МИНИСТЕРСТВО ОБРАЗОВАНИЯ РФ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

На правах рукописи
УДК 519.68:[681.3.06+007.51]

СОШНИКОВ Дмитрий Валерьевич

МЕТОДЫ И СРЕДСТВА ПОСТРОЕНИЯ РАСПРЕДЕЛЕННЫХ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ НА ОСНОВЕ ПРОДУКЦИОННО-ФРЕЙМОВОГО ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Специальность: 05.13.11 “Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей”

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель
Кандидат физико-математических наук
доцент Зайцев В. Е.

Москва 2002

Оглавление

Список иллюстраций

Список алгоритмов

Введение

В настоящее время компьютерные сети различных масштабов приобретают все большую распространенность как в нашей стране, так и во всем мире. Развитие всемирной сети Интернет приводит к тому, что уже сейчас большинство компьютерных систем в мире объединены в единую информационную инфраструктуру, содержащую огромное число общедоступных сервисов и гигантские объемы информации. Благодаря развитию удобных средств доступа к информации, в первую очередь всемирной паутины, популярность Интернет быстро растет не только среди профессионалов, но и в более обширных группах пользователей.

Дальнейшее развитие интерактивных средств доступа к данным в Интернет оказывается невозможным в рамках традиционного подхода к представлению информации в виде гипертекста и гипермедиа с использованием простых языков разметки типа HTML или VRML. Частично эта проблема решается использованием различных процедурных включений, а именно Java-скриптов и Java-аплетов (выполняемых на компьютере-клиенте) или механизмов CGI-скриптов и сервлетов (выполняемых на сервере). Таким образом, в рамках Интернет становится возможным представлять не только пассивную текстовую и мультимедийную информацию, но и более сложные процедуральные структуры, взаимодействующие с пользователем на уровне локальных приложений с достаточно развитой функциональностью. Стремительно растущие области применения интернет-приложений включают в себя автоматизацию бизнес-процессов как внутри предприятия (интранет-системы), так и на внешнем рынке (приложения электронной коммерции), системы организации интернет-сообществ, системы дистанционного обучения и тестирования и многие другие.

Простейшие клиент-серверные архитектуры позволяют осуществлять доступ клиентов к одному централизованному ресурсу сервера. Однако более сложные ситуации могут потребовать объединения нескольких вычислительных или информационных ресурсов для совместного функционирования в рамках сети, что приводит к возникновению более сложного класса распределенных систем с несколькими серверными узлами. Для создания таких систем оказывается недостаточным использования традиционных WWW-технологий, а требуются более сложные механизмы удаленного взаимодействия (RPC, RMI, CORBA, DCOM и др.).

С другой стороны, понятие "новые информационные технологии" обычно связывают с интеллектуализацией компьютерных приложений [17]. Отмечается растущий интерес к таким областям, как экспертные системы и интеллектуальные системы принятия решений, интеллектуальное тестирование, системы общения на естественном языке, интеллектуальные пользовательские интерфейсы и др.

Таким образом, особую актуальность приобретают вопросы применения технологий искусственного интеллекта в компьютерных сетях. Можно отметить два основных взаимосвязанных аспекта такого применения: **обмен знаниями** по сети (knowledge sharing) [99] и **совместное решение задач** (cooperative problem solving) [106] с применением распределенных по сети знаний.

Компьютерные сети предоставляют эффективные средства обмена сообщениями,

однако, в общем случае, проблема передачи знаний является более сложной. В простейшем случае, выбрав определенное представление знаний, можно представить некоторое множество знаний в виде сообщений и таким образом свести передачу знаний по сети к передаче соответствующих им сообщений в определенном представлении. Однако, в более общем случае возможно обмениваться знаниями путем более сложного взаимодействия распределенных программных компонентов, которые могут оперировать различными внутренними представлениями знаний. Такой подход будет включать в себя не только обмен знаниями на различных уровнях представления, но и распределенный механизм применения этих знаний для решения поставленной задачи.

Возможность обмена знаниями по сети является чрезвычайно привлекательной, так как позволяет концентрировать различные знания в различных узлах сети и затем комбинировать эти знания, используя их совместно для решения определенной задачи. Это позволит поднять содержательность информационных ресурсов сети на принципиально новый уровень, так как сетевые ресурсы будут содержать не слабоструктурированную текстовую информацию, а представленные определенным образом знания, пригодные для использования в процессе логического вывода для решения задач пользователей.

Примерами использования такого подхода могут быть: система интеллектуального поиска в Интернет, основанная на поддержании на каждом включенном в нее узле базы знаний по содержанию узла, более информативной, нежели список ключевых слов или контекстный поиск; система медицинской диагностики, когда в постановке диагноза участвует целый "виртуальный консилиум" баз знаний, подготовленных различными специалистами; обучающая система, функционирующая в рамках глобальной или Интранет-сети; интеллектуальная составляющая информационной системы поддержки деятельности виртуальной корпорации и т.д.

Помимо создания эффективных средств представления, структурирования и поиска информации (знаний) в компьютерных сетях, развитие теории интеллектуальных распределенных систем представляет особый интерес с точки зрения создания систем искусственного интеллекта нового поколения по принципу "перехода количества в качество".

В области распределенного искусственного интеллекта основное внимание исследователей сосредоточилось на **многоагентных системах** [80, 91], которые строятся из множества взаимодействующих агентов (зачастую представляющих собой полноценные интеллектуальные системы с символьным представлением знаний), совместно решающих поставленную задачу в распределенной среде. Для взаимодействия агентов различной природы разработаны специализированные языки обмена знаниями (KQML, KIF) [114, 115], а для обеспечения единого пространства знаний создаются **онтологии** [14, 76] эксплицитные спецификации концептуализации предметной области, как правило, в виде таксономии концептов и некоторого количества сопровождающих знаний (common knowledge) в виде системы аксиом-правил.

Однако, при построении многоагентной системы разработчики сталкиваются со многими трудностями: необходимостью создания оригинальной архитектуры системы (методологии построения распределенных агентных систем пока слабо развиты, учитывая разнообразие существующих в рамках агентного подхода направлений), проектирования онтологии предметной области, выбора инструментария и транспортной среды для взаимодействия агентов, а также выбором программных средств для реализации интеллектуального наполнения агента, и наконец сопряжением всех указанных компонентов в единую функционирующую систему.

При реализации интеллектуального наполнения агентов в большинстве случаев приходится либо разрабатывать внутреннее представление знаний и машину вывода "с нуля" на традиционных языках программирования высокого уровня (C++, Java, Python, Lisp) или языках искусственного интеллекта (Пролог), либо использовать существующую оболочку для создания интеллектуальных систем. На сегодняшний день существует не так много доступных оболочек с достаточно развитым программным интерфейсом: наибольшего внимания заслуживают CLIPS [75, 117], ее Java-аналог JESS [110] и система программирования AMZI Prolog[1] [113]. Система JESS используется во многих проектах, однако процесс ее интеграции в агентные системы и веб-приложения требует написания значительного количества программного кода "промежуточного" уровня, обеспечивающего в том числе преобразование знаний из внешнего, используемого в агентной системе, представления во внутреннее. Кроме того, CLIPS и JESS являются классическими продукционными системами прямого вывода, что несколько ограничивает возможности по управлению знаниями, а также требует представления знаний на языке класса OPS5, доступного в основном профессиональным инженерам по знаниям.

