постоянными и временными

2.11.2. Списки

Списки G2 имеют иерархическую структуру (см. рис. 2.19). Могут быть организованы списки значений и списки сущностей. Классы списков могут быть определены пользователем. Пример организации списка сущностей:

in order

create a bottle B and

transfer B to this workspace and

insert B at the beginning of the item-list bottle-list-9

Для отображения иерархии списков необходимо войти в режим инспектирования базы данных и ввести команду *show on a workspace the class hierarchy of g2-list*.

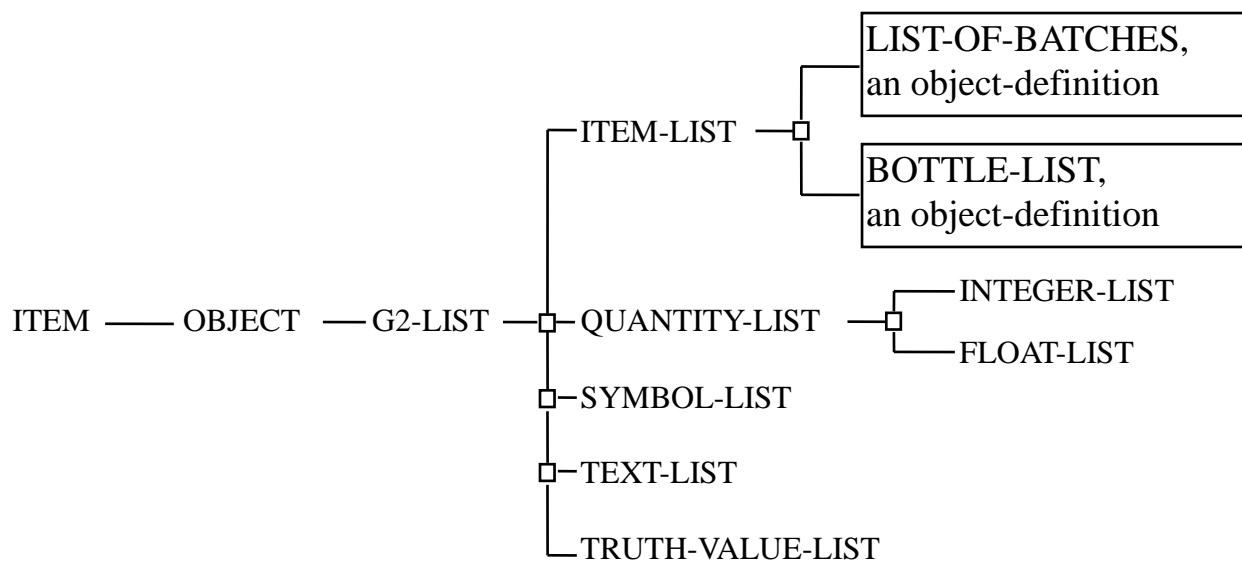


Рис. 2.19. Иерархия списков G2

2.11.3. Массивы

Для отображения иерархии массивов необходимо войти в режим инспектирования базы данных и ввести команду *show on a workspace the class hierarchy of g2-array*.

Могут быть организованы массивы значений и массивы сущностей. G2 позволяет организовать многомерные массивы и определенные пользователем классы массивов (рис. 2.20).

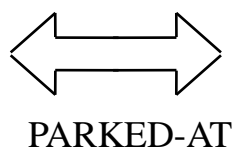
CAR-ARRAY, an object definition	
Notes	OK
User restrictions	none
Class name	car-array
Superior class	item-array
Attributes specific to class	none
Capabilities and restrictions	none
Class restrictions	none
Change	none
Menu option	a final menu choice

Inherited attributes	element type for item-array: car; initial values for item-array: car-1, car-2, car-3; array-length for g2-array:3
Default settings	none
Attribute displays	inherited
Stubs	inherited
Color	inherited
Icon description	inherited

Рис. 2.20. Пример организации определенного пользователем класса массивов

2.11.4. Отношения

Отношение определяется посредством таблицы атрибутов (рис. 2.21). Отношения можно устанавливать, используя команду *conclude*. Установление отношения не должно противоречить его описанию и инициирует прямой вывод. Установленные отношения являются временными, и для их просмотра используется пункт меню *Describe*.



PARKED-AT, a relation	
Notes	OK
User restrictions	none
First class	car
Second class	parking-space
Relation name	parked-at
Inverse of relation	occupied-by
Type of relation	one-to-one
Relation is symmetric?	no
a car may be parked-at most one parking space; a parking-space may be occupied-by at most one car	

Рис. 2.21. Пример определения отношения

Ниже приведены примеры, иллюстрирующие ссылки на сущности, связанные отношением, а также завершение (*break*) отношений.

change the body icon-color of the car that is parked-at space2 to yellow

change the body icon-color of the parking-space that is occupied-by blue-bomber2 to green

if the car that is parked-at space2 exists then inform the operator that “Стойнка 2 занята”

*if not (the car that is parked-at space2 exists)
then inform the operator that “Стойнка 2 свободна”*

*if there exists a parking-space PS such that (not (the car that is parked-at PS exists))
then inform the operator that “Стойнка [the name of PS] свободна”*

*if for every parking-space (the car that is parked-at the parking-space exists)
then inform the operator that “Все места парковки заняты”*

conclude that car3 is now parked-at space4

conclude that car3 is not parked-at space4

conclude that car3 is not parked-at every parking-space

*for any car C whenever C ceases to be parked-at any parking-space PS
then conclude that the status of PS is unoccupied
and move C to (the item-x-position of PS + 50, the item-y-position of PS + 50)*

Отношения могут быть использованы для ограничения области действия общих утверждений. Например: *for any bottle that is part-of batch-1 if the bottle is full then ...*

2.11.5. Сообщения

Сообщения являются встроенным классом сущностей. Их можно создавать вручную, используя меню РП: *KB-Workspace → New Free Text → Message*. Динамически можно выполнять следующие действия над сообщениями: создать – *create*, удалить – *delete*, передать – *transfer* и переместить – *move*. Можно изменять динамически текст, цвет фона – *background-color*, рамки – *border-color* и текста – *text-color*. Для сообщений можно определять отношения и вносить сообщения в списки и массивы. Сообщения помещаются в Message Board и Logbook.

Пользователь может определить класс сообщений, используя меню РП: *KB-Workspace → New-Definition → message-definition*. Message – корневой класс в иерархии классов сообщений. Можно определить свойства по умолчанию (*Default-Message-Properties*) для класса сообщений: цвет текста – *text-color*; цвет рамки – *border-color*; минимальная ширина – *minimum-width*; минимальная высота – *minimum-height*; шрифт – *font*.

При подготовке текста сообщения используется операция конкатенации (*concatenate*) и текстовые функции: перевод в верхний регистр первых букв слов – *capitalize-words*; извлечение из текста – *get-from-text*; вставка в текст – *insert-in-text*; поиск подстроки – *is-contained-in-text*; длина текста – *length-of-text*; перевод в нижний регистр – *lower-case-text*; пропустить в тексте – *omit-from-text*; позиция в тексте – *position-of-text*; число в начале текста – *text-begins-with-quantity*; перевод в верхний регистр – *upper-case-text*.

Пример использования операции конкатенации:

*conclude that t1 = "checked"
conclude that t2 = "g2"
conclude that t3 = "I [t1] [t2] on [the current day of the week]
[the current real time as a time stamp]."*

2.12. Пользовательский интерфейс

2.12.1. Кнопки и слайдеры

Кнопки пользовательского интерфейса подразделяются на командные (*Action Buttons*), альтернативные (*Radio Buttons*), селективные (*Check Boxes*) и кнопки, определяемые пользователем (*User-Defined Buttons*). Пример определения командной кнопки приведен на рис. 2.22.

an action-button	
Notes	OK
User restriction	none
Names	none
Label	"Show Action Button Workspaces"
Action	show the subworkspace of red-workspace with its bottom left corner 5 units to the right of and 5 units above the bottom left corner of the screen and show the subworkspace of white-workspace with its bottom left corner 5 units to the right of and 5 units above the bottom left corner of the screen
Action priority	2

Рис. 2.22. Пример определения командной кнопки

Для более наглядного представления информации используются поля ввода (*Type-in Boxes*) и слайдеры (*Sliders*) (рис. 2.23).

Введите новое значение newer-num: Переместите маркер для ввода числа:

 NEWER-NUM


-66  99
18

Рис. 2.23. Пример использования полей ввода и слайдеров

2.12.2. Дисплей, диаграммы, графики, таблицы

В G2 имеется развитый набор средств пользовательского интерфейса для представления **дисплеев** (*Readout Tables*), **диаграмм** (*Charts*) (простых линейных, столбчатых и точечных), **графиков** и **таблиц**. Для их определения применяются соответствующие таблицы.

2.13. Подсистема моделирования

2.13.1. Назначение и основные характеристики

Подсистема моделирования G2 предназначена для выполнения следующих функций:

- имитации входных сигналов перед запуском приложения on-line, используется во время разработки и тестирования;
- проверки точности датчиков путем сравнения их показаний с модельными значениями;
- использования модельных значений вместо реальных при сбое датчика;
- оценка состояний, которые не могут быть получены путем наблюдений.

Подсистема моделирования полностью интегрирована с G2 и позволяет:

- решать алгебраические, разностные и дифференциальные уравнения N-го порядка;
- определять специфические и общие формулы и задавать различный временной шаг для разных переменных;
- строить процедуры моделирования и имитационные модели.

Основные характеристики подсистемы моделирования:

- подсистема моделирования состоит из отдельных программных фрагментов;
- можно рассматривать подсистему моделирования как источник данных для переменных и параметров G2;
- значения предназначены только для переменных и параметров;
- результирующие значения могут быть числовыми, истинностными, символьными и текстовыми;
- в формулах моделирования могут использоваться простые атрибуты, параметры и переменные, которые в свою очередь тоже могут моделироваться.

2.13.2. Взаимосвязь с другими компонентами G2

1. Модельные и регулярные значения переменных:

- переменные могут использовать подсистему моделирования в качестве источника данных;
- подсистема моделирования вычисляет очередные значения через заданный интервал времени, даже если переменная не использует модель в качестве источника данных;
- переменная может иметь как регулярное (*regular*), так и модельное (*simulated*) значения;
- ссылка на модельное значение: *the simulated value of ...*
- самое последнее значение, полученное на основе модели, не обязательно совпадает с текущим значением переменной, которое использует G2.

2. Взаимосвязь с переменными:

- переменные могут запрашивать значения у подсистемы моделирования, подсистема моделирования предоставляет последние модельные значения;
- можно установить модельные значения с помощью команды *set*;
- чтобы установленное с помощью *set* значение повлияло на последующие, переменная должна иметь начальное значение.

Рис. 2.24 иллюстрирует структуру связи подсистемы моделирования с переменными.

3. Взаимосвязь с параметрами и простыми атрибутами:

- формулы моделирования могут использовать параметры, вне зависимости от того, моделируются они или нет;
- параметры не вызывают поиск значений (не имеют источника данных), однако, если для параметра определена формула моделирования, модель устанавливает новое значение параметра через заданный временной интервал;
- для параметров можно определить обобщенную формулу моделирования и нельзя – специфическую;
- формулы моделирования могут использовать простые атрибуты;

- простые атрибуты не могут моделироваться.

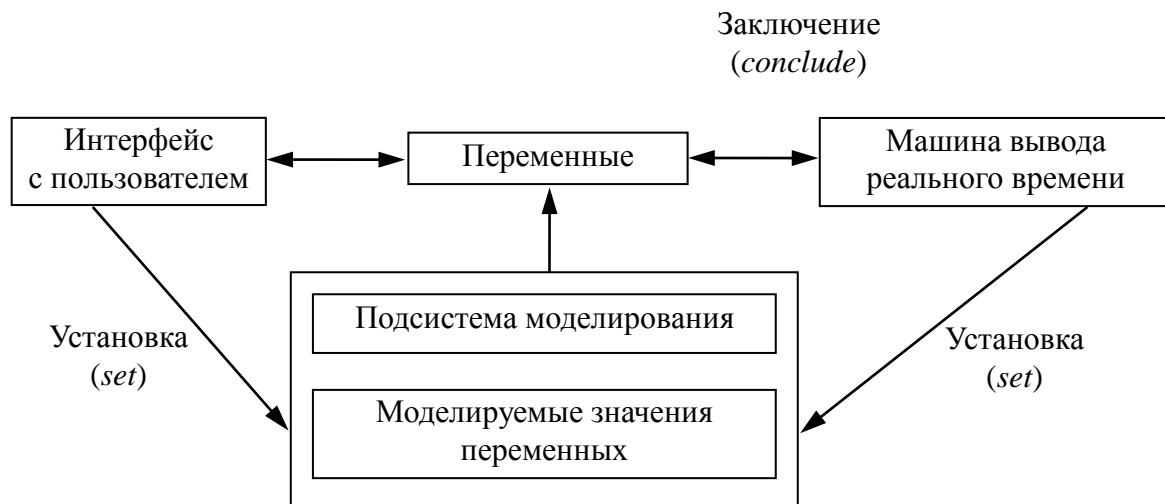


Рис. 2.24. Связь подсистемы моделирования с переменными

Рис. 2.25 иллюстрирует структуру связи подсистемы моделирования с параметрами (структура связи с простыми атрибутами аналогична).

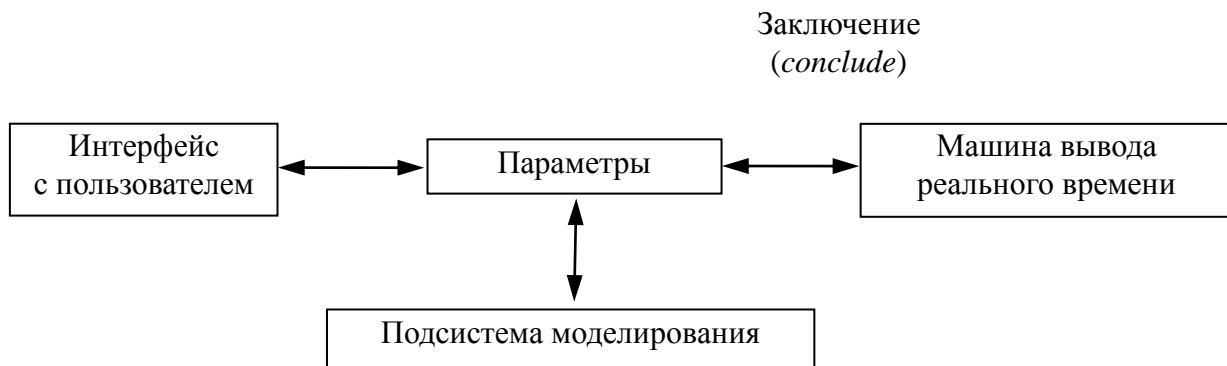


Рис. 2.25. Связь подсистемы моделирования с параметрами

2.13.3. Формулы моделирования

Формулы моделирования могут иметь вид алгебраических, разностных или дифференциальных уравнений N-го порядка. Формулы подразделяются на специфические и общие. Специфические формулы:

- описывают поведение конкретной переменной;
- не применимы к параметрам и простым атрибутам;
- более приоритетны, чем общие формулы;
- вводятся в подтаблице *simulation subtable* переменной (левая часть уравнения опускается).

Пример определения специфической формулы приведен на рис. 2.26.

Общие формулы описывают поведение класса объектов и определяются посредством меню РП: *KB-Workspace* → *New Definition* → *Generic Simulation Formula*.

AIRCRAFT-2, an aircraft	
Notes	OK
Name	AIRCRAFT-2
Altitude	340.0

simulation subtable for the altitude of AIRCRAFT-2	
Notes	OK
Names	none
Tracing and breakpoints	default
Time increment for update	none
Simulation formula	340
History keeping spec	do not keep history

a float-variable, the altitude of AIRCRAFT-2	
Options	do not forward chain, breadth first backward chain
Notes	OK
Name	nine
Tracing and breakpoints	default
Data type	float
Initial value	none
Last recorded value	340.0, expires 5 Jul 95 3:15:35 p.m.
History keeping spec	do not keep history
Validity interval	2 seconds
Formula	none
Simulation details	dependent, and has specific subtable
Initial value for simulation	default
Data server	G2 simulator
Default update interval	none

Рис. 2.26. Определение специфической формулы

Алгебраические уравнения применяются для моделирования значений переменных, являющихся функцией от текущих значений других моделируемых переменных:

*the flow of any valve V = the position of V *
sqrt(the pressure0in of V/the pressure-out of V)*

Разностные уравнения применяются для переменных, являющихся функцией от собственных предыдущих значений, и требуют начальных условий:

*state variable: next value of the principal of any loan L = the principal of L -
(the principal of L * the interest of L), with initial value 100*

Дифференциальные уравнения решаются на основе метода Эйлера или Рунге-Куты и также требуют начальных условий:

*state variable: d/dt (the volume of any flow-tank) = the inflow
of the flow-tank - the outflow of the flow-tank, with initial value 300*

Специфические формулы приоритетнее (перекрывают) общих формул, а более специфические – менее специфических.