Следовательно, актуальной задачей является создание технологии и программного обеспечения, совмещающих в себе способы преодоления отмеченных проблем для определенного класса типовых задач. Можно выделить множество таких задач, связанных с **распределенным накоплением и использованием знаний** в компьютерных сетях, для решения которых классическая агентная архитектура представляется излишне общей. В то время как при построении агентных систем внимание как правило акцентируется на унифицированном внешнем представлении знаний с целью обмена между агентами различной природы, для некоторых задач представляется разумным базировать принципы распределения знаний на классической многоуровневой модели взаимодействия (в простейшем случае — модели клиент-сервер), в которой набор интеллектуальных систем или их составных компонентов взаимодействует и обменивается знаниями в некотором внутреннем представлении. Предлагаемая в работе новая архитектура **распределенной фреймовой иерархии** является одной из возможных реализаций такого подхода.

В то время как для агентных систем онтология представляет собой некоторое внешнее по отношению к агентам соглашение, в предлагаемом подходе сами компоненты системы задают структуру предметной области. Иерархическая комбинация фреймовых субиерархий по сути дела является одной из естественных реализаций таксономической онтологии, при этом распределенная фреймовая иерархия может естественным образом быть расширена продукционными правилами для задания динамики предметной области. При достаточно широком понимании понятия агента такая архитектура может быть отнесена к классу статических делиберативных коллаборативных агентов [91].

Немаловажную роль при построении реальных систем играет возможность взаимодействия с хранилищами структурированных данных (в первую очередь с реляционными и объектно-ориентированными базами данных и хранилищами слабоструктурированной информации на базе XML/HTML), а также интеграции в существующие информационные системы и веб-приложения. Представляет существенный интерес разработка такой модели представления знаний, которая обеспечивала бы естественную прозрачную интеграцию в единую модель реляционных структур данных, императивной и объектно-ориентированной парадигм программирования и компонентных моделей (COM,

[1] Строго говоря, система AMZI Prolog представляет собой реализацию языка логического программирования Пролог, но она может быть также отнесена к инструментам для построения экспертных систем, т.к. имеется практика такого использования (примеры которого содержатся на веб-сайте [113])

JavaBeans, CORBA-объекты и др.). Такая модель при достаточном богатстве представления знаний позволила бы в некотором роде объединить в себе идеологию **активных** [79] и **дедуктивных баз данных** [94] (безусловно, с некоторыми модификациями и потерей производительности) с возможностью распределения и удаленного использования знаний.

Таким образом, цель диссертационной работы заключается в разработке методов построения распределенных интеллектуальных систем на основе распределенной фреймовой иерархии, реализации на основе предложенной модели гибридного многоплатформенного инструментария JULIA, проектировании и реализации на основе инструментария препроцессора интеллектуальных веб-страниц IASP для интеграции интеллектуальных компонент в веб-приложения, разработке и реализации системы онтологического поиска в массивах аннотированных гипертекстовых документов, а также в применении инструментария при реализации реальных интеллектуальных и интеллектуально-информационных систем диагностики, контроля и дистанционного обучения.

В соответствии с поставленной целью в работе решаются следующие основные задачи:

- Исследование моделей представления знаний в интеллектуальных системах и формулирование продукционно-фреймовой модели представления знаний с кластеризацией продукционных правил прямого и обратного вывода относительно слотов в виде присоединенных процедур.

- Разработка модели распределенной фреймовой иерархии для распределенного накопления и использования знаний, соотнесение данной модели с существующими многоагентными системами, моделью распределенного вывода с общей доской объявлений, с онтологическими системами.

- Выделение характерных топологий и конфигураций в задачах распределенного накопления и использования знаний и методов представления таких конфигураций распределенной иерархией с множественным, псевдомножественным (поочередным) и мобильным наследованием.

- Разработка методов интеграции во фреймовую модель реляционных структур, объектно-ориентированных императивных включений и ООБД, компонентных моделей (COM, JavaBeans, CORBA).

- Проектирование и реализация на языке Java многоплатформенного гибридного инструментария JULIA (Java Universal Library for Intelligent Applications) с открытым программным интерфейсом, основанного на предложенных выше принципах, с использованием CORBA как протокола распределенного взаимодействия, а также специализированного языка представления знаний.

- Проектирование и реализация на базе инструментария препроцессора интеллектуальных веб-страниц IASP (Intelligent Active Server Pages), разработка различных конфигураций построения интеллектуальных веб-приложений на базе IASP.

- Проектирование и реализация на базе инструментария прототипа интеллектуальной поисковой системы **Jewel**, основанной на явном внедрении иерархии онтологических описаний в тексты HTML-страниц.

- Проектирование, разработка, реализация и внедрение реальных интеллектуальных систем на базе инструментария JULIA, включая интеллектуально-информационную систему диагностики больных заболеваниями предстательной железы, интеллектуальную систему дистанционного обучения логическому программированию, систему интеллектуального адаптивного тестирования знаний в области интернет-технологий, экспертную систему по продвижению интернет-ресурсов, систему моделирования управления производственным процессом.

В работе использовались понятия и методы математической логики, теории множеств, теории графов, элементы λ-исчисления, системного анализа, формальной семантики языков, а также современные методологии построения программных комплексов и систем, методы искусственного интеллекта и системного программирования.

Научная новизна результатов работы состоит в разработке оригинальной модели распределенной фреймовой иерархии для задач распределенного накопления и использования знаний, отличающейся высоким уровнем структурированности распределенных знаний вокруг неявно задаваемой таксономической онтологии предметной области. Предложенная модель позволяет сочетать элементы активных и дедуктивных баз данных, распределенного вывода с общей доской объявлений, языка интеллектуального скриптинга над программными компонентами в распределенной среде CORBA или Enterprise JavaBeans, а также традиционных продукционных баз знаний прямого и обратного вывода. Показано, что данная модель может эффективно использоваться в задачах с непересекающимися проблемными доменами, с совпадающими проблемными доменами, в задачах удаленной консультации, как интерактивной, так и автоматической (через программный интерфейс), которые могут расширять и дополнять удаленные знания локальными.

В ходе выполнения работы создано оригинальное программное обеспечение инструментария на языке Java, доведенное до уровня исследовательского прототипа. Общий объем разработанного программного обеспечения инструментария составляет около 13000 строк кода на языках Java, JavaCC и IDL. Инструментарий JULIA зарегистрирован в отделе регистрации программ ЭВМ, баз данных и топологий ИМС Федерального института промышленной собственности РОСПАТЕНТа (свидетельство №2002610609 от 25.04.2002, см. приложение С).

На основе инструментария реализовано и внедрено несколько интеллектуальных и интеллектуально-информационных систем, включающих в себя прототипы баз знаний на языке JFMDL, а также в ряде случаев дополнительные модули на языке Java. Предложен специализированный язык внедрения знаний в HTML-страницы, а также набор механизмов взаимодействия интеллектуальных страниц для построения комплексных интеллектуальных веб-приложений.

Практическая значимость работы состоит в разработке технологии создания распределенных интеллектуальных систем и веб-приложений в форме распределенной фреймовой иерархии, методологии проектирования таких систем и инструментария для практической реализации предложенной технологии в многоплатформенной гетерогенной среде. Разработанный инструментарий позволяет снизить трудозатраты на разработку интеллектуальных систем для обмена знаниями в сети, и открывает путь к созданию интеллектуальных приложений различной природы. Ценность разработки подтверждает внедрение инструментария в учебный процесс кафедры Вычислительной математики и программирования МАИ, как в виде базового средства обучения по поставленному автором

курсу "Экспертные системы", так и в составе интеллектуальной системы дистанционного обучения по курсу "Логическое программирования", а также внедрение основанной на инструментарии интеллектуально-учетной системы в лечебную практику отделения урологии ГКБ им. С.П.Боткина и прототипа системы диагностики и управления процессами рафинирования и дезодорирования растительных масел в ООО "Сид-Ойл".

Результаты работы докладывались и обсуждались на 7-ой и 9-ой международных школах-семинарах "Новые информационные технологии" (Судак, 1999 г., 2001 г.), на I и III международных конференциях "Computer Science and Information Technology Workshop" (Москва, 1999 г., Уфа, 2001 г.), были представлены на 10-ой конференции по вычислительной механике и современным прикладным программным системам (Переславль-Залесский, 1999 г.), на II международной конференции "Computer Science and Information Technology Workshop" (Уфа, 2000 г.). Работа прошла обсуждение на семинаре кафедры Вычислительной математики и программирования МАИ, на спецсеминаре лаборатории искусственного интеллекта факультета компьютерных наук и информатики университета Любляны и отдела искусственного интеллекта института Йозефа Стефана под руководством проф. И. Братко (материалы семинара, включая видеозапись выступления, можно найти в Интернет по адресу `http://solomon.ijs.si/seminarji/seminar.asp?id=44`), была рассмотрена в отделе прикладных интеллектуальных систем ВЦ РАН. Разработанная интеллектуально-учетная система для диагностики больных в области урологии была представлена на ряде медицинских научно-практических конференций.

По теме диссертации опубликовано 13 работ, а также 6 работ в медицинских изданиях по результатам внедрения.

Диссертация состоит из введения, 3-х глав, заключения, списка литературы и 3-х приложений, а также электронного приложения на компакт-диске. Общий объем работы 195 страниц, в том числе основной текст 142 страницы, 37 рисунков, 2 таблицы, список литературы из 125 наименований. Электронное приложение содержит несколько ограниченную по функциональности свободно распространяемую версию инструмепария, примеры и прототипы реальных экспертных и интеллектуальных систем, разработанных на базе инструментария, а также электронную версию текста диссертации, автореферата и избранных публикаций.

1. Инструментарий для построения гибридных интеллектуальных систем на основе фреймового представления знаний

При создании интеллектуальных и интеллектуально-информационных систем разработчик каждый раз сталкивается с необходимостью использовать те или иные концепции или программные средства для реализации интеллектуальных функций. В данной главе дается обзор технологий построения интеллектуальных систем, подробнее рассматривается класс интеллектуальных систем с эксплицитным представлением знаний, в особенности продукционно-фреймовые системы. Дается формальная математическая модель семантики языка представления знаний и процесса комбинированного логического вывода при одном из подходов к построению продукционно-фреймовой системы. На основе этого подхода разрабатывается открытая архитектура гибридного инструментария в виде библиотеки классов на языке Java, которая может использоваться в широком классе программных комплексов как самостоятельная оболочка экспертных систем, так и в качестве основы для создания приложений с интегрированной интеллектуальной функциональностью. Инструментарий выгодно отличается от аналогов прозрачной семантикой, поддержкой прямого и обратного логического вывода, высокой степенью расширяемости и интеграции с другими компонентами программных комплексов, как то реляционные базы данных и элементы императивного программного кода в виде программных компонентов JavaBeans.

1.1. Обзор современных технологий построения интеллектуальных систем

Современный уровень развития информационных технологий требует все более эффективных способов решения на ЭВМ различных задач автоматизации человеческой деятельности. В то время как для ряда задач существуют явные алгоритмы решения (например, задачи вычислительной математики) либо развитые методологии построения программных систем (например, классические информационные системы основанные на моделировании данных и процессов предметной области), существует целый класс задач, обычно называемых **интеллектуальными**, в которых решение базируется на той или иной формализации процесса решения аналогичных задач человеком и использовании программной реализации соответствующей модели.

Интеллектуальные методы в настоящее время пытаются применять не только в "традиционных" областях (диагностика, управление, поддержка принятия решений и др.), но и в решении таких задач, как поиск релевантных документов (в частности, улучшение качества поиска в Интернет), дистанционное обучение, активные хранилища данных и др. В связи с этим возникает особая потребность в создании, с одной стороны, универсальных моделей для интеграции систем, основанных на знаниях, с программными системами, основанными на других моделях вычислений (классические информационные системы и базы данных, распределенные системы, вычислительные алгоритмы и др.),

а также в программной реализации соответствующих средств гибридного типа. Обеспечение эффективной технологии использования интеллектуальных методов в как можно более широком классе задач является одним из направлений развития современного искусственного интеллекта.

1.1.1. Основные подходы к построению систем, основанных на знаниях

Существует множество подходов к определению понятия **интеллектуальной системы**. На теоретическом [56] и общефилософском [1] уровне ответ на вопрос о том, что же такое интеллектуальность, дает тест Тьюринга — мысленный или реальный эксперимент по символьному диалогу наблюдателя с тестируемой на интеллектуальность системой с одной стороны и с реальным человеком с другой, в ходе которого наблюдателю предлагается определить, кто же из его собеседников является человеком, а кто искусственной системой. Таким образом, определение понятия интеллектуальности производится путем сравнение *некоторых*[1] свойств системы со свойствами человека.

Тест Тьюринга не утверждает, что интеллектуальное поведение может быть достигнуто только путем моделирования человеческого способа рассуждений. Поскольку, однако, человек является по сути единственным примером подлинно интеллектуальной системы, то исследования в рамках искусственного интеллекта (ИИ) сосредоточились именно на моделировании человеческого поведения. Различают [14] два основных направления: нейрокибернетический или восходящий подход, состоящий в моделировании низкоуровневых процессов в мозге на нейронном уровне, и символьный или нисходящий подход, основанный на понимании высокоуровневых процессов мышления и принципов рассуждений. В рамках обоих подходов в настоящее время достигнут существенный прогресс, позволяющий успешно решать определенные классы задач.

В данной работе мы будем рассматривать исключительно символьный подход к решению задач. Таким образом мы будем предполагать, что интеллектуальная система основана на некоторой (математической или психологической) модели человеческих рассуждений и, соответственно, на некотором способе представления знаний человека.

Еще Аристотель [2] предположил, что интеллектуальные возможности человека основываются на способности к **символьным рассуждениям**, т.е. к рассуждениям, основанным на некоторой **формальной аксиоматической системе**. В такой системе отдельные атомарные понятия, которыми оперирует человек в процессе мышления, обозначаются некоторыми атомами, на основе атомов и сложных структур (термов) строятся формулы, к которым применяются определенные правила вывода для получения из исходных посылок более сложных формул. Логика Аристотеля оперировала сравнительно простыми правилами вывода (силлогизмами), однако даже на их основе удается описать целый класс свойственных человеку рассуждений.

С философской точки зрения сложно однозначно сказать, имеют ли все рассуждения символьный характер[2]. Известно, что в процессе рассуждения человек оперирует

[1]Противники теста Тьюринга не без основания считают, что ограничение диалога только символьными сообщениями сужает возможности по передаче информации (в данном контексте интерес представляет также лингвистический характер общения в свете гипотезы Сапира-Ворфа [3, 105], а также характер взаимосвязи информации и сообщения, изучаемый в основах информатики [4, 15]). Кроме того, компьютерной системе приходится "скрывать" от наблюдателя некоторые свои "области превосходства", как то, например, способность к быстрым вычислениям.