В общих формулах могут устанавливаться начальные значения переменных, которые могут перекрываться путем установки начальных значений в атрибуте *Initial value for simulation* в таблице конкретной переменной.

Частота вычисления новых значений переменных определяется временным шагом, который устанавливается одним из следующих способов:

- редактирование атрибута *Time Increment for Update* в подтаблице *simulation subtable* для переменной;
- указание временного шага в общей формуле моделирования;
- редактирование атрибута *Default Simulation Time* в системной таблице *Simulation Parameters*.

Возможные ошибки моделирования делятся на ошибки инициализации и ошибки времени исполнения. Ошибки инициализации

возможны при наличии конфликтующих общих формул (при использовании первой введенной формулы) и при задании начальных условий, когда в этом нет необходимости. В случае ошибок времени исполнения моделирование продолжается до тех пор, пока это возможно, но процесс моделирования может не завершиться. Сообщения об ошибках заносятся в журнал оператора (*operator logbook*).

2.14. Планировщик

В связи с тем, что приложение G2 управляет множеством одновременно возникающих задач, необходим **планировщик**. Планировщик управляет всеми процессами в G2. Хотя пользователь никогда не взаимодействует с ним, планировщик контролирует как всю активность, видимую пользователем, так и активность фоновых задач. Планировщик определяет порядок обработки задач, взаимодействует с источниками данных и пользователями, запускает процессы и осуществляет коммуникацию с другими процессами в сети. На рис. 2.27 изображена блок-схема работы планировщика G2.

Планировщик циклически выполняет следующую **последовательность** шагов.

1. Проверка наступления начала цикла. Если начало цикла наступило, планировщик начинает цикл и переходит к следующему шагу.
2. Планирование ожидающих задач. Планировщик формирует список задач, которые будут выполняться на данном цикле. Этот список называется очередью текущих задач.
3. Обслуживание источников данных. Планировщик обменивается данными с каждым источником данных. Каждому из источников данных отводится не более 0,1 с на выполнение этой операции. Для источников, не закончивших обмен за выделенное время, планируются задачи для попытки закончить передачу данных.
4. Выполнение задач. Планировщик пытается выполнить как можно большее число задач из очереди задач. Любая из задач, не закончившаяся в течение 0,2 с, откладывается для выполнения в конце данного цикла или в следующем цикле.
5. Обработка сетевых пакетов. Планировщик посылает и получает сообщения через сеть. На это выделяется до 0,2 с.
6. Обслуживание пользователей. Планировщик принимает и передает данные всех пользователей, работающих в данном сеансе G2, включая пользователей Telewindows.
7. Подготовка к следующему циклу. Планировщик проверяет, осталась ли какая-либо активность в рамках данного цикла (получение данных, завершение отложенных задач и т.д.). Если да, он возвращается к шагу 1 и проверяет, не наступило ли время нового цикла. Если да, планировщик переходит на следующий цикл. Если нет, он переходит к

шагу 3 для завершения всех отложенных действий. Если отложенных задач не осталось и время нового цикла не наступило, планировщик «засыпает» на 40 мс и после этого переходит к шагу 1.

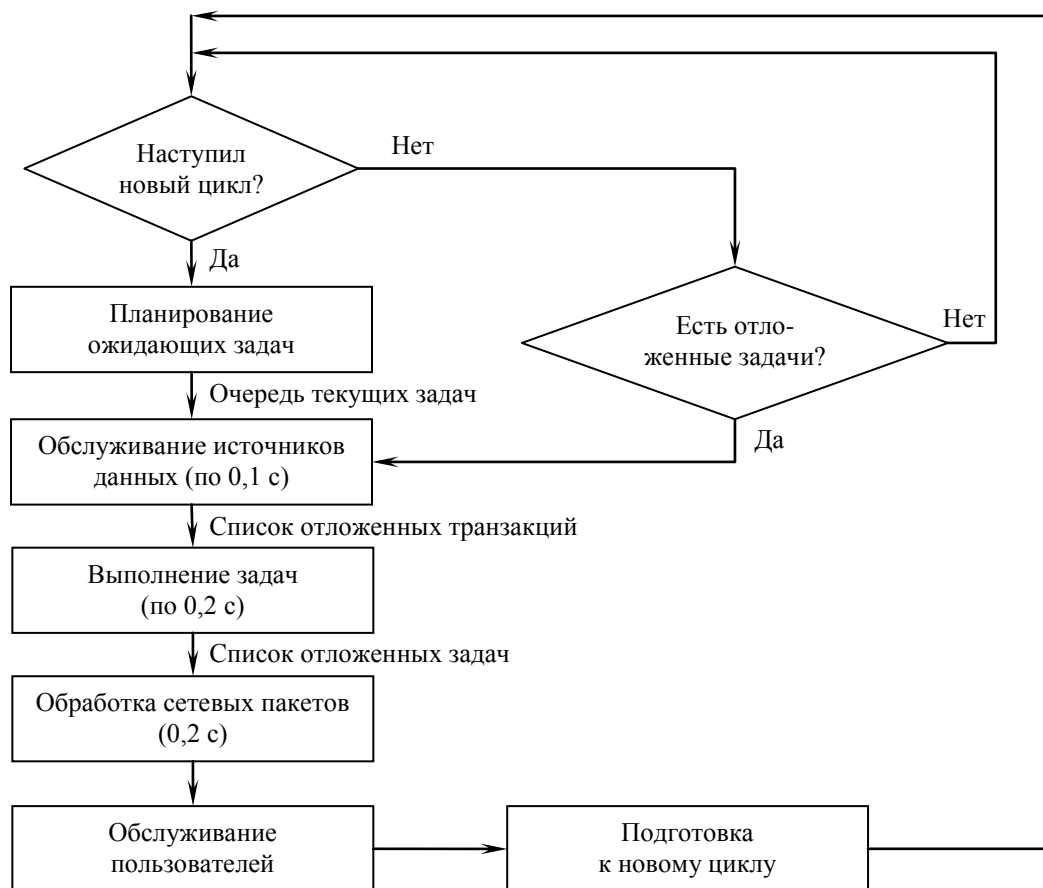


Рис. 2.27. Блок схема работы планировщика G2

2.15. Контрольные вопросы к разделу 2

1. Какие типы сущностей **имеются в G2**?
2. Какие типы правил существуют в G2? Каковы их особенности и назначение?
3. Что такое прямой и обратный вывод?
4. Какие способы возбуждения правил существуют в G2?
5. Как записываются общие правила в G2? Зачем они нужны?
6. В чем состоят различия между атрибутами, параметрами и переменными?
7. Для чего нужна подсистема моделирования в G2?
8. Для чего нужен планировщик в G2?

3. ПОРЯДОК СОЗДАНИЯ МОДЕЛИ В СРЕДЕ G2

3.1. Запуск G2

Для работы с G2 необходимо запустить **сервер** и **клиента**:

1. Для запуска сервера выберите в главном меню операционной системы MS Windows в папке «Gensym G2» ярлык «Start G2 Server» и обязательно дождитесь появления информации о версии G2 и имени хоста и TCP/IP порта. Например, на рис. 3.1 имя хоста PM-laboratory, а порт 1111. Эта же информация отобразится в заголовке окна серверного приложения.

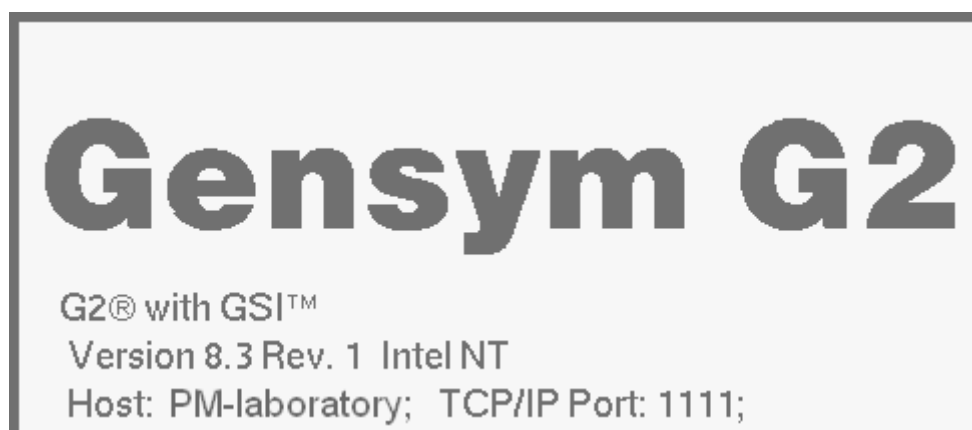


Рис. 3.1. Сервер готов к работе

2. Если клиент будет работать на том же компьютере (**и порт 1111**), то просто запускаем клиента из того же меню, выбирая соседний ярлык «Telewindows Next Generation», если же клиент (один или несколько) надо запустить на соседнем компьютере локальной сети, или сервером использован другой порт, то необходимо создать копию этого ярлыка и указать в его свойствах еще два параметра в командной строке помимо имени вызываемого файла – это имя хоста и порта. Например,

D:\Gensym\g2-8.3r1\g2\twng.exe PM-laboratory 1111

С одним и тем же сервером могут работать одновременно несколько клиентов, при этом они будут работать с одной базой знаний, что удобно при коллективной разработке.

Максимальное количество серверов запущенных на одном компьютере и максимальное количество клиентов, подключающихся к одному серверу, ограничены типом лицензии.

3.2. Создание рабочего пространства

Для создания базы знаний необходимо создать хотя бы одно рабочее пространство, поскольку все остальные объекты базы также имеют графическое представление, и их можно создавать и размещать только на каком-либо пространстве.

1. В основном меню выберите пункт *Workspace* → *New Workspace*, либо пункт *New Workspace* в контекстном меню рабочей области окна Telewindows (рис. 3.2).
2. Назовем пространство, чтобы впоследствии к нему можно было обращаться по имени (через соседний пункт меню *Get Workspace*). Для этого в таблице атрибутов пространства, **вызываемого** из контекстного меню этого пространства, отредактируйте атрибут *Names*. Для редактирования значения этого атрибута вызовите редактор двойным щелчком левой кнопки мыши справа от имени нужного атрибута. Например, дадим окну имя ГЛАВНОЕ-ОКНО (рис. 3.3), и закроем редактор для сохранения. Кнопка закрытия *End* появится только после ввода пробела в конце имени.



Рис. 3.2. Создание нового рабочего пространства

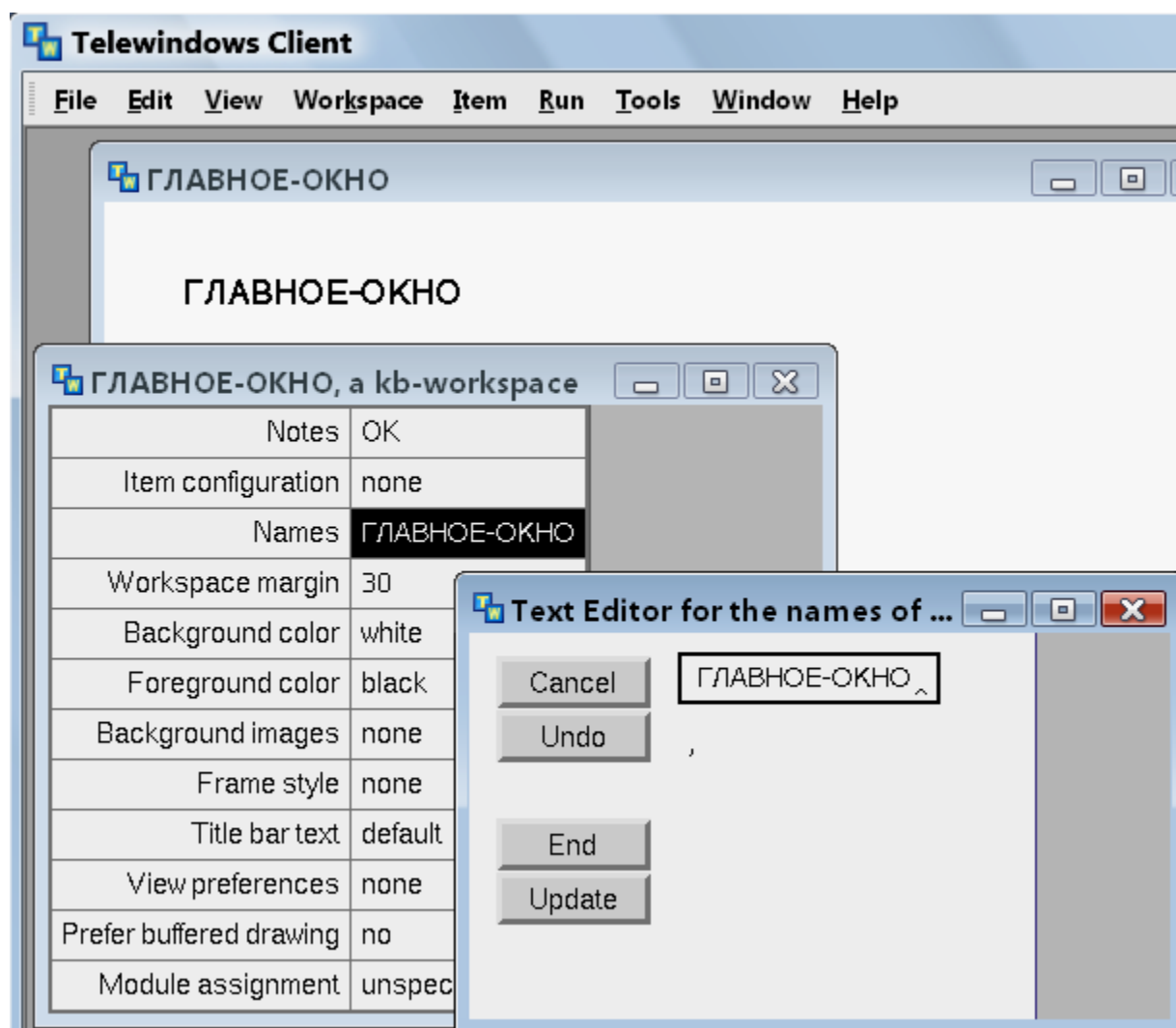


Рис. 3.3. Редактирование имени рабочего пространства

3. Сохраните базу знаний с помощью меню *File* → *Save KB...* Имя желательно задавать без использования символов кириллицы и пробелов, а только латинскими буквами, дополняя цифрами и разделяя тире с подчеркиваниями. При использовании кириллицы возможны проблемы с открытием базы знаний.

3.3. Создание иерархии рабочих пространств

Для более удобного размещения объектов базы знаний поместим объекты разных типов (правила, процедуры, классы, методы, меню) на отдельных рабочих пространствах и создадим удобную навигацию для перемещения между ними. Для создания навигации воспользуемся стандартным модулем GFR, расположенным в папке G2. Для использования его возможностей надо подключить этот модуль к своему модулю.

3.3.1. Порядок подключения модуля (убрать «.»)

1. **Задайте** паузу, если база запущена. Для этого в основном меню выберите *Run* → *Pause*.
2. Выполните слияние текущей базы, в которой должно быть хотя бы одно рабочее пространство, с базой знаний GFR: в основном меню выберите *File* → *Merge KB...* и укажите базу gfr.kb, лежащую в папке G2 *C:\Program Files\Gensym\g2-8.3r1\g2\kbs\utils*. Предупреждение: если при слиянии обнаружатся противоречия с текущей базой знаний, например, по именам процедур, часть возможностей базы окажется заблокированной. Следите за сообщениями в окне логов (*Operator Logbooks*) справа. Для предупреждения таких ошибок рекомендуется к названиям всех объектов модуля всегда добавлять специфичный префикс, например, для объектов модуля uilroot.kb, который подключится вместе с модулем gfr.kb, таким префиксом будет «uil-».
3. Далее надо сообщить о необходимости включения этого модуля в постоянный состав текущей базы (*Edit* → *System Tables* → *Module Information*). При правильном заполнении появившейся таблицы (рис. 3.4), все замечания (*Notes*) будут устранены (рис. 3.5).
4. Если текущему модулю еще не дали имя, то его надо придумать (желательно без использования кириллицы, т.к. это будет имя базы) и вписать в качестве значения атрибута *Top-level-module*. Назовем его, например, grv-main. Для устранения замечаний (1) и (3) из таблицы на рис. 3.4, необходимо будет в таблицах атрибутов всех рабочих пространств своей базы (в частности у пространства ГЛАВНОЕ-ОКНО) в качестве значения атрибута *Module-assignment* прописать имя модуля, к которому они приписаны – grv-main.

5. Для включения модуля GFR в состав необходимых укажите его имя в поле *Directly-required-modules*. Это устранил замечания (2) и (4) с рис. 3.4. Обратите внимание, что редактор предлагает на выбор список доступных модулей, выберите gfr и закройте редактор появившейся кнопкой *End*.
6. Посмотрите для проверки иерархию модулей, набрав в Инспекторе (*Tools → Inspect...*) команду *show on a workspace the module hierarchy*. На отобразившейся диаграмме модуль gfr должен быть подчинен вашему основному модулю (рис. 3.6), а ему в свою очередь подчинены еще два модуля.

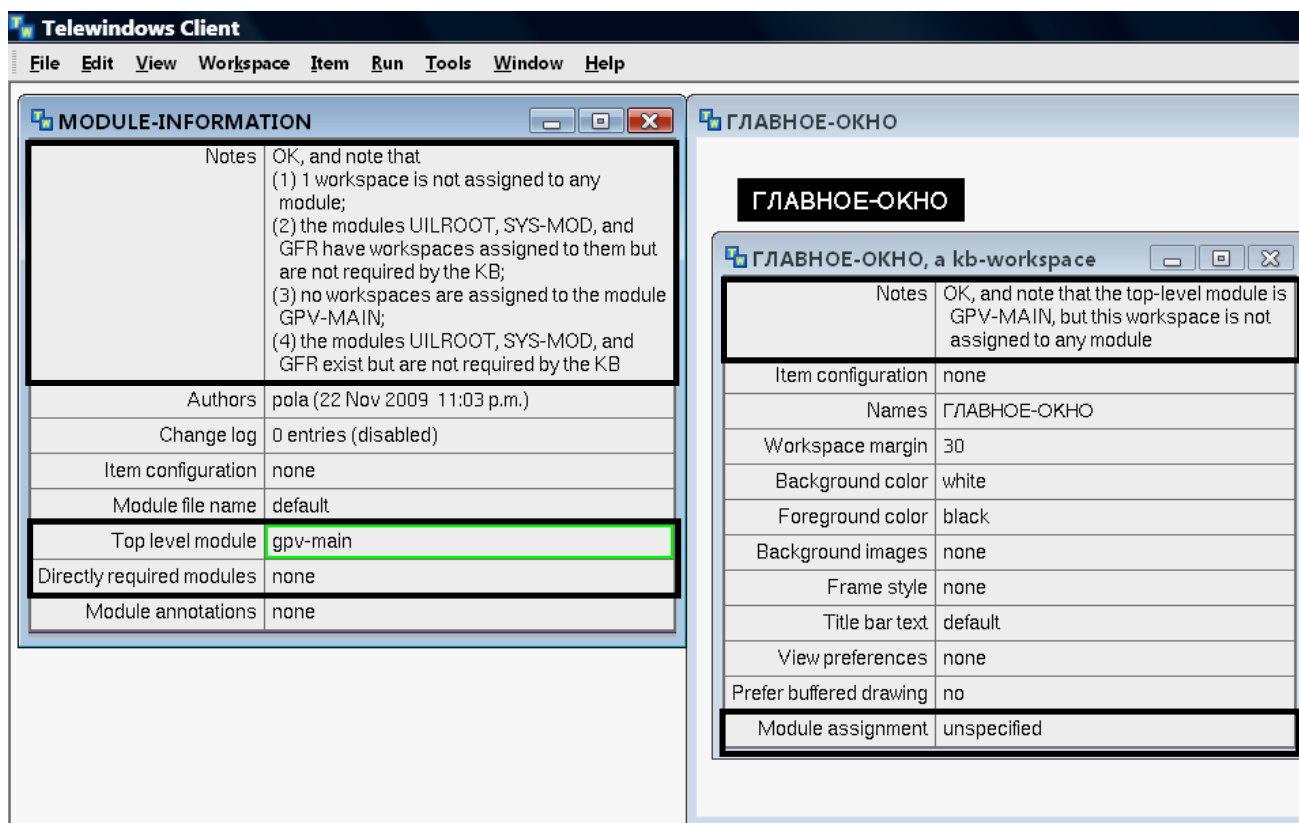


Рис. 3.4. Информация о модуле и таблица атрибутов главного окна

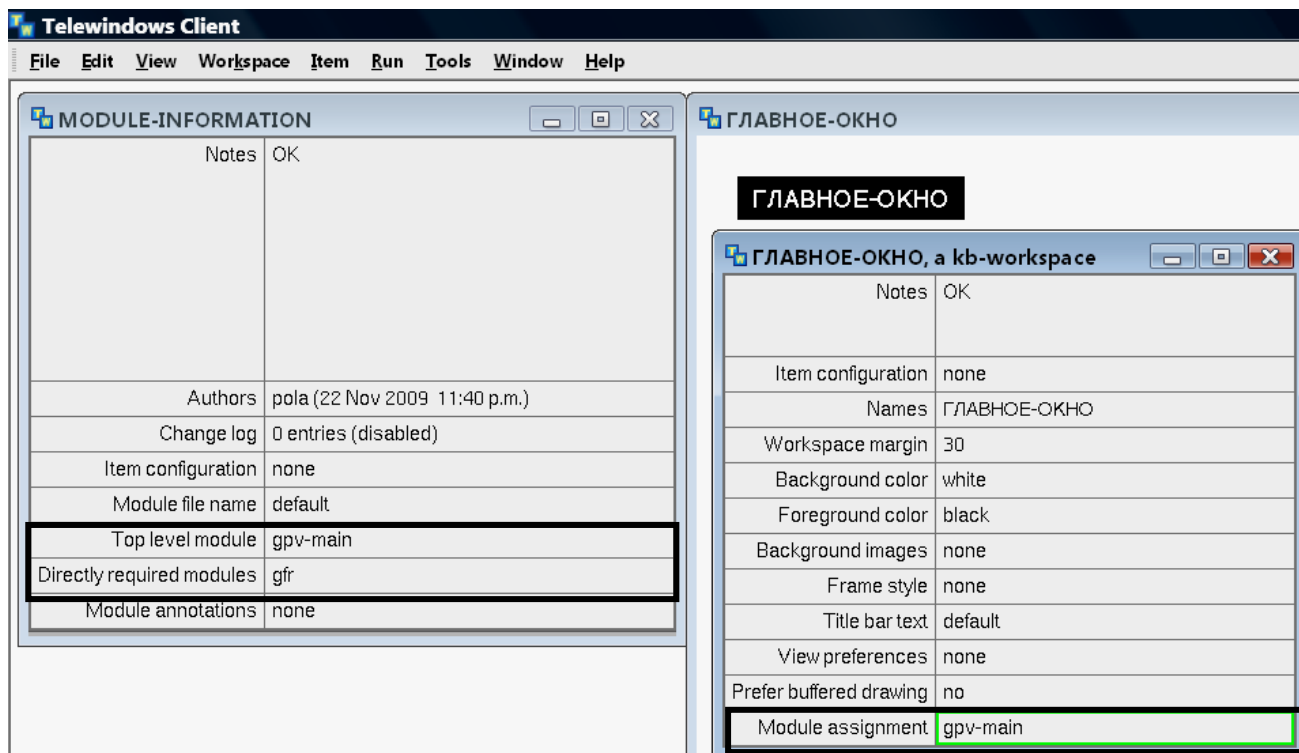


Рис. 3.5. Результат редактирования – замечания устранены

7. Сохраните базу знаний (*File → Save KB...*). Если было изменено имя главного модуля, при сохранении будет предложено сохранить базу под этим именем (в данном примере – «gprv-main.kb»), не препятствуйте.

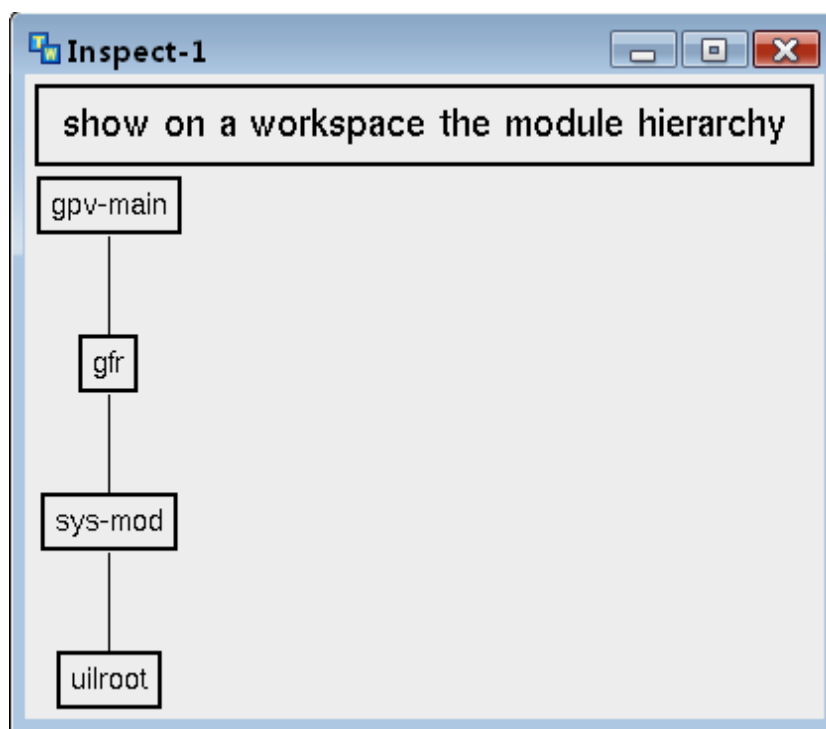


Рис. 3.6. Иерархия модулей

3.3.2. Создание кнопок навигации и изменение режимов доступа

1. Закройте все лишние окна с помощью кнопок с крестиком в правом верхнем углу окон, оставив только ГЛАВНОЕ-ОКНО. Если его тоже случайно закрыли, открыть его можно с помощью меню *Workspace* → *Get Workspace*. Таким образом, можно перейти к любому именованному рабочему пространству. После выполнения слияния в вашу базу были добавлены окна из подключенных модулей, их имена тоже видны в списке имен, доступных через указанный пункт меню.
2. Перед созданием кнопок навигации приостановите работу базы (*Run* → *Pause*) и перейдите в режим разработчика (*developer*) с помощью меню *Tools* → *Change Mode* (рис. 3.7). При создании нового модуля вы изначально находитесь в роли администратора (*administrator*), но можно поменять режим и даже ограничить права доступа и полномочия для разных групп пользователей базы в целях ее безопасности и разделения обязанностей. В данном случае кнопки перехода между пространствами будут по-разному работать в режимах разработчика и администратора: для разработчика по умолчанию кнопки срабатывают на перемещение, а для администратора – на редактирование, а для перемещения необходимо выбрать из контекстного меню команду *Select*. Разработчик может только создавать и использовать данные кнопки, администратору же доступно еще и изменение атрибутов, перемещение и удаление данных кнопок.

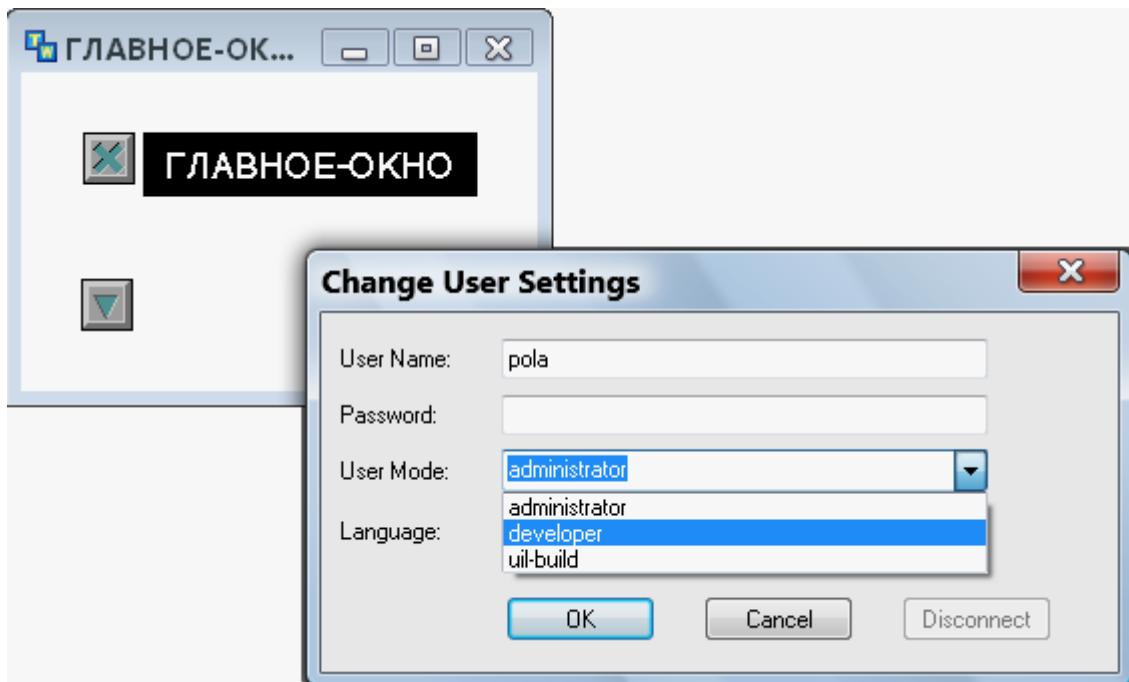








Рис. 3.7. Смена режима

3. Для создания необходимой кнопки воспользуйтесь контекстным меню рабочего пространства *New Object* → *uil-navigation-button-template*. Для навигации предусмотрены следующие командные кнопки (рис. 3.8):

- Перейти к подпространству 
- Перейти на уровень вверх 
- Перейти к следующему рабочему пространству 
- Перейти к предыдущему рабочему пространству 
- Скрыть текущее пространство 
- Вызов справки (пространства со справочной информацией) 

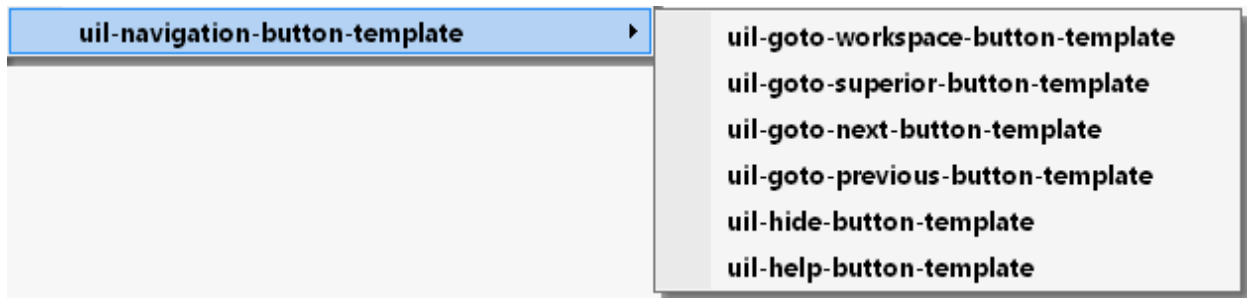


Рис. 3.8. Классы кнопок навигации

4. Создайте кнопку сокрытия (hide button) главного окна слева от названия пространства и кнопку перехода к подпространству (go to workspace) чуть ниже (рис. 3.9). Для того, чтобы увидеть как они работают, убедитесь, что база работает (*Run* → *Start* или *Run* → *Resume*) и, вы находитесь в режиме разработчика (*developer*). Если вы закрыли главное окно, воспользуйтесь меню *Workspace* → *Get Workspace*.
5. Создайте в главном окне еще одно кнопку для перехода к подпространству. Откройте оба подпространства, дайте им имена и создайте на них кнопки для сокрытия и перехода на уровень вверх (рис. 3.9). В режиме администратора можно подписать кнопки переходов, изменив значение атрибута *Label* в таблице атрибутов нужной кнопки.
6. Перейдите в режим разработчика, чтобы проверить работу всех созданных кнопок. Для удаления или перемещения кнопок перейдите в режим администратора. Для обрезки неиспользуемых областей пространств воспользуйтесь командой *Shrink Wrap* из контекстного меню нужного рабочего пространства.