[2]В пользу этого утверждения говорит гипотеза, выдвинутая А.Ньюэллом и Г.Саймоном, согласно

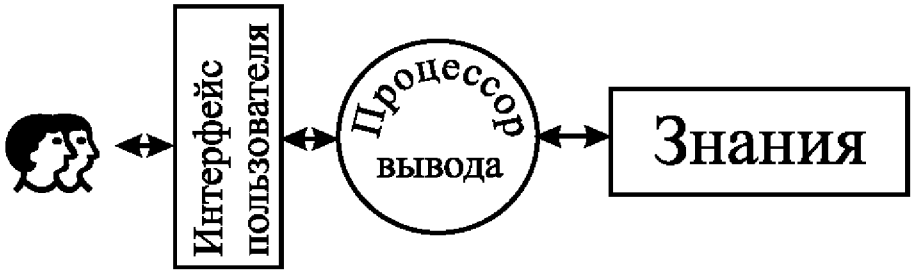


Рис. 1.1. Основные компоненты простейшей символьной интеллектуальной системы

нестрогими способами вывода, такими, как индукция или абдукция, которые плохо поддаются формализации [55]. Даже попытки построить основания математики на формальной логике оказываются неуспешными. Однако, формальная логика и символьные рассуждения в более общем случае являются практически единственным способом эффективного представления и использования знаний в интеллектуальных системах[3].

Таким образом, интеллектуальная система включает в себя, как минимум, следующие компоненты (см. рис. 1.1):

- Процессор логического вывода, осуществляющий манипуляцию знаниями в символьном представлении и координирующий работу системы.
- Базу знаний, содержащую знания о классе решаемых задач в том или ином представлении, а также знания о конкретной решаемой задаче.
- Пользовательский интерфейс (или интерфейс сопряжения интеллектуальной системы с другими программными компонентами).

1.1.2. Обзор различных представлений знаний

Термин **знания**, используемый в предыдущем разделе, нуждается в некотором уточнении. Как известно [4], любая процедура автоматической обработки информации оперирует данными, т.е. символьным представлением информации с некоторой функцией интерпретации $\phi : \mathcal{N} \to \mathcal{I}$. Однако то, что мы обычно называем знаниями[4], напрямую информацией не является[5], и, следовательно, для представления знаний в символьном виде необходимо построить некоторый способ преобразования знаний в информацию $\psi : \mathcal{K} \to \mathcal{I}$, чтобы затем представлять знания к символьном виде при помощи композиции $\psi \circ \phi^{-1}$.

Знания, представленные в виде сообщений, обычно обладают следующими свойствами, которые можно также принять за определение знаний [25]:

которой поведение любой интеллектуальной системы может быть смоделировано формальной аксиоматической системой [40].

[3]Упомянутый ранее нейрокибернетический подход не содержит представленных в явном виде знаний

[4]Словари [58, 109] обычно определяют знания как "нечто, полученное в результате познания", либо более детально как "совокупность понятий предметной области и связей между ними, полученный в результате опыта и практической деятельности".

[5]Например, специалист в некоторой области медицины обладает знаниями, позволяющими ему точно диагностировать пациентов, однако попытка представить эти знания в символьной форме (в письменном виде) или даже передать студенту представляют значительные трудности

- Внутренняя интерпретируемость
- Структурированность
- Связность
- Семантическая метрика
- Активность

Помимо широких экспрессиональных возможностей, способ представления знаний в интеллектуальных системах должен допускать эффективные алгоритмы обработки знаний и их применения для решения практических задач.

Обычно выделяют четыре группы способов представления знаний [88]:

Логическое представление, основанное на логике предикатов, обычно первого порядка. Наиболее распространенным представителем этого семейства является язык логического программирования Пролог [26], а также его многочисленные расширения.

Сетевое представление, при котором множество знаний представляется в виде графа, в вершинах которого расположены объекты предметной области (концепты), а дугами являются различные отношения между ними (наследования, включения и др.) Вывод в сетевом представлении обычно основан на различных алгоритмах поиска фрагмента сети по образцу, на применении свойств отношений (например, транзитивности отношения наследования) для вывода новых фрагментов знаний.

Иерархическое представление, основанное на построении иерархии реальных и абстрактных понятий связанных отношением наследования[6]. К иерархическому представлению относится фреймовое представление, сценарии и др.

Процедурное представление, при котором знания кодируются набором элементарных действий, применение которых ведет к решению задачи. Последовательность этих действий может быть строго определенной (**алгоритмическое представление**), или же определяться динамически в процессе решения по некоторым эвристическим критериям (**продукционное представление**).

Наибольшее распространение получило продукционное представление знаний [44], в котором знания представляются множеством правил перехода из одного состояния задачи в другое $\alpha \to \beta$, называемых **продукциями**. Продукционные системы предполагают использование некоторого представления знаний для описания статической картины предметной области S, а также некоторого механизма сопоставления по образцу левой части продукции с текущим состоянием $s \in S$.

Таким образом, продукционная система определяет, возможно бесконечный, граф состояний G, в узлах которого расположены состояния предметной области $s_i \in S$, а дуги задаются продукционными правилами. Процесс вывода в такой системе представляет собой поиск по графу состояний, который может вестись от некоторого начального состояния к целевому (т.н. **прямой вывод** или вывод, управляемый данными), или же от некоторой гипотезы к ее подтверждению (**обратный вывод** или вывод, управляемый

[6] или, в более общем случае, другим таксономическим отношением

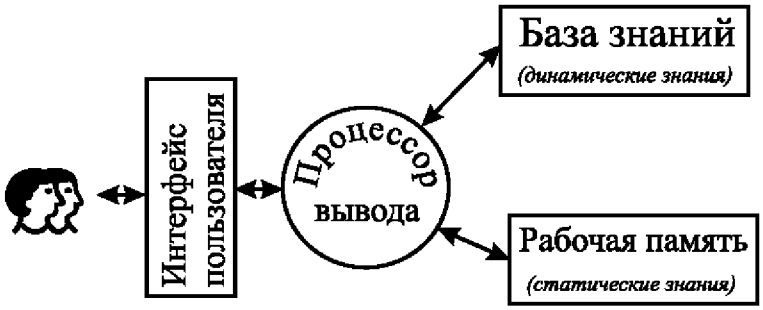


Рис. 1.2. Структура простейшей продукционной системы

целями)[7]. В каждом из алгоритмов вывода могут использоваться различные алгоритмы поиска пути в графе: поиск в глубину, поиск в ширину, поиск по принципу первый-лучший и др. Из-за большой вычислительной сложности, как правило, используют эвристический поиск в глубину (с возвратами или без), в котором применяются различные эвристики для определения стратегии выбора применяемой на каждом шаге продукции.

Частным случаем продукционной системы являются **нормальные алгоритмы Маркова** (НАМ) [36], в которых состояния определяются строками в некотором алфавите A^*, а сопоставление левой части продукции с состоянием представляет собой поиск подстроки. Продукция таким образом играет роль замены одной подстроки на другую. Из теории алгоритмов известно, что алгоритмическая модель, определяемая НАМ, эквивалентна другим алгоритмическим моделям (например, машине Тьюринга [15]) и понятию эффективной вычислимости [9].