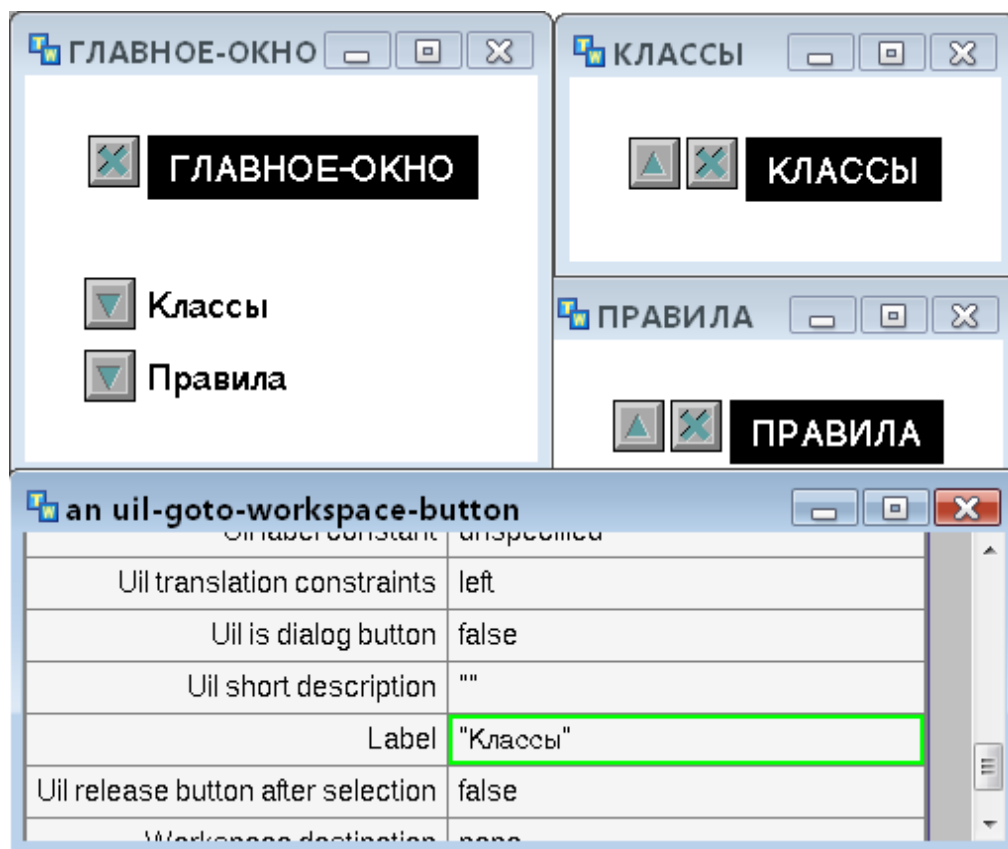


Рис. 3.9. Создание иерархии рабочих пространств

Если необходим переход к уже существующему пространству, создайте кнопку перехода к подпространству и измените в его таблице атрибутов значение атрибута *Workspace-destination* на имя существующего рабочего пространства.

3.3.3. Командные кнопки

Для того чтобы быстрее переключаться между режимами администратора и разработчика можно создать две командные кнопки в главном окне с помощью контекстного меню *New Button → action-button*. У этих кнопок надо заполнить как минимум 2 свойства в таблице: *Label* и *Action* (рис. 3.10).

В качестве действия введите фразы, пользуясь подсказками редактора:

conclude that the g2-user-mode of this window = the symbol administrator и
conclude that the g2-user-mode of this window = the symbol developer

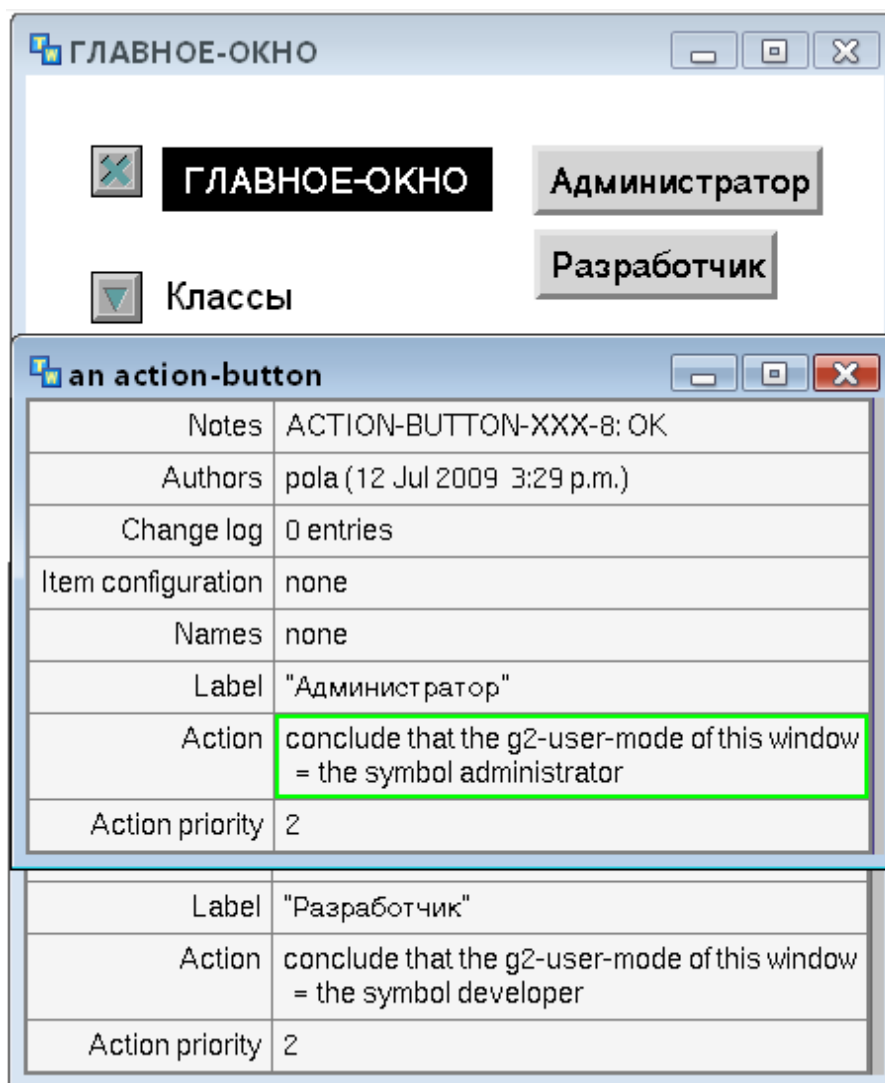


Рис. 3.10. Создание командных кнопок для смены режима

Убедитесь, что база запущена, замечаний в верхней строке таблиц атрибутов кнопок нет (Notes: Ok), и проверьте работу кнопок.

Используя кнопки, создайте в главном окне еще две кнопки для перехода к подпространствам «Процедуры» и «Моделирование», назовите эти подпространства, и создайте кнопки для возвращения в главное окно и скрытия пространств.

Для просмотра получившейся иерархии пространств используйте Инспектор (Inspect) и команду *show on a workspace the workspace hierarchy of главное-окно*.

3.4. Создание классов и объектов

3.4.1. Создание классов

Класс, базовое понятие объектно-ориентированной технологии, – основа представления знаний в G2. Данный подход уменьшает избыточность и упрощает определение классов на основе родительских классов (суперклассов), позволяет использовать общие правила для объектов одного класса, процедуры, формулы, уменьшая их число.

Все, что хранится в базе знаний и с чем оперирует система, является объектом того или иного класса, даже базовые типы данных – символьные, числовые, булевы. Классом самого верхнего уровня является класс с именем *item*.

Перейдем в рабочее пространство «Классы» и создадим класс «человечек», который будет **стоять, сидеть** на стуле и ходить:

1. Создайте новый класс с помощью контекстного меню пространства, выбрав пункт *New Definition* → *class-definition* → *class-definition*. Расположите его под именем рабочего пространства.
2. Заполнить необходимо как минимум 2 атрибута из таблицы созданного класса – имя класса (*Class name*) и имена родительских классов (*Direct superior classes*), **т.е.** возможно множественное наследование. Также обычно заполняются список специфичных атрибутов класса (*Class specific attributes*) и иконка (*Icon description*). Иконка используется для придания специфичного вида объектов данного класса. Создадим класс «Человечек», заполнив атрибуты в соответствии с рис. 3.11.
 - Имя нового класса – «человечек»;
 - Имя непосредственного суперкласса – «object»;
 - Специфичный атрибут человека «состояние» перечисляемого типа и может иметь 3 значения: «стоит», «сидит» и «идет»; по умолчанию при создании объекта этого класса данному атрибуту будет присвоено значение «стоит»;
 - Второй атрибут – «шаг» целого типа, и по умолчанию получает значение 5;
 - Третий атрибут – «движется» булевого типа, показывает, находится ли в данный момент человек в движении или нет, начальное значение – ложь (*false*).
3. Значение атрибута *Class inheritance path* сформируется автоматически. Для просмотра иерархии классов в графическом виде используйте **Инспектор** (*Inspect*), набрав команду *show on a workspace the class hierarchy of человек* (рис. 3.12).
4. Иконку можно создавать как в текстовом режиме, так и с помощью встроенного графического редактора. Выбор первого пункта (рис. 3.13) контекстного меню (*edit icon*) приведет к вызову графического редактора, а второго (*edit*) – текстового. Графическое изображение может состоять из нескольких регионов-слоев, имеющих собственное имя и цвет (рис. 3.14). Например, иконка человека состоит из 4 регионов: «стоя», «шагом», «сидя» и безымянного, получившего имя по имени использованного синего (*blue*) цвета. С помощью изменения цвета регионов можно будет менять изображение на иконке, чтобы оно соответствовало «состоянию» человека (рис. 3.15). Изображение для иконки также может быть загружено и из внешнего файла (*jpg* или *gif*

формата), если его предварительно загрузить в G2, создав описание файла с помощью *New Definition* → *image-definition*.