Классические продукционные системы [75] используют в качестве состояния множество пар атрибут значение или трок объект атрибут значение. Такое представление знаний является по сути дела линеаризованной записью семантической сети [42,58] и допускает наиболее естественную интерпретацию продукций в виде правил ЕСЛИ *выражение* ТО *операция*, где *выражение* — произвольное логическое выражение относительно значений атрибутов текущего состояния, а *операция* является либо операцией присваивания значения атрибуту, либо операцией удаления атрибута из текущего состояния[8].

Таким образом знания $\mathcal{K}$, с которыми оперирует продукционная система, можно разделить на **динамические** $\mathcal{K}_d$ (продукции), представляющие динамику переходов между состояниями, и **статические** $\mathcal{K}_s$, описывающие текущее состояние решаемой в данный момент задачи. Структура простейшей продукционной системы приведена на рис.1.2.

Для более удобного описания сложных предметных областей модели объект атрибут значение оказывается недостаточно. Одним из основных средств борьбы со сложностью является **абстракция** [10], проявлением которой является построение в предметной области иерархии абстрактных понятий, связанных отношением наследования свойств. Для минимизации объема хранимых знаний человек выделяет некоторые общие свойства объектов предметной области и характеризует эти свойства принадлежностью объектов к некоторым классам[9].

---

[7]В этом случае текущая вершина графа при поиске аналогична "дырке" в теории проводимости: она содержит не полученное на данный момент подтвержденное фактами состояние задачи, а "желаемое" состояние, факты для подтверждения которого требуется получить

[8]В некоторых системах с монотонным рассуждением операция удаления не предусматривается.

[9]Психологические эксперименты, в ходе которых измерялось время реакции испытуемых на те или иные вопросы о свойствах реальных или абстрактных объектов, подтвердили, что человек действительно

Теория фреймов была впервые предложена Марвином Минским в работе [37], как попытка объяснить механизм мышления человека применительно к различным задачам (зрительному восприятию, пониманию естественного языка и др.). В данной работе не описывается детальная структура фрейма, а лишь обозначаются основные характеристики систем фреймов и операций над ними. **Фрейм** понимается как некоторая структура, представляющая собой модель стереотипной ситуации или класса объектов. Когда человек сталкивается с новым объектом или ситуацией, он пытается сопоставить ее с существующей системой фреймов, таким образом определяя место данной ситуации в системе понятий. При этом сопоставлении данные о ситуации могут уточняться в процессе дальнейшего наблюдения или логического вывода, в частности, приводящего к активизации других связанных фреймов. Таким образом, основными операциями в системе фреймов являются **сопоставление** и **активизация** фрейма или его составляющих.

В дальнейшем при реализации фреймовых систем понятие фрейма уточнялось тем или иным способом. Наиболее распространен подход [14, 58], при котором под фреймом понимается структура, состоящая из набора **слотов**. Каждый слот представляет собой шаблон для хранения значения или набора значений определенного типа и обычно содержит следующие элементы:

- **Текущее значение слота** — значение/список значений слота, или неопределенность ($\perp$)
- **Значение слота по умолчанию**
- Присоединенная процедура для определения значение слота (**процедура-запрос**)
- Присоединенная процедура (набор процедур), срабатывающая при присваивании или изменении значения слота (**процедура-демон**).
- Ограничения на значения слота: по типу данных, фасетные ограничения, ограничения в виде произвольных логических выражений.

1.2. Обзор существующих фреймовых систем

В области создания прикладных интеллектуальных систем наблюдается ситуация, когда создается большое количество достаточно сложных специализированных программных средств, каждое из которых находит узкое применение для решения очень небольшого числа практических задач (иногда — одной задачи) и не получает широкого распространения. За последние 20-30 лет было создано более 50 систем на основе фреймового представления знаний, однако лишь очень небольшое число из них являются широко известными и используются для решения задач за пределами организации-разработчика. В работе [81] содержится сравнительный обзор принципов построения таких систем, на которых мы лишь кратко остановимся.

Большинство фреймовых систем используют приведенное выше трактование понятия фрейма как набора активных слотов. В большинстве систем функции поддержки фреймового представления знаний отделяют от процесса логического вывода: в этом случае логический вывод и соответствующие динамические знания рассматриваются как

хранит знания о классах объектов на как можно более высоком уровне иерархии [55].

"надстройка" над базовой функциональностью поддержания структурной модели предметной области, которая обеспечивается фреймовой системой.

Процесс логического вывода может быть основан на продукционных правилах (например, в системах KEE, CLASS, PROTEUS, THEO), на процессе классификации (в системах класса KL-ONE) либо на комбинации обоих подходов (LOOM, CLASSIC). Продукционные правила могут в свою очередь представляться фреймами определенного класса (KEE, CLASS), либо быть выражениями базового языка программирования (например, в системе THEO слоты могут содержать произвольные Пролог-выражения, запускаемые при вычислении значения слота). Некоторые системы могут включать в себя компоненты поддержания истинности (KEE, CYCL, LOOM и др.), контекстов для обеспечения параллельного вывода по нескольким возможным направлениям (KEE, THEO, STROBE и др.), механизма ограничений (constraints) и др.

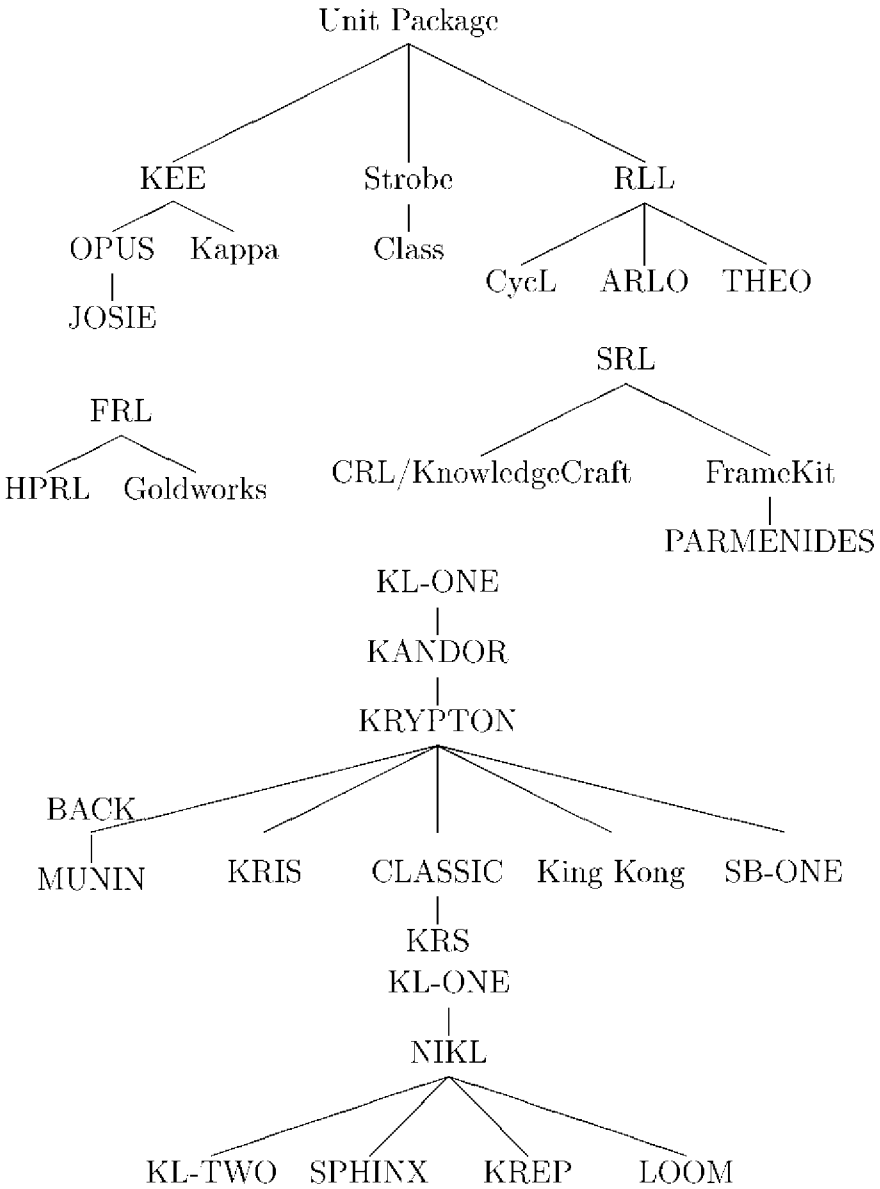


Рис. 1.3. Различные семейства фреймовых систем

Среди наиболее известных фреймовых систем промышленного масштаба следует отметить ART и KEE. Обе эти системы основаны на сочетании фреймового представле-

ния знаний с продукционными правилами, однако в KEE упор делается на фреймовую составляющую (в частности, правила рассматриваются как фреймы), а в ART на продукционные правила, а в качестве фреймообразующей составляющей вводятся схемы и образцы. В обоих системах имеется компонент поддержания истинности (Truth Maintenance System), графический интерфейс пользователя и средства разработки. Среди более распространенных коммерческих систем следует отметить G2 фирмы Gensym Corp., которая позиционируется как система реального времени, способная к одновременному слежению за большим количеством изменяющихся во времени объектов на основе различных методов рассуждения.

Помимо чисто фреймовых систем имеются продукционные системы, использующие ту или иную модификацию фреймового или объектного формализма для представления знаний, совместно с классическим продукционным механизмом логического вывода. Среди этого класса систем следует отметить CLIPS [75, 117] и ее Java-аналог Jess [110], которые получили весьма широкое распространение при создании разнообразных интеллектуальных программных комплексов. Эти системы используют классическую продукционную модель прямого вывода, основанную на алгоритме Rete [74], таким образом обеспечивая высокую производительность при значительной гибкости языка и семантики выполнения. Для представления знаний используется объектно-ориентированный язык COOL, реализующий концепции шаблонов, ограничений, наследования и др., характерные для фреймовых систем.

С точки зрения реализации, большинство фреймовых систем представляют собой надстройки над тем или иным языком программирования высокого уровня, в роли которых обычно выступают Лисп (KEE) или Пролог (THEO). Продукционно-фреймовые и продукционно-объектные системы прямого вывода (CLIPS, JESS) обычно реализуются на традиционных языках программирования (C/C++ или Java).

Значительный интерес представляет отечественная система FRAME/2 [98], разработанная в Рязанском Радиотехническом институте совместно с ВЦ РАН и используемая в составе PiES Workbench [61, 82]. FRAME/2 использует преобразование продукционно-фреймовой модели в программу на C++, которая затем может использоваться для создания законченных и встраиваемых интеллектуальных систем.

На основе FRAME/2 в ВЦ РАН реализована полноценная WorkBench система с языком представления знаний PILOT/2 [14, 61, 82]. Отличительной особенностью системы является наличие ярко выраженной инференциальной составляющей базы знаний помимо традиционной декларативной и процедурной, что по мнению специалистов необходимо для достаточно богатого описания знаний. Язык содержит набор конструкций для описания стратегии выбора правил на основе многоуровневых фильтров, позволяющих учитывать предварительный отбор правил на основании флага активности и секционных разрешений, проверку левых частей продукций в отобранном множестве и затем дополнительный отбор правил на основе правиловых разрешений. В заключении из полученного конфликтного множества выбор одного правила производится случайным образом.

Другой интересной особенностью PiES WorkBench является возможность определения правил-демонов и присоединенных процедур на базовом языке C++ или Java. Таким образом оказывается удобным специфицировать структуру предметной области в виде фреймовой иерархии и затем связывать с некоторыми из объектов достаточно сложную императивную логику, реализация которой посредством правил менее эффективна или вообще невозможна. В качестве примера такого подхода в [14, 84] рассмотрен интеллек-

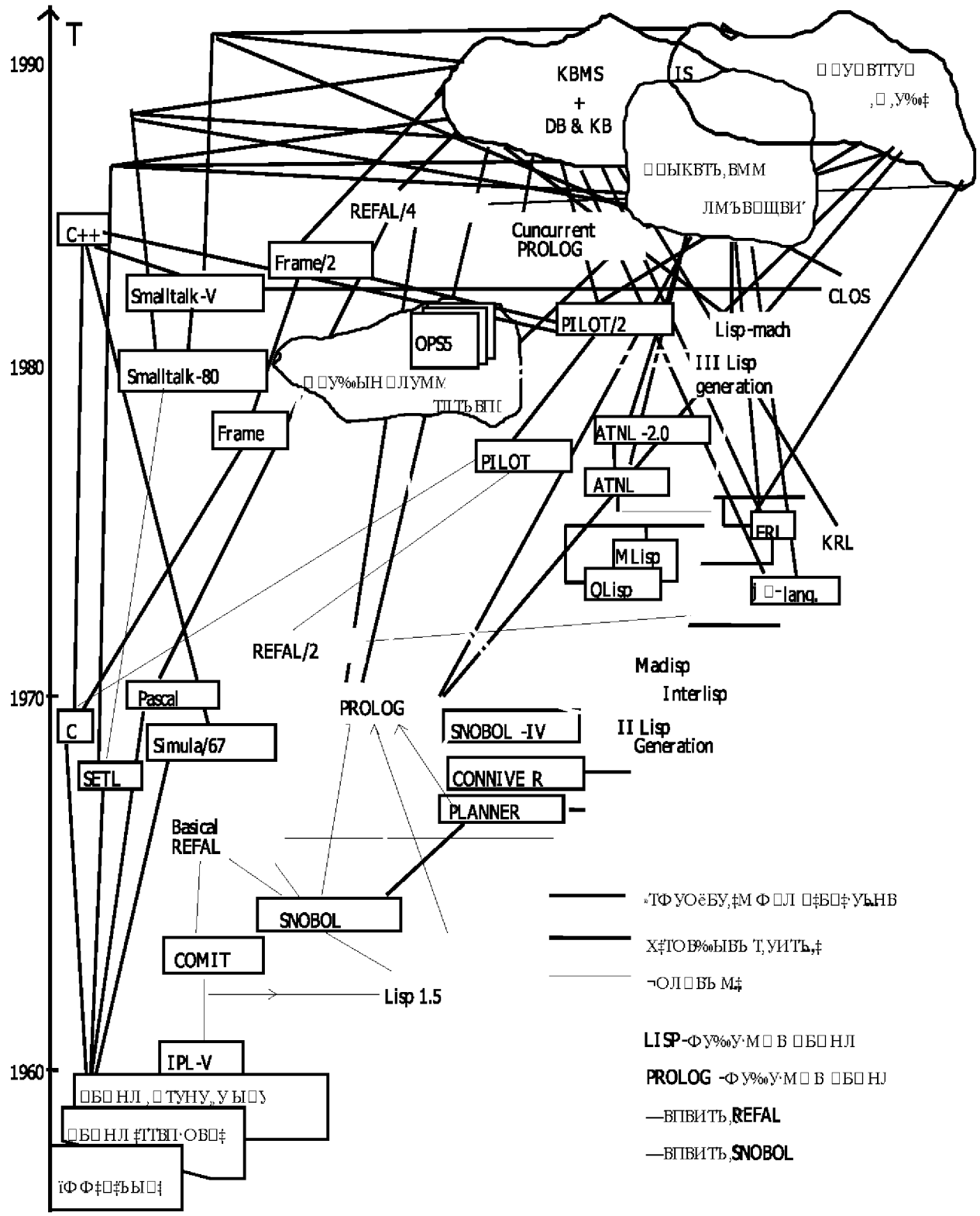


Рис. 1.4. Высокоуровневые системы манипулирования знаниями в контексте языков программирования (заимствовано из [83])

туальный HTML-конвертор.