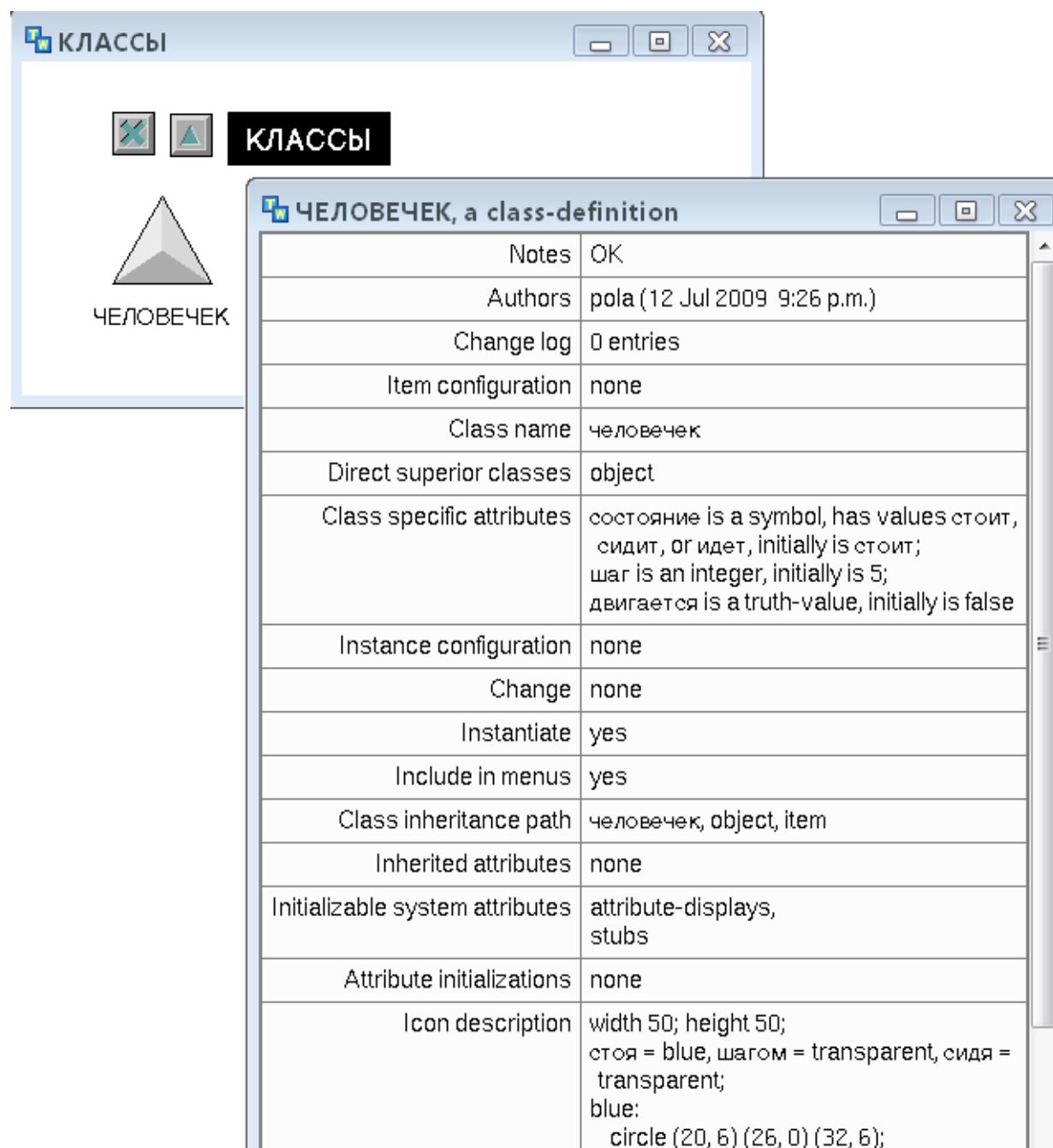


Рис. 3.11 Создание описания класса

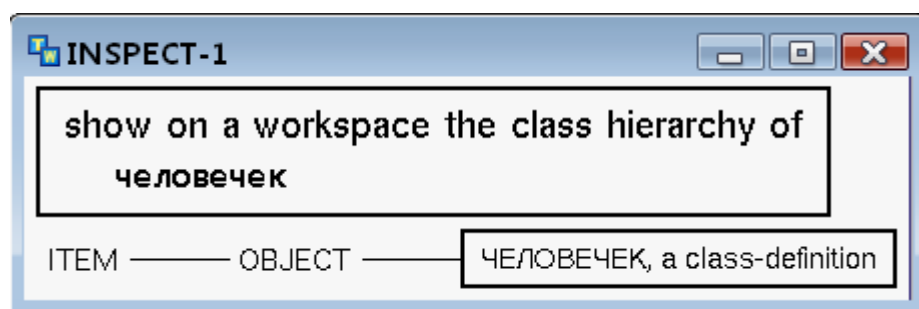


Рис. 3.12. Положение класса в иерархии

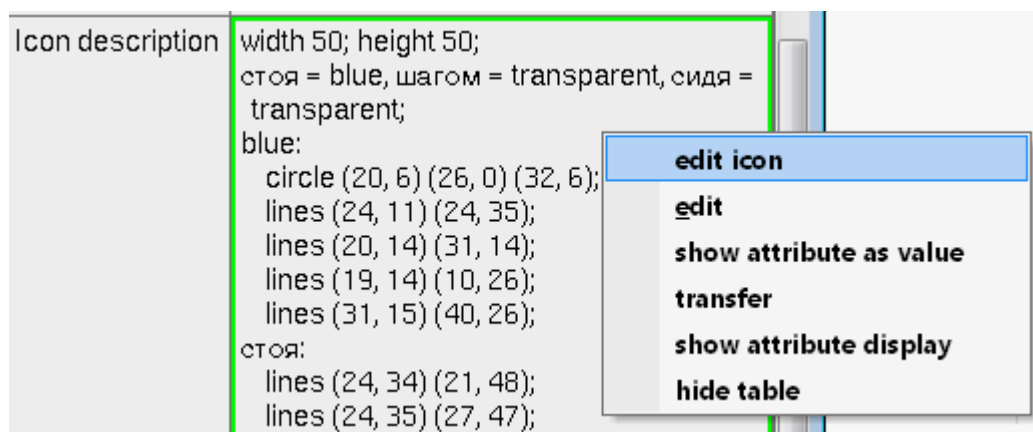


Рис. 3.13. Выбор редактора иконки

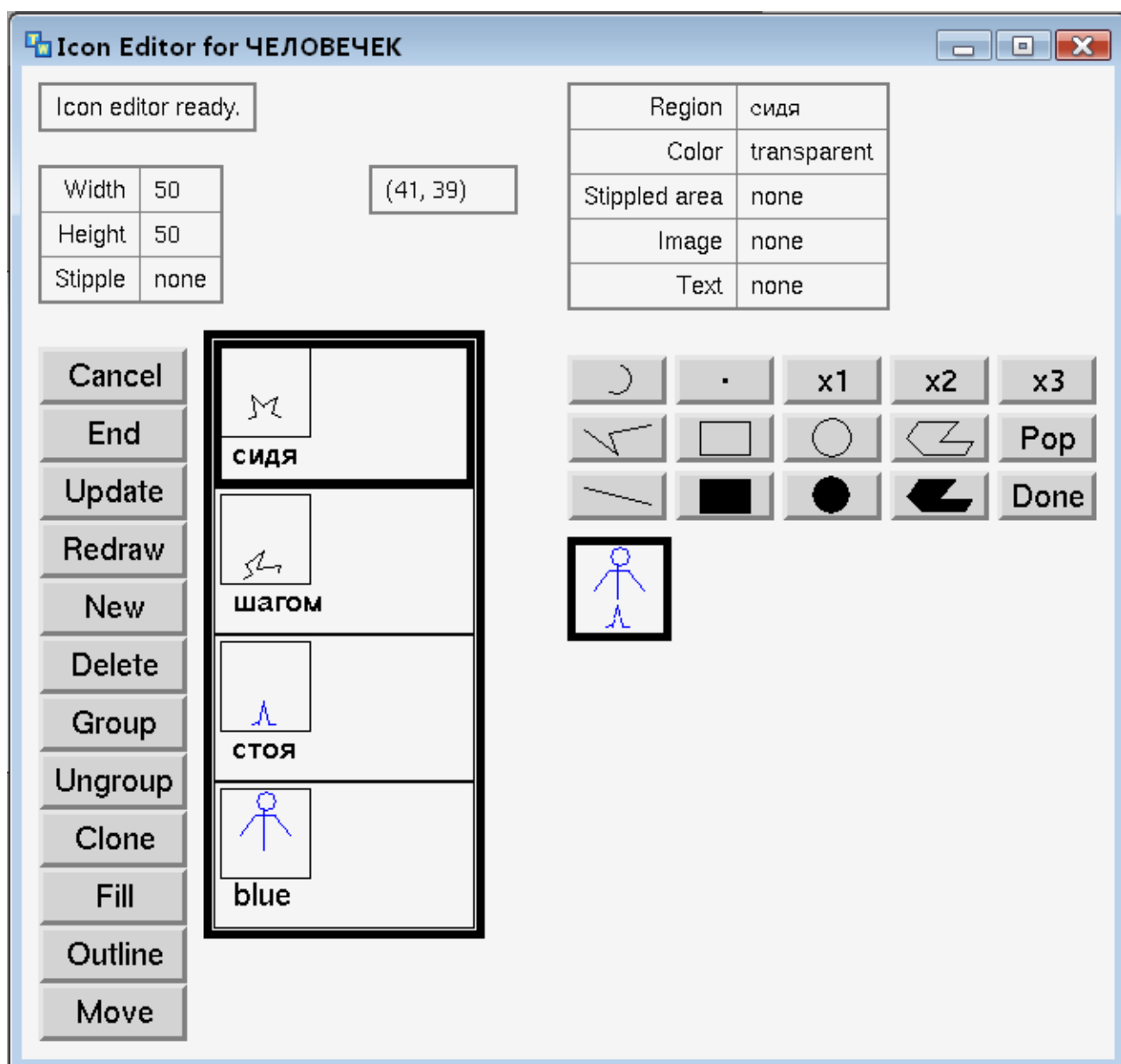


Рис. 3.14. Графический редактор



Рис. 3.15. Разные состояния человечка и стула

5. Создадим аналогичным образом еще один класс – «стул» (рис. 3.16).
 - Имя класса – «стул»;
 - Имя непосредственного суперкласса – «object»;
 - Специфичный атрибут – «состояние» со значениями «занят» и «свободен», последнее используется по умолчанию при создании нового стула;

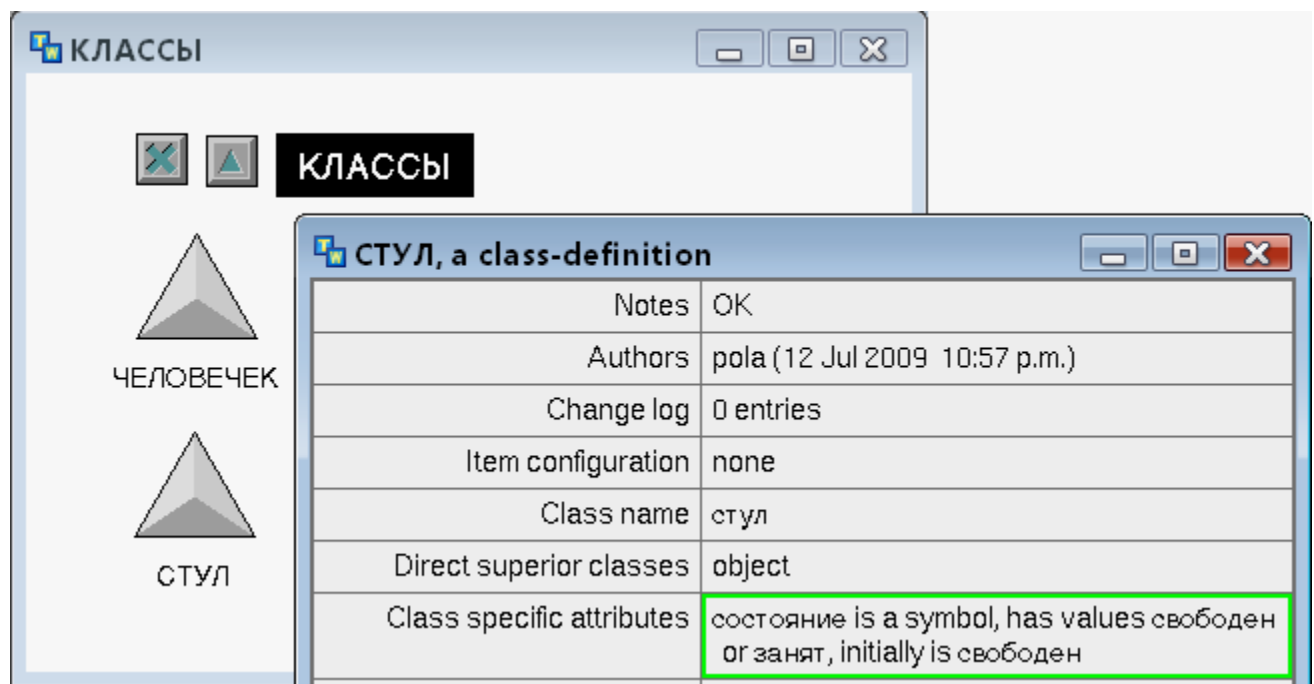


Рис. 3.16. Описание класса «стул»

Создадим также *отношение (relation)* для установления невидимой связи между человечком и стулом, на котором он сидит:

1. Создайте на пространстве «Классы» с помощью контекстного меню рабочего пространства New Definition → relation отношение «сидеть-на».
2. Заполните значения следующих атрибутов:
 - Первый класс (*First class*) – человечек;
 - Второй класс (*Second class*) – стул;
 - Имя отношения (*Relation name*) – сидеть-на;

- Имя обратного отношения (*Inverse of relation*) – быть-занятым;
- Тип отношения (*Type of relation*) – one-to-one.

В результате любой *человечек* может *сидеть-на* максимум одном *стуле*, а любой *стул* может *быть-занятым* максимум одним *человечком*.

3.4.2. Создание объектов

Создадим на рабочем пространстве «Моделирование» несколько объектов ранее описанных классов «человечек» и «стул».

1 способ. Перейдите в пространство «Классы», вызовите контекстное меню для класса «человечек», выберите пункт *Create Instance* и укажите место расположения человека на нужном пространстве.

2 способ. В контекстном меню пространства «Моделирование» выберите пункт *New object* → *человечек* и укажите место, где расположить человека.

Около созданного объекта можно вывести значения его атрибутов. Для этого откройте таблицу нужного объекта, для нужного атрибута вызовите контекстное меню и выберите пункт *Show attribute display*. Появившееся значение можно переместить в подходящее место около объекта.

Объекты можно клонировать (*clone*) и перемещать (*transfer*) на другие пространства.

3.4.3. Создание палитры

Для удобства создания новых человечков и стульев создадим палитру, вновь используя подключенный модуль GFR. В качестве палитры воспользуемся главным окном своего модуля.

1. Откройте рабочее пространство верхнего уровня модуля GFR с помощью верхнего меню *Workspace* → *Get Workspace* → *GFR-TOP-LEVEL*.
2. На открывшемся пространстве установите флажок *palette preparation tools*.
3. Поместите на ГЛАВНОЕ-ОКНО своего модуля по одному экземпляру человека и стула, а затем выберите в контекстном меню каждого пункт *add palette behavior*, появившееся после установки флажка.
4. Снимите флажок на GFR-TOP-LEVEL и опробуйте палитру. Если база запущена, и вы находитесь не в режиме администратора, то при каждом щелчке на стуле и человеке, появляется новый экземпляр, который можно поместить на другое рабочее пространство своего модуля кроме палитры.

3.4.4. Методы классов

Методы наследуются классами-потомками, но могут и изменяться, реализуя возможность полиморфизма, тогда для вызова **унаследованного метода** из тела нового метода используется команда *call next method*.

Первым параметром **метода** обязательно должен быть параметр, имеющий тип класса, для которого метод написан.

Создадим несколько простых методов для созданных классов:

1. Создайте в рабочем пространстве классов ссылки на новые рабочие пространства **Базовые-методы-человечка** и **Базовые-методы-стула**, куда поместим методы для чтения и записи значений всех атрибутов без проверки корректности данных (рис. 3.17), и пространства методов для общего использования (рис. 3.18).

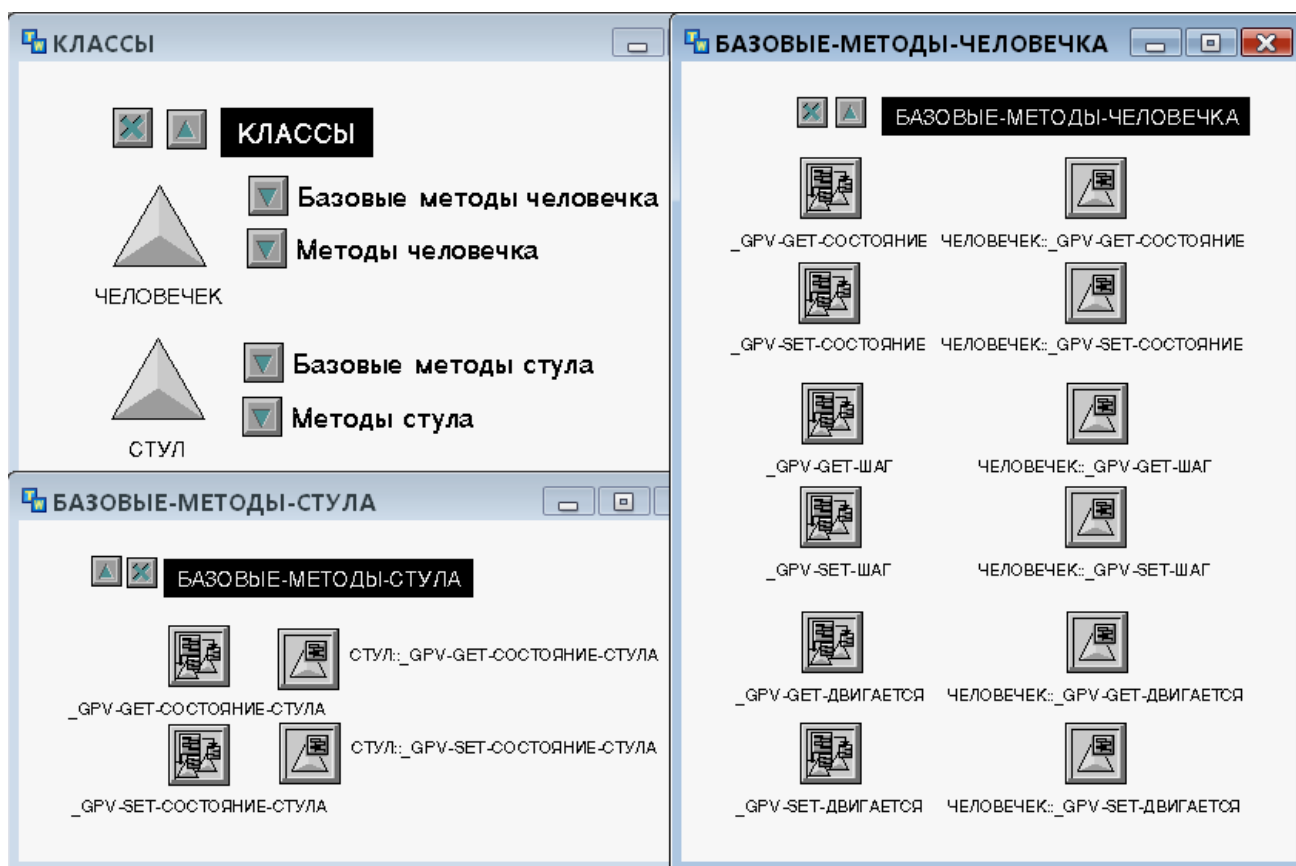


Рис. 3.17. Список базовых методов

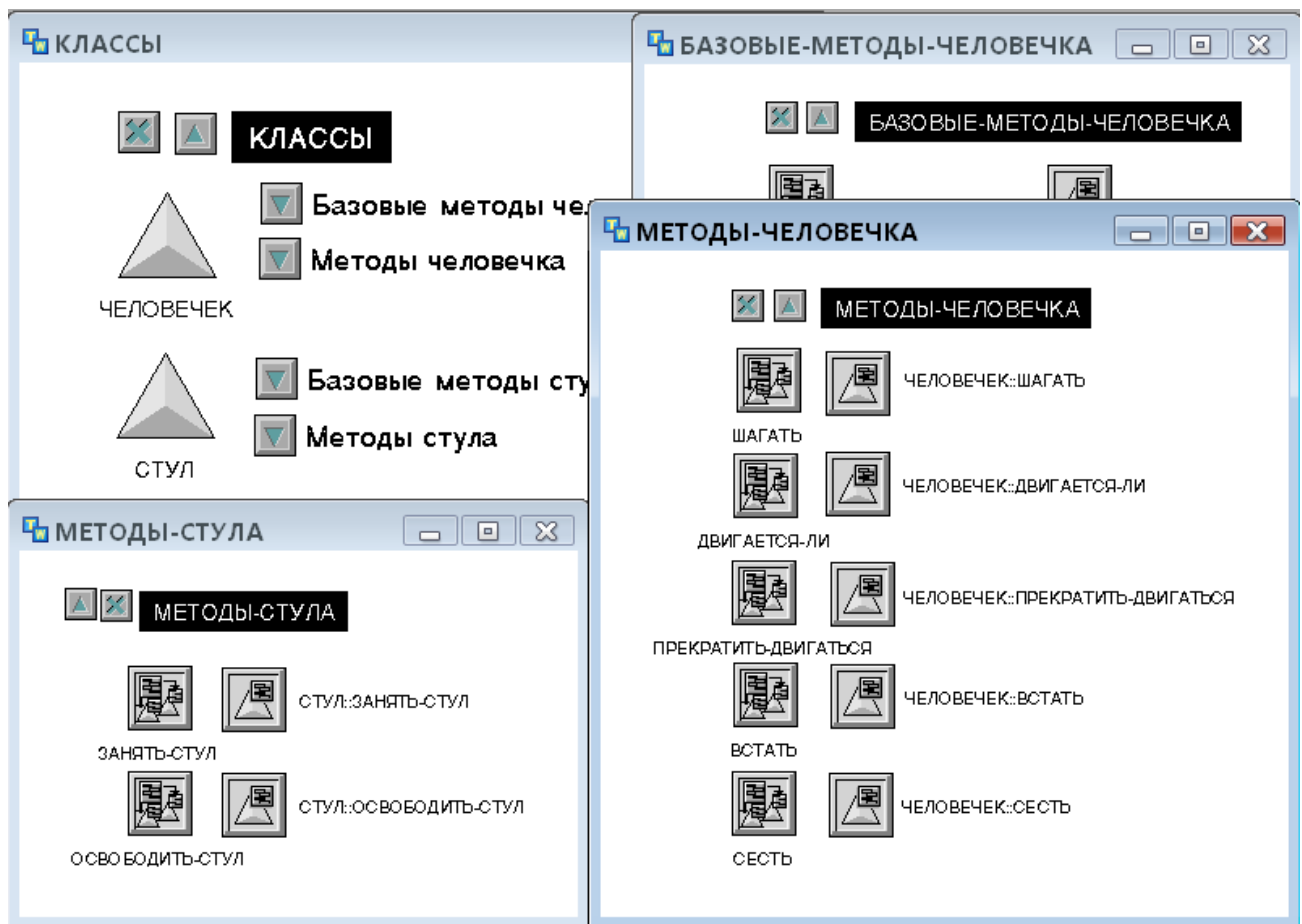


Рис. 3.18. Остальные методы классов

2. Воспользуйтесь контекстным меню пространства *New Definition* → *procedure* → *method* для создания метода получения значения атрибута «состояние» у человека. Поместите метод в пространство **Базовые-методы-человечка** и заполните в его таблице самую широкую область, выделенную для кода метода. Значение поля *Qualified-name* появится само при правильном заполнении кода в виде *имя-класса-первого-параметра::имя-метода* (рис. 3.19). Например, метод чтения *человек::_gpv-get-шаг*, таков:

```
_gpv-get-шаг(чел: class человек) = (integer)
begin
  return the шаг of чел;
end
```