1.3. Математическая модель продукционно-фреймовой системы с прямым и обратным выводом

1.3.1. Формализация представления знаний

Как было отмечено выше (см. раздел 1.1.2), знания $\mathcal{K}$ в интеллектуальной системе можно разделить на статические $\mathcal{K}_s$, служащие для представления состояния предметной области в некоторый момент времени, и динамические $\mathcal{K}_d$, описывающие множество возможных правил перехода по графу состояний, вершинами которого являются элементы $\mathcal{K}_s$. В классической продукционной системе $\mathcal{K}_d$ описывается множеством продукционных правил, образующих базу знаний, а $\mathcal{K}_s$ представляет собой множество атрибутов, которое формируется и пополняется в процессе решения конкретной задачи.

Рис. 1.5. Различные типы знаний в продукционно-фреймовой системе

В предлагаемой модели продукционно-фреймовой системы ситуация обстоит несколько сложнее. Предметная область при фреймовом представлении подвергается таксономической декомпозиции на множество представляемых фреймами концептов $\mathcal{F}$ при помощи отношения наследования :. Эта декомпозиция предметной области может быть отнесена к статическим знаниям и в зависимости от сложности вводимой семантики может меняться либо оставаться постоянной в процессе логического вывода. В большинстве систем множество фреймов описывается в базе знаний и остается более или менее постоянным[10] в процессе решения, однако следует отличать такие знания (назовем их структурными $\mathcal{K}_{ss}$) от множества атрибутов задачи, аналогичных рабочей памяти и представляемых значениями слотов всех фреймов (назовем соответствующее множество $\mathcal{K}_{sv}$, или состоянием рабочей памяти). Таким образом, база знаний (являющаяся плодом разработки инженеров по знаниям и задаваемая в некотором внешнем представлении на специализированном языке до начала работы системы) состоит из структурной $\mathcal{K}_{ss_1}$ и процедурной (или динамической) $\mathcal{K}_d$ составляющих; рабочая память состоит из множества значений слотов $\mathcal{K}_{sv}$ и, возможно, некоторой структурной составляющей $\mathcal{K}_{ss_2}$. Взаимосвязь рассмотренных понятий показана на рис. 1.5

1.3.1.1. Представление статических знаний

Статические знания во фреймовой системе, представленные в основном составляющей $\mathcal{K}_{sv}$, образованы множеством значений слотов всех фреймов. Предположим, что слоты могут принимать значения из некоторого множества $\mathbb{T}$, определяемого в соответствии с некоторой системой типов (см. раздел 1.3.2).

[10]То, что в процессе вывода в модель добавляются новые фреймы, или же некоторый фрейм подвергается процессу классификации является весьма частой ситуацией; однако основное дерево концептов предметной области остается постоянным, и таксономические связи в нем в процессе вывода не изменяются (за исключением рассмотренного ниже псевдомножественного наследования).

Для описания фреймовой структуры введем функцию состояния $\mathcal{W} : I \to \mathcal{F}$, которая отображает множество идентификаторов I в множество фреймов $\mathcal{F}$. Каждый фрейм $f \in \mathcal{F}$ представляет собой функцию $f : I_f \to \mathcal{S}$, отображающую множество идентификаторов слотов данного фрейма в множество слотов. Структура слотов может быть различной в зависимости от сложности рассматриваемой модели; в простейшем случае можно предположить $\mathcal{S} \equiv \mathrm{expr}(\mathbb{T})$, где под $\mathrm{expr}(\mathbb{T}) \subset \Lambda(\mathbb{T})$ подразумевается множество λ-выражений над системой типов $\mathbb{T}$, включающих в себя, возможно, ссылки на другие слоты.

1.3.1.2. Представление динамических знаний

В более сложном случае имеет смысл рассматривать слот как кортеж, состоящий из нескольких компонент, представляющих собой текущее значение слота и значение по умолчанию, процедуры-демоны и процедуры запросы, ограничения и т.д. Таким образом можно будет совмещать в рамках одной функции $\mathcal{W}$ как собственно состояние системы, характеризуемое множеством значений слотов, так и правила, управляющие процессом смены состояний. Это позволит, в принципе, определять семантику с динамической модификацией множества правил и/или других характеристик логического вывода (например, стратегий выбора правил) в процессе вывода. В данной работе мы не будем останавливаться на такого рода семантиках, так как их строгое математическое изложение сопряжено с определенными трудностями и может составить предмет отдельного исследования.

Мы будем рассматривать множество слотов вида $\mathcal{S} = \{\langle v, u, \{Q_i\}, \{D_j\}, \{C_k\}, \sqsubseteq_q, \sqsubseteq_d, \alpha\rangle\}$, где:

- $v \in \mathbb{T}$ — текущее значение слота или $\perp$ в случае, если значение не определено
- $u \in \mathbb{T}$ — значение слота по умолчанию
- $\{Q_i\}$ — множество присоединенных к слоту процедур-демонов. Каждая процедура может быть произвольным выражением из некоторого множества выражений $\mathbb{E}$, структура которого будет рассмотрена более подробно ниже в разделе 1.3.4.
- $\{D_j\}$ — множество процедур-демонов, срабатывающих при присваивании слоту некоторого значения. В соответствии с описанной ниже в разделе 1.3.4 семантикой, процедуры-демоны $D_j : \mathbb{E} \times \mathbb{W} \to \mathbb{W}$ применяются к функции состояния $\mathcal{W} \in \mathbb{W}$ в случае истинности некоторого выражения, и порождают тем самым новое состояние системы.
- $\{C_k\}$ — множество ограничений на значение слота, сформулированных в виде выражений-предикатов $C_k \in \mathbb{E}$.
- $\sqsubseteq_d, \sqsubseteq_q$ — линейные порядки на множествах $\{D_j\}$ и $\{Q_i\}$ соответственно, определяющие порядок применения соответствующих процедур в процессе вывода.
- $\alpha \in \{\mathbf{true}, \mathbf{false}\}$ — флаг, указывающий на участие слота в процессе рекуррентного обратного вывода и служащий для предотвращения бесконечного зацикливания.

Для доступа к элементам $s = \langle u, v, Q, D, C, \sqsubseteq_q, \sqsubseteq_d, \alpha\rangle$ будем, по аналогии с [72], использовать обозначения $s.value, s.defvalue, s.rules, s.daemons, s.constraints, s.rss_q, s.rss_d$

и $s.busy$ соответственно. Аналогично, будем обозначать $f.s = \langle f, s\rangle \in \mathbb{I}, f \in I, s \in I_f$ для именования фрейма, а для доступа к слоту писать $\mathcal{W}.f.s = \mathcal{W}(f, s)$.