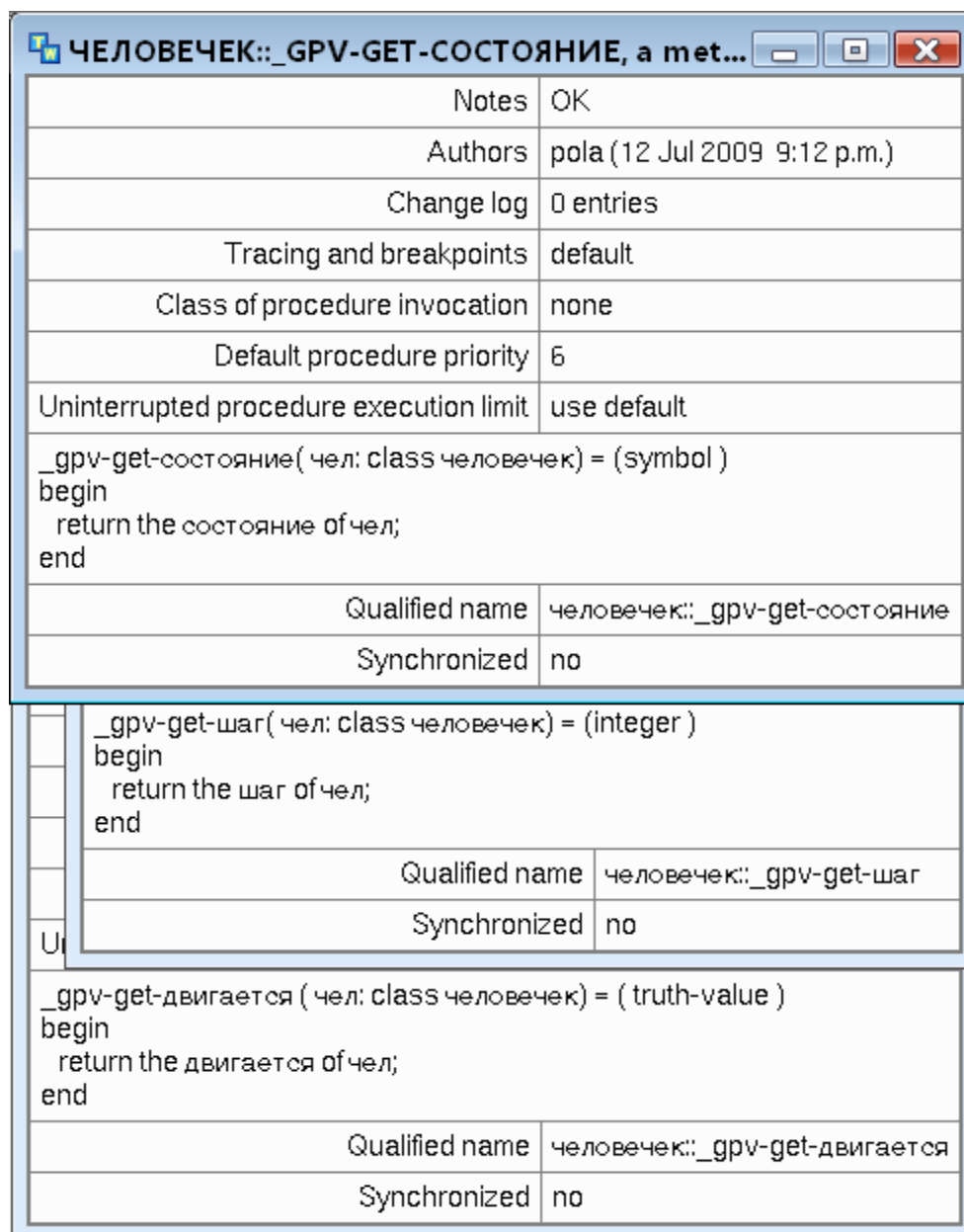


Рис. 3.19. Методы для чтения значения атрибута

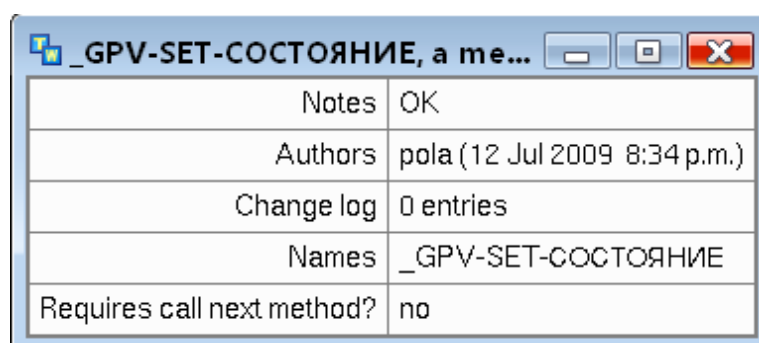


Рис. 3.20. Декларация метода

3. Для каждого метода необходимо создать еще один объект – Объявление-метода (method-declaration) с помощью соседнего пункта

контекстного меню рабочего пространства. В объявлении достаточно заполнить только атрибут Имя (Names), написав имя метода (рис. 3.20);
(Рис. 3.20 лучше переместить сюда (после ссылки на него!))

4. Аналогично создаем методы установки значения атрибутов, например, метод изменения состояния человека *человек::_gpv-set-состояние*:

```
_gpv-set-состояние(чел: class человек, значение: symbol)
begin
  conclude that the состояние of чел = значение;
end
```

5. Создадим и остальные методы **для** человека и стула:

Метод *человек::двигается-ли*:

```
двигается-ли (чел: class человек) = (truth-value)
результат: truth-value;
begin
  результат = call _gpv-get-двигается (чел);
  return результат
end
```

Метод *человек::прекратить-двигаться*:

```
прекратить-двигаться (чел: class человек)
begin
  call _gpv-set-двигается (чел, false);
end
```

Метод *человек::встать*:

```
встать (чел: class человек)
begin
  call _gpv-set-состояние (чел, the symbol стоит);
end
```

Метод *человек::сесть*:

```
сесть (чел: class человек)
begin
  call _gpv-set-состояние (чел, the symbol сидит);
end
```

Метод *человек::шагать*, передвигающий человека на значение шага по горизонтали каждую секунду, пока состояние человека остается «идет»:

```
шагать (чел: class человек)
состояние: symbol;
шаг: integer;
```

```

begin
  call _gprv-set-двигается(чел, true);
  repeat
    шаг = call _gprv-get-шаг(чел);
    состояние = call _gprv-get-состояние(чел);
    if состояние = the symbol идет then move чел by (шаг, 0)
    else exit if true;
    wait for 1 second;
    allow other processing;
  end;
  call _gprv-set-двигается( чел, false);
end

```

Метод стул::_gprv-get-состояние-стула:

```

_gprv-get-состояние-стула (ст: class стул) = (symbol)
begin
  return the состояние of ст;
end

```

Метод стул::_gprv-set-состояние-стула:

```

_gprv-set-состояние-стула (ст: class стул, значение: symbol)
begin
  conclude that the состояние of ст = значение;
end

```

Метод стул::занять-стул:

```

занять-стул(ст: class стул)
begin
  call _gprv-set-состояние-стула (ст, the symbol занят)
end

```

Метод стул::освободить-стул:

```

освободить-стул(ст: class стул)
begin
  call _gprv-set-состояние-стула (ст, the symbol свободен)
end

```

6. Для всех созданных методов создаем Объявления метода (*method-declaration*) (см. пункт 3).

3.5. Правила и процедуры

3.5.1. Меняем иконку

Для начала, заставим человечка менять свой облик (иконку) при изменении его состояния. Для реакции на это событие, создадим правило. Но поскольку для изменения иконки потребуется не **одно** действие, а

целый набор, создадим и процедуру для последовательного выполнения этих изменений.

1. На пространстве «Моделирование» создайте человечка и назовите его, например, **Человечек-1**.
2. Создайте три кнопки для управления его состоянием: стой, сиди, иди; с помощью меню *New Button* → *action-button*. Заполните атрибуты *Label* и *Action* у кнопок (рис. 3.21). Например, для кнопки «Стой!» подойдет действие: *start _gpv-set-состояние (человечек-1, the symbol стоим)* с вызовом метода, либо непосредственно *conclude that the состояние of человек-1 = the symbol стоим*
3. На пространстве «Процедуры» создадим процедуру для изменения иконки в зависимости от текущего состояния человечка:

```
изменить-изображение-человечка (чел : class человек)
begin
  case (the состояние of чел) of
    сидит: begin
      change the сидя icon-color of чел to blue;
      change the шагом icon-color of чел to transparent;
      change the стоя icon-color of чел to transparent;
      end;
    стоит: begin
      change the сидя icon-color of чел to transparent;
      change the шагом icon-color of чел to transparent;
      change the стоя icon-color of чел to blue ;
      if there exists a стул Y such that
        (Y is быть-занятым чел) then
        begin
          conclude that Y is not быть-занятым чел;
          call освободить-стул (Y);
        end;
      end;
    идет: begin
      change the сидя icon-color of чел to transparent;
      change the шагом icon-color of чел to blue;
      change the стоя icon-color of чел to transparent;
      if there exists a стул Y such that
        (Y is быть-занятым чел) then
        begin
          conclude that Y is not быть-занятым чел;
          call освободить-стул (Y);
        end;
      end;
    end;
  end
end
```

4. На пространстве «Правила» создадим правило, **утверждающее**, что как только атрибут **«состояние»** какого-либо **человечка X** с пространства

«Моделирование» получает значение, надо вызвать выше написанную процедуру для этого **человечка X**:

whenever the состояние of any человек X upon моделирование receives a value then start изменить-изображение-человечка (X)

5. Для проверки работы правила и процедуры, нажмите по очереди все из трех созданных кнопок, иконка человечка **человечек-1** должна поменяться сразу после изменения его состояния.

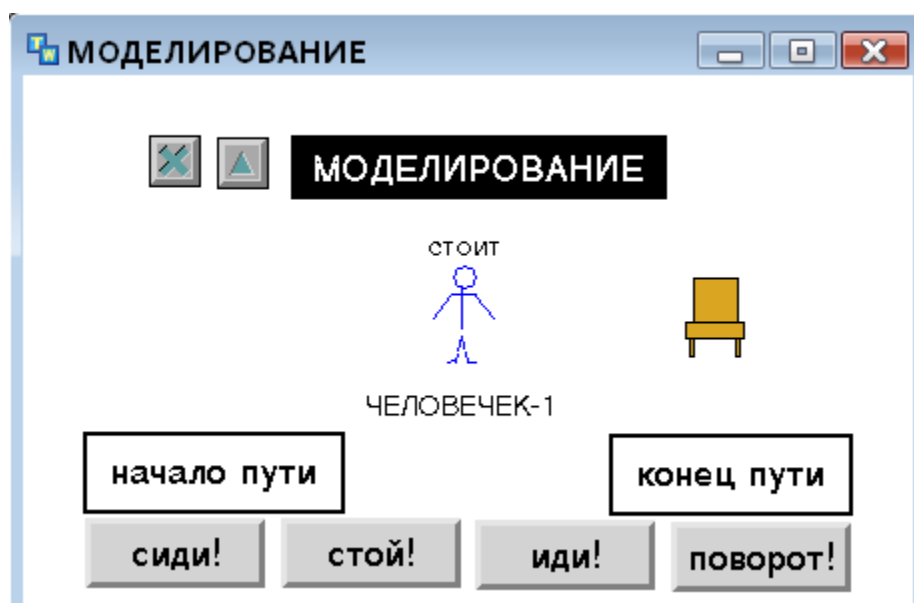


Рис. 3.21. Моделирование

3.5.2. Движение человечка

На пространства правил создадим новое правило, по которому как только какой-либо человечек получает значение атрибута «состояние» и это значение «идет», запускается процедура, приводящая человечка в движение, если он неподвижен.

Правило:

whenever the состояние of any человек X upon моделирование receives a value and when the состояние of X is идет then start процедура-шагать(X)

Процедура:

```
процедура-шагать (чел: class человек)
двиг: truth-value;
begin
    двиг = call двигается-ли (чел);
    if двиг = false then call шагать(чел);
end
```


Атрибут «двигается» необходим, чтобы для одного человечка был запущен только один метод *человечек::шагать*, вначале которого как раз значение атрибута «двигается» изменяется на истину, а после завершения метода – на ложь.

3.5.3. Поворот

1. Ограничим дистанцию, по которой ходит человечек, создав два ограничителя – начало и конец пути (рис. 3.21). С помощью контекстного меню рабочего пространства *New Free Text* → *free-text* создайте ориентиры для человечка, расположив их слева и справа от человечка, и дав имена соответственно – *начало* и *конец*. Имена нужны, чтобы можно было ссылаться на них еще в одном правиле:

for any человечек X upon моделирование
if the состояние of X = the symbol идём and
*(the item-x-position of X <= the item-x-position of **начало** and the шаг of X < 0*
or
*the item-x-position of X >= the item-x-position of **конец** and the шаг of X > 0)*
then conclude that the шаг of X = - the шаг of X

Данное правило определяет, когда надо поворачивать (менять значение шага на противоположное) при движении вправо с положительным шагом или влево с отрицательным.

В отличие от предыдущих правил, которые сработают когда бы ни (whenever) изменилось значение атрибута человечка, для периодического запуска данного правила надо установить *Интервал-сканирования* (Scan-interval) в таблице атрибутов данного правила, допустим, 1 секунда.

2. Добавим также кнопку «Поворот!» для *человечка-1* с действием: *conclude that the шаг of ЧЕЛОВЕЧЕК-1 = - the шаг of ЧЕЛОВЕЧЕК-1*

3.5.4. Стулья

1. Увеличим количество задействованных объектов, добавив с помощью палитры на рабочее пространство еще несколько человечков и стульев (рис. 3.22).
2. Переделаем кнопку «иди!» так, чтобы она изменяла состояние не только *человечка-1*, но и всех человечков на рабочем пространстве «Моделирование». Для этого изменим действие по кнопке на более общее: *conclude that the состояние of every человечек upon моделирование = the symbol идём*
3. Создадим правило, согласно которому, если человечек встречает свободный стул на одной (плюс/минус 1) с ним вертикали пространства «Моделирование», он садится на этот стул, меняя свое и его состояние и устанавливая отношение «сидеть-на».

for any человек X upon моделирование
for any стул Y upon моделирование
if the состояние of X = the symbol идет and
the состояние of Y = the symbol свободен and
 $abs(\text{the item-x-position of X} - \text{the item-x-position of Y}) < 2$
then conclude that the состояние of Y = the symbol занят and
conclude that the состояние of X = the symbol сидит and
move Y to (the item-x-position of X, the item-y-position of X) and
conclude that X is now сидеть-на Y

Для данного правила также требуется установить интервал сканирования. Пусть будет снова 1 секунда.

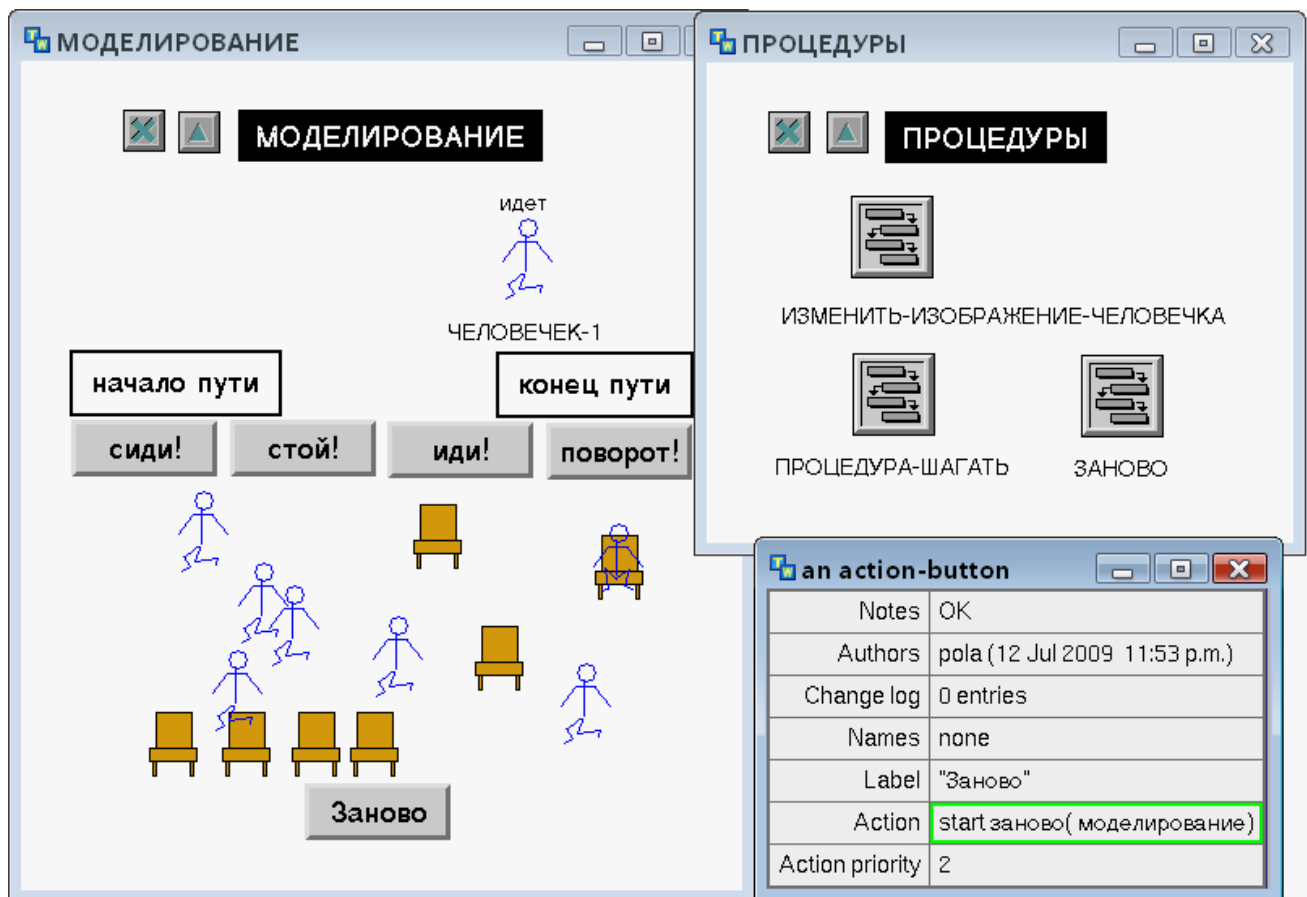


Рис. 3.22. Процедура инициализации

- После того, как все сели или все стулья заняты повторное нажатие на кнопку «иди!» не приводит к желаемому результату, поскольку все тут же садятся обратно. Потому создадим еще одну кнопку, которая будет запускать процедуру (рис. 3.22), задающую всем человечкам с пространства «Моделирование» начальный шаг и направление, и сдвигающую их с текущей позиции, а так же переносящую все стулья на задний план, чтобы они не загораживали человечков.

Код процедуры заново:

```
заново (ws: class kb-workspace)
чел: class человек;
стульчик: class стул;
шаг: integer;
begin
  for чел = each человек upon ws do
    call прекратить-двигаться (чел);
    call встать (чел);
    call изменить-изображение-человечка (чел);
    шаг = random (2, 5);
    if random(10) > 5 then шаг = - шаг;
    move чел by (шаг, 0);
    conclude that the шаг of чел = шаг;
  end;

  allow other processing;

  for стульчик = each стул upon ws do
    call освободить-стул ( стульчик);
    call g2-drop-item-to-bottom( стульчик);
  end;
end
```

3.6. Дальнейшая разработка модели

Расширим и определим модель следующим образом: пусть объекты класса «человек» символизируют лица, принимающие решения (ЛПР). Будем считать, что процесс принятия решения происходит при взаимодействии ЛПР, сидящего на стуле, и объекта нового класса «пульт». Опишем этот новый класс, а также все прочие элементы, позволяющие промоделировать данную концепцию.

3.6.1. Создание нового класса «пульт»

1. С помощью контекстного меню пространства создайте новый класс, выбрав пункт *New Definition* → *class-definition* → *class-definition*, и расположите его на пространстве «классы».
2. Заполните атрибуты из таблицы созданного класса в соответствии с рис. 3.23.
 - Имя нового класса – «пульт».
 - Имя непосредственного суперкласса – «object».
 - Специфичный атрибут человека – «состояние» перечисляемого типа, с двумя возможными значениями: «работает» и «выключен»; по умолчанию начальное значение «выключен».
 - Второй атрибут – «время-пользования» целого типа, которому по умолчанию зададим значение 5, этот атрибут показывает, сколько времени ЛПР может пользоваться данным пультом.

- Третий атрибут – «таймер» целого типа, который показывает, сколько времени работы осталось у текущего пользователя, начальное значение – 5.

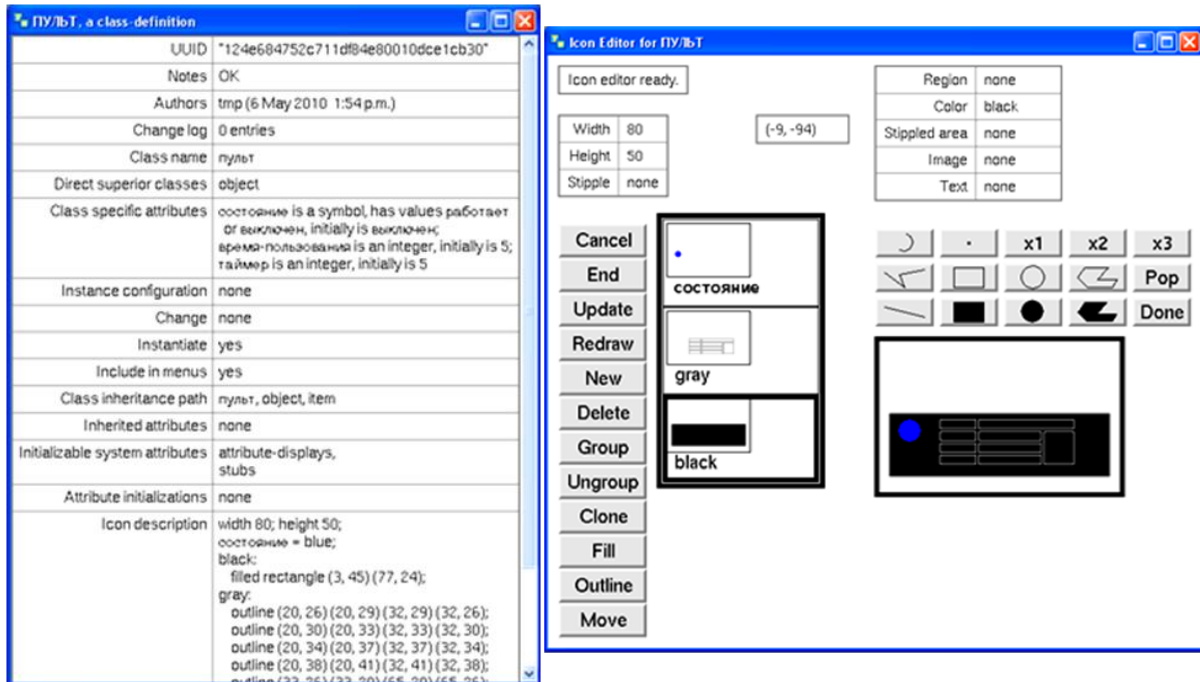


Рис. 3.23. Класс «пульт» (убрать «.»)

3.6.2. Методы класса «пульт»

Создадим методы для работы с объектами класса «пульт».

1. Создайте в рабочем пространстве классов ссылки на новые рабочие пространства *Базовые-методы-пульта*, куда поместим методы для чтения и записи значений всех атрибутов без проверки корректности данных, и *Методы-пульта* для методов общего использования (рис. 3.24).

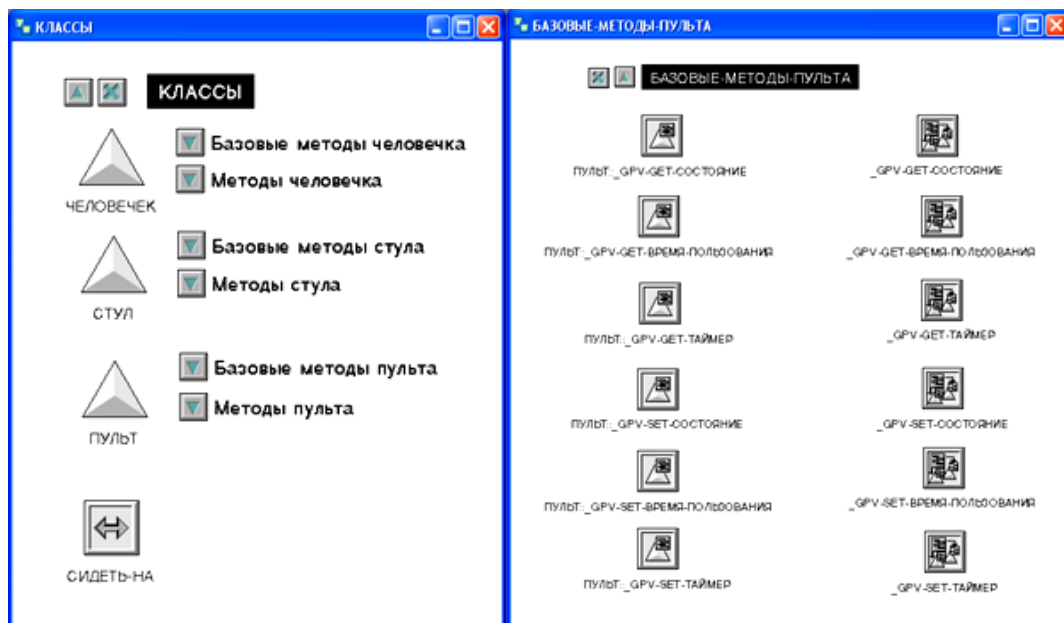


Рис. 3.24. Рабочие пространства «Классы» и «Базовые-методы-пульта»

2. Воспользуйтесь контекстным меню пространства *New Definition* → *procedure* → *method* для создания трех методов получения значения атрибутов и трех методов установки значения атрибутов пульта. Поместите каждый метод в пространство *Базовые-методы-человечка* (рис. 3.25).

<pre>_gpv-get-таймер (пул: class пульт)=(integer) begin return the таймер of пул; end</pre>
<pre>_gpv-get-время-пользования (пул: class пульт)=(integer) begin return the время-пользования of пул; end</pre>
<pre>_gpv-get-состояние (пул: class пульт)=(integer) begin return the состояние of пул; end</pre>
<pre>_gpv-set-таймер (пул: class пульт, значение: integer) begin conclude that the таймер of пул = значение end</pre>
<pre>_gpv-set- время-пользования (пул: class пульт, значение: integer) begin conclude that the время-пользования of пул = значение end</pre>
<pre>_gpv-set- состояние (пул: class пульт, значение: integer) begin conclude that the состояние of пул = значение end</pre>

Рис. 3.25. Базовые методы класса «пульт» (убрать «.»)

3. Для каждого метода создадим *method-declaration* с помощью соседнего пункта контекстного меню рабочего пространства.
4. Добавим на пространство «методы-пульта» метод *Работает-ли*, позволяющий определить, работает ли данный пульт в настоящий момент, и его декларацию (рис. 3.26).

```
работает-ли (пул: class пульт) = (truth-value)
сост: symbol;
результат: truth-value;
begin
  сост = call _gpv-get-состояние (пул);
  результат = false;
  if сост = the symbol работает
    then результат = true;
  return результат;
end
```

ПУЛЬТ::РАБОТАЕТ-ЛИ, a method	
UUID	"083527a358fb11df84ee0010dce1cb30"
Notes	OK
Authors	tmp (6 May 2010 2:40 p.m.)
Change log	0 entries
Tracing and breakpoints	default
Class of procedure invocation	none
Default procedure priority	6
Uninterrupted procedure execution limit	use default
работает-ли (пул: class пулт) = (truth-value) соот: symbol; результат: truth-value; begin соот = call_gpv-get-состояние (пул); результат = false; if соот = the symbol работает then результат = true; return результат; end	
Qualified name	пулт:работает-ли
Synchronized	no

Рис. 3.26. Метод класса «пулт» *работает-ли* (убрать «.»)

3.6.3. Процедуры и правила для корректного моделирования

1. На пространстве «Процедуры» создадим процедуру для изменения иконки в зависимости от текущего состояния пульта:

```
изменить-изображение-пульта (пул: class пулт)
begin
  case (the состояние of пул) of
    выключен: begin
      change the состояние icon-color of пул to blue;
    end;
    работает: begin
      change the состояние icon-color of пул to red;
    end;
  end;
end;
end
```

2. На пространстве «Правила» создадим правило, гласящее, что как только атрибут состояние какого-либо пульта X с пространства «Моделирование» получает значение, надо вызвать выше написанную процедуру для этого пульта X:

```
whenever the состояние of any пулт X
  upon моделирование receives a value
then start изменить-изображение-пульта (X)
```

3. На пространстве «Процедуры» создадим процедуру «сесть-за-пулт», которая отображает работу конкретного ЛПР с конкретным пультом: если на рабочем пространстве есть неактивный пулт, он перемещается к ЛПР и переводится в состояние «работает». По истечении срока,

определенного атрибутом пульта «время-пользования», последний переводится в состояние «выключен».

```
сесть-за-пульт (X: class человек )
Y: class пульт;
begin
  if there exists a пульт Y upon моделирование
    such that (the состояние of Y = the symbol выключен)
  then begin
    move Y to (the item-x-position of X,
               the item-y-position of X);
    conclude that the состояние of Y = the symbol работает;
    conclude that the таймер of Y = the время-пользования of Y;
    repeat
      exit if the таймер of Y = 0;
      wait for 1 second;
      conclude that the таймер of Y = the таймер of Y - 1;
    end;
    conclude that the состояние of Y = the symbol выключен;
    conclude that the таймер of Y = the время-пользования of Y;
  end;
end
```

4. На пространстве «Правила» создадим правило, запускающее эту процедуру в момент, когда какой-либо ЛПР садится на стул:

```
whenever the состояние of any стул Y
  upon моделирование receives a value
  and
when there exists a человек X
  such that (X is сидеть-на Y)
then start сесть-за-пульт(X)
```

5. На пространстве «Процедуры» изменим код процедуры инициализации «заново» с учетом нового объекта «пульт»:

```
заново (ws: class kb-workspace)
чел: class человек;
стульчик: class стул;
пул: class пульт;
шаг: integer;
begin
  for чел = each человек upon ws do
    call прекратить-двигаться (чел);
    call встать (чел);
    call изменить-изображение-человечка (чел);
    шаг = random (2, 10);
    if random(10) > 5 then шаг = -шаг;
    move чел by (2 * шаг, 0);
    conclude that the шаг of чел = шаг;
  end;
```

```

allow other processing;

for стульчик = each стул upon ws do
    call освободить-стул (стульчик);
    call g2-drop-item-to-bottom(стульчик);
end;

for пул = each пульта upon ws do
    conclude that the состояние of пул = the symbol выключен;
    conclude that the время-пользования of пул = 5;
    conclude that the таймер of пул = 5;
    move пул to (the item-x-position of конец,
                the item-x-position of конец);
end;
end
end

```

Теперь, когда база знаний системы корректно дополнена, приступим непосредственно к моделированию.

3.6.4. Моделирование процесса принятия решений

Создайте объект класса «пульта» и поместите его на рабочее пространство «моделирование». С помощью пункта контекстного меню таблицы атрибутов объекта Show attribute display вынесите значение атрибута «состояние» и «таймер» на рабочее пространство (рис. 3.27).



Рис. 3.27. Рабочее пространство «моделирование» до запуска

При нажатии на кнопку «иди» первому ЛПР, севшему за стул, представится возможность в течение 5 секунд пользоваться пультом (рис. 3.28).



Рис. 3.28. ЛПР, сидящий на стуле, работает с пультом (убрать «.»)

3.6.5. Отправка сообщений

Организуем отправку сообщений, фиксирующих состояние пульта, на панель сообщений (Message Board) приложения. Для этого на рабочем пространстве *Методы-пульта* создадим метод *сообщение*:

```
сообщение (пул: class пульт)
сост: symbol;
текст: text;
время, тайм: integer;
begin
    сост = call _grpv-get-состояние (пул);
    if сост = the symbol работает
        then текст = "используется ЛПР"
        else текст = "свободен";
    время = call _grpv-get-время-пользования (пул);
    тайм = call _grpv-get-таймер (пул);
    post "
Пульт: [текст];
время пользования: [время - тайм] секунд;
осталось: [тайм] секунд";
end
```

При вызове этого метода на панели сообщений появится сообщение, содержащее сведения о том, свободен ли пульт, сколько времени пульт уже используется ЛПР и сколько времени осталось.

Изменим код процедуры *«сесть-за-пульт»* таким образом, чтобы постоянно следить за работой пульта посредством отправки сообщений:

```
сесть-за-пульт (X: class человек )
Y: class пульт;
begin
  if there exists a пульт Y upon моделирование
    such that (the состояние of Y = the symbol выключен)
  then begin
    move Y to (the item-x-position of X,
               the item-y-position of X);
    conclude that the состояние of Y = the symbol работает;
    conclude that the таймер of Y = the время-пользования of Y;
    repeat
      exit if the таймер of Y = 0;
      call сообщение(Y);
      wait for 1 second;
      conclude that the таймер of Y = the таймер of Y - 1;
    end;
    conclude that the состояние of Y = the symbol выключен;
    conclude that the таймер of Y = the время-пользования of Y;
    call сообщение(Y);
  end;
end
```

Теперь при запуске приложения в зависимости от ситуации на рабочем пространстве «моделирование» будут появляться различные сообщения о текущем состоянии пульта (рис. 3.29).

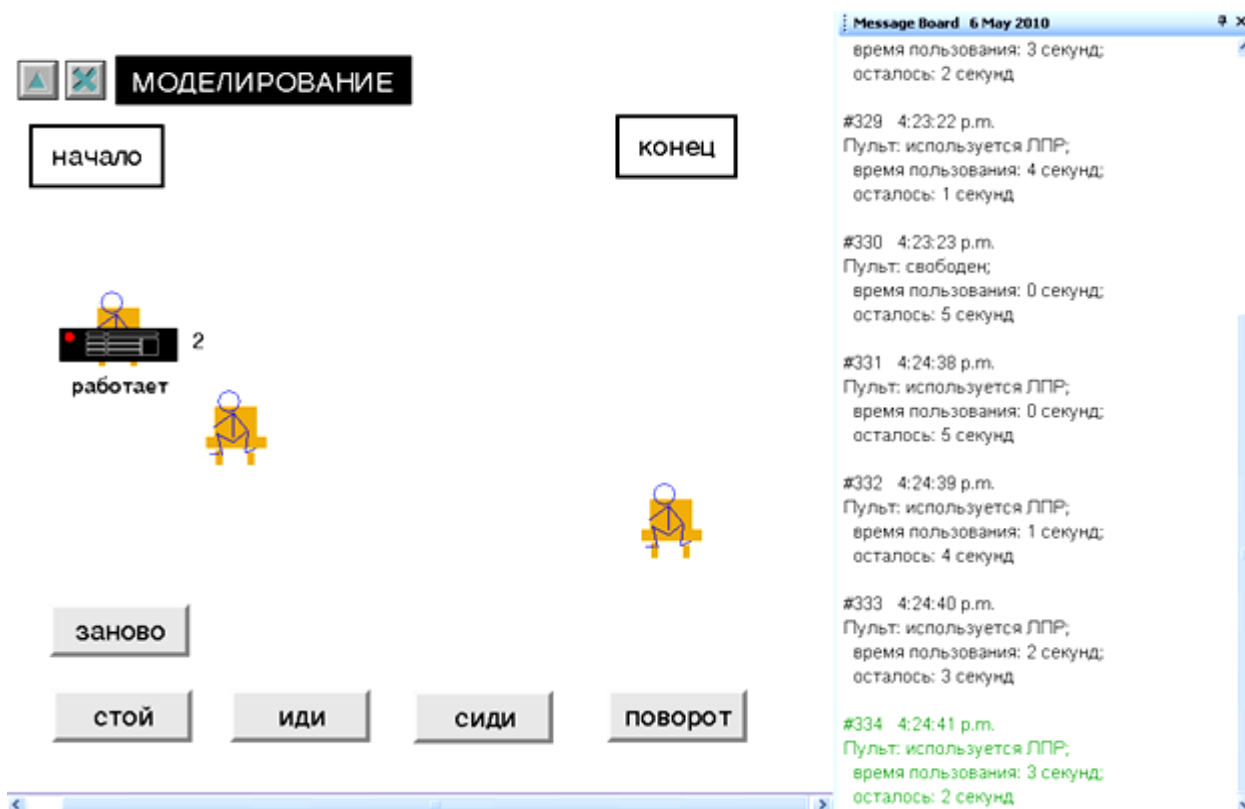


Рис. 3.29. Отправка сообщений (убрать «.»)

3.7. Контрольные вопросы к разделу 5

1. Как выполнить запуск сервера и клиента?
2. Зачем нужны рабочие пространства?
3. Как создать рабочее пространство?
4. Зачем надо давать имя рабочему пространству?
5. Как перейти к нужному рабочему пространству?
6. Как подключить модуль GFR?
7. Как создать иерархию рабочих пространств?
8. Как создать новый класс? Как указать его место в иерархии классов?
9. Как создать палитру?
10. Как описать метод класса? Как создать декларацию метода?
11. Как вызвать метод? Как вызвать унаследованный метод?
12. Как можно создать иконку класса?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. **Вагин В.Н., Еремеев А.П.** Некоторые базовые принципы построения интеллектуальных систем поддержки принятия решений реального времени // Известия РАН. Теория и системы управления, 2001, № 6. с. 114-123.
2. **Еремеев А.П.** Экспертные модели и методы принятия решений. М.: Изд-во МЭИ, 1995, 111 с.
3. **Башлыков А.А., Еремеев А.П.** Экспертные системы поддержки принятия решений в энергетике / Под ред. А.Ф. Дьякова. М.: Изд-во МЭИ, 1994, 216 с.
4. **Башлыков А.А., Вагин В.Н., Еремеев А.П.** Экспертные системы поддержки интеллектуальной деятельности операторов АЭС // Вестник МЭИ. N 4, 1995, с. 27-36.
5. **Статические и динамические экспертные системы:** Учебное пособие / Э.В. Попов, И.Б. Фоминых, Е.Б. Кисель, М.Д. Шапот. – М.: Финансы и статистика, 1996, 320 с.
6. **Еремеев А.П., Чибизова Н.В.** Проектирование экспертных систем реального времени на основе инструментального комплекса G2 – М.: Изд-во МЭИ, 1998, 40 с.
7. **G2 Reference Manual. Version 3.0.** // Gensym Corp., Cambridge, MA, USA, 1993, 600 p.
8. **Еремеев А.П., Симонов Д.Н., Чибизова Н.В.** Реализация прототипа системы поддержки принятия решений реального времени на основе инструментального комплекса G2 // Программные продукты и системы, 1996, № 3, с. 21-26.
9. **Еремеев А.П.** О корректности продукционной модели принятия решений на основе таблиц решений // Автоматика и телемеханика, 2001, № 10, с. 78-90.
10. **Еремеев А.П., Денисенко Л.С.** Обработка недоопределенной информации в системе поддержки принятия решений реального времени применительно к оперативной службе электростанций // Известия РАН. Энергетика, 2002, № 2, с. 32-43.
11. **Вагин В.Н., Еремеев А.П.** Исследования и разработки кафедры прикладной математики по конструированию интеллектуальных систем поддержки принятия решений на основе нетрадиционных логик // Вестник МЭИ, № 5, 2008, с. 16-26.
12. **Поспелов Д.А.** Моделирование рассуждений. Опыт анализа мыслительных актов. М.: Радио и связь, 1989, 184 с.
13. **Поспелов Д.А., Осипов Г.С.** Введение в прикладную семиотику // Новости искусственного интеллекта, 2002, № 6, с. 28-35.
14. **Вагин В.Н., Головина Е.Ю., Загорянская А.А., Фомина М.В.** Достоверный и правдоподобный вывод в интеллектуальных системах /

- Под ред. В.Н. Вагина, Д.А. Поспелова. 2-е изд., испр. и доп. М.: ФИЗМАТЛИТ, 2008, 712 с.
15. **Еремеев А.П., Троицкий В.В.** Темпоральные рассуждения в интеллектуальных системах поддержки принятия решений. Коллективная монография «Интеллектуальные системы». / Под ред. В.М. Курейчика. М.: Физматлит, 2005, с. 169-180.
 16. **Еремеев А.П., Куриленко И.Е.** Применение временных рассуждений в интеллектуальных системах реального времени // Интеллектуальные системы. Коллективная монография. Выпуск второй. / Под ред. В.М. Курейчика. М.: Физматлит, 2007, с. 114-130.
 17. **Варшавский П.Р., Еремеев А.П.** Реализация методов поиска решения на основе аналогий и прецедентов в системах поддержки принятия решений // Вестник МЭИ, 2006, № 2, с. 77-87.
 18. **Виноградов О.В., Еремеев А.П.** Современные программные средства поддержки принятия решений на основе таблиц решений // Интеллектуальные системы. Коллективная монография. Выпуск 3. / Под ред. В.М. Курейчика. М.: Физматлит, 2009, с. 140-154.
 19. **Еремеев А.П., Подогов И.Ю.** Методы подкрепленного обучения для систем поддержки принятия решений реального времени // Вестник МЭИ, № 2, 2009, с. 153-161.
 20. **Еремеев А.П., Тихонов Д.А.** Применение средств параллельной обработки информации в интеллектуальных системах поддержки принятия решений для управления энергообъектом // Вестник МЭИ, 1999, № 2, с. 54-60.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
1. ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ И АРХИТЕКТУРА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ	5
1.1. Архитектура и основные принципы построения ИСППР РВ	5
1.2. ИСППР РВ как динамическая система семиотического типа.....	7
1.3. Адаптивность и модифицируемость	8
1.4. Распределенная и параллельная обработка информации	14
1.5. Когнитивная графика и гипертекст в отображении информации	17
1.6. Контрольные вопросы к разделу 1	19
2. ОСНОВНЫЕ КОМПОНЕНТЫ КОМПЛЕКСА G2	20
2.1. Главное меню и меню рабочих пространств.....	20
2.2. Сущности	20
2.3. Редактирование выполняемых сущностей	20
2.4. Выполнение приложения	23
2.5. Представление конкретных знаний.....	24
2.5.1. Типы данных и атрибуты	24
2.5.2. Команды	25
2.5.3. Командные кнопки	26
2.5.4. Правила.....	26
2.5.5. Процедуры	27
2.6. Представление общих знаний.....	30
2.6.1. Общие правила	30
2.6.2. Утверждение <i>every</i>	30
2.6.3. Порты.....	30
2.6.4. Общие процедуры	31
2.6.5. Проверка на существование.....	31
2.6.6. Ссылки на арифметические конструкции и счетчик.....	31
2.6.7. Классы объектов.....	32
2.6.8. Классы связей	33
2.6.9. Рекомендации	33
2.7. Ограничения базы знаний	34
2.7.1. Возможности и синтаксис ограничений	34
2.7.2. Изменение статуса пользователя.....	35
2.7.3. Ограничение класса	35
2.7.4. Пункты меню, определяемые пользователем	35
2.8. Команда <i>Inspect</i>	35
2.9. Параметры и истории	36
2.10. Поиск данных и реальное время.....	38
2.10.1. Переменные, поиск данных, обратный вывод	38
2.10.2. Конструкции, используемые при работе со временем.....	40

2.10.3.	Процедуры в контексте реального времени	41
2.11.	Представление динамических знаний	42
2.11.1.	Динамические сущности	42
2.11.2.	Списки	43
2.11.3.	Массивы	43
2.11.4.	Отношения	44
2.11.5.	Сообщения	45
2.12.	Пользовательский интерфейс	45
2.12.1.	Кнопки и слайдеры.....	45
2.12.2.	Дисплеи, диаграммы, графики, таблицы	46
2.13.	Подсистема моделирования	46
2.13.1.	Назначение и основные характеристики	46
2.13.2.	Взаимосвязь с другими компонентами G2	47
2.13.3.	Формулы моделирования	48
2.14.	Планировщик.....	50
2.15.	Контрольные вопросы к разделу 2	51
3.	ПОРЯДОК СОЗДАНИЯ МОДЕЛИ В СРЕДЕ G2	52
3.1.	Запуск G2	52
3.2.	Создание рабочего пространства	52
3.3.	Создание иерархии рабочих пространств	54
3.3.1.	Порядок подключения модуля.....	54
3.3.2.	Создание кнопок навигации и изменение режимов доступа.....	57
3.3.3.	Командные кнопки	59
3.4.	Создание классов и объектов.....	60
3.4.1.	Создание классов.....	60
3.4.2.	Создание объектов	65
3.4.3.	Создание палитры	65
3.4.4.	Методы классов	65
3.5.	Правила и процедуры	70
3.5.1.	Меняем иконку	70
3.5.2.	Движение человечка	72
3.5.3.	Поворот.....	73
3.5.4.	Стулья.....	73
3.6.	Дальнейшая разработка модели	75
3.6.1.	Создание нового класса «пульт»	75
3.6.2.	Методы класса «пульт»	76
3.6.3.	Процедуры и правила для корректного моделирования	78
3.6.4.	Моделирование процесса принятия решений	80
3.6.5.	Отправка сообщений.....	81
3.7.	Контрольные вопросы к разделу 5	83
	БИБЛИОГРАФИЧЕСКИЙ СПИСОК	84

Учебное издание

Еремеев Александр Павлович
Гречкина Полина Викторовна
Чибизова Наталья Владимировна

**КОНСТРУИРОВАНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ ПОДДЕРЖКИ
ПРИНЯТИЯ РЕШЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ
НА ОСНОВЕ ИНСТРУМЕНТАЛЬНОГО КОМПЛЕКСА G2**

Учебное пособие
по курсам

«Экспертные системы» и «Интеллектуальные системы»
для студентов, обучающихся по направлениям
«Прикладная математика и информатика»,
«Информатика и вычислительная техника»,
«Информационные системы и технологии»

Редактор издательства Г.Ф. Раджабова

Темплан издания МЭИ 2004(II), метод.	Подписано в печать
Печать офсетная	Формат 60×84/16
Тираж 150 экз	Изд. № 203
Заказ	Цена 23 руб.
Физ. печ. л. 5,5	

ЗАО «Издательский дом МЭИ», 111250, Москва, Красноказарменная ул., д. 14
Отпечатано в типографии НИИ «Геодезия», 141292, Московская обл.,
г. Красноармейск, просп. Испытателей, д. 14