Для дальнейших построений нам необходимо ввести функцию присваивания значения слоту $write : \mathbb{I} \times \mathbb{T} \to \mathbb{W} \to \mathbb{W}$, формирующую новое состояние, которое мы будем обозначать как $\mathcal{W}[f.s \leftarrow v] = write(\langle f, s\rangle, v, \mathcal{W})$. Эта функция может быть определена следующим образом:

$$\mathcal{W}[f.s \leftarrow x] = \lambda f_1 \lambda s_1.(\langle f_1, s_1\rangle = \langle f, s\rangle \to \mathcal{W}(f, s)[1 \leftarrow x], \mathcal{W}(f_1, s_1)),$$

где $(b \to u, v)$ — операция условного вычисления, а через $s[n \leftarrow x]$ обозначена функция замены n-го компонента кортежа s на x: $(s[n \leftarrow x])_i = (i = n) \to x, s_i$. Аналогично будем определять операцию $\mathcal{W}[f.s.busy \leftarrow \textbf{true/false}]$ для присвоения логического значения компоненте $s.busy$.

Определим также операцию разности между состояниями $|\cdot| : \mathbb{W} \times \mathbb{W} \to \mathcal{P}(\mathbb{I})$, возвращающую список идентификаторов слотов, значения которых отличаются в двух данных состояниях:

$$\forall \langle f, s\rangle \in \mathbb{I} \quad \langle f, s\rangle \in |\mathcal{W}_1 - \mathcal{W}_2| \iff \mathcal{W}_1(f, s) \neq \mathcal{W}_2(f, s) \tag{1.1}$$

1.3.2. Система типов

Система типов $\mathbb{T}$, использованная выше при описании представления статических знаний, грубо говоря представляет собой множество значений, которые могут принимать слоты фреймов. Возможны различные подходы к описанию системы типов: например, можно рассматривать систему **со строгой типизацией**, в которой соответствие типов будет каждый раз проверяться при присваивании и вычислении выражений, или же систему **с динамическим преобразованием типов**, в которой разрешены операции над значениями различной структуры — при этом одни и те же операции могут нести различную семантическую нагрузку (т.е. будет иметь место *полиморфизм операций*).

Нам оказывается удобнее рассматривать именно систему с динамическим приведением типов[11], или бестиповую систему наподобие классического λ-исчисления [5]. При описании семантики логического вывода конкретное множество используемых типов данных и операций не имеет большого значения, поэтому допускается значительная свобода в отношении используемой системы типизации. Мы будем предполагать, что система типов удовлетворяет свойствам, сформулированным в следующем определении:

Определение 1.1. Под **системой типов** будем понимать совокупность $\mathcal{T} = \langle \mathbb{T}, \mathbb{O}\rangle$ из **множества значений** $\mathbb{T}$ **и множества операций** $\mathbb{O}$, таких, что:

- $\mathbb{T}$ является решеткой с отношением порядка $\sqsubseteq$
- $\exists \bot \in \mathbb{T} : \forall x \in \mathbb{T} \quad \bot \sqsubseteq x$
- $\forall \circledast \in \mathbb{O}, \forall u, v \in \mathbb{T} \quad u \circledast v \in \mathbb{T}$
- $\forall \circledast \in \mathbb{O}, \forall u \in \mathbb{T} \quad \bot \circledast u = u \circledast \bot = \bot$

[11] На самом деле ранее мы уже воспользовались свойством динамики в разделе 1.3.1.1 при определении операции write.

Кроме того, потребуем, чтобы на множестве значений были определены функции $\mathrm{isTrue} : \mathbb{T} \to \mathbb{B} = \{\mathbf{true}, \mathbf{false}\}$, преобразующие значения из множества значений к логическому типу.

Определим также условную операцию $\cdot \to_{\mathbb{T}} \cdot, \cdot : \mathbb{T}^3 \to \mathbb{T}$ следующим образом: $b \to_{\mathbb{T}} u, v =_{\mathrm{def}} \mathrm{isTrue}(b) \to u, v$.

Такое понятие системы типов по существу является достаточно общим. Нам будет удобно вводить систему типов на основе понятия доменов, как это делается в [12] в этом случае составные элементы системы типов наделяются дополнительными полезными свойствами с точки зрения теории вычислений, в частности, автоматически можно гарантировать, что множества непрерывных в смысле отношения порядка функций $[D_1 \to D_2]$ будут также образовывать решетку. Не останавливаясь на подробностях, кратко рассмотрим одну из возможных систем типов, которая в дальнейшем ляжет в основу практической реализации рассматриваемой модели.

Будем рассматривать следующие основные типы данных:

- $\mathbb{B} = \{\mathbf{true}, \mathbf{false}\}$ — логический тип.
- $\mathbb{N}$ — множество натуральных чисел. При необходимости можно также рассматривать множество представимых в ЭВМ действительных чисел $\mathbb{D} \subset \mathbb{R}$, однако в целях упрощения изложения мы это опустим.
- $\mathbb{S} = A^*$ — множество строк в некотором базовом алфавите A.
- $\mathbb{R}$ — множество ссылок. Семантика множества ссылок задается рассматриваемой ниже вычислительной моделью, а их структура — представлением статических знаний. В нашем случае $\mathbb{R} = \mathbb{R}_1 + \mathbb{R}_2$, $\mathbb{R}_1 = I$, $\mathbb{R}_2 = \langle f, s \rangle$, $f \in I, s \in I_f$.
- $\mathbb{L}$ — множество списков конечной длины из элементов $\mathbb{T}$. Это множество можно определять по-разному:
  - как отображение из $\mathbb{N}$ в множество λ-выражений, представляющих функции высшего порядка.
  - как дерево [4, 7], представляющее собой структуру вложенных подсписков.
  - теоретико-множественным способом как $\mathbb{L} = \cup_{k=1}^{\infty} \mathbb{L}^{(k)}$, где $\mathbb{L}^{(k)}$ — множество списков, длина подсписков которых не превосходит k, $\mathbb{L}^{(k)} = \cup_{n=0}^{\infty} \mathbb{L}_n^{(k)}$, $\mathbb{L}_n^{(k)} = \langle a_1, \ldots, a_n \rangle, \quad a_i \in V \cup \mathbb{L}^{(k-1)}$, где $V = \mathbb{B} + \mathbb{N} + \mathbb{S} + \mathbb{R}$ [12]

Нас не будут интересовать подробности определения понятия список, лишь бы для него была определена операция принадлежности элемента $\mathbf{in} : \mathbb{T} \times \mathbb{L} \to \mathbb{B}$. Для последнего случая эта операция вводится обычным теоретико-множественным способом. В дальнейшем для удовлетворения определения 1.1 следует распространить эту операцию на все множество типов $\mathbb{T}$ следующим стандартным образом:

[12] Множество списков можно также определять "наивным" рекурсивным способом как $\mathbb{L} = \cup_{n=1}^{\infty} \mathbb{L}_n$, где $\mathbb{L}_n = \langle a_1, \ldots, a_n \rangle, a_i \in \mathbb{T}$. Такое опеределение, совместно с определением множества $\mathbb{T}$, на самом деле определяет систему уравнений относительно множеств, решение которой может быть найдено как неподвижная точка соответствующего множественного отображения. При этом решение будет получаться как предел многократного применения этого отображения и будет совпадать с приведенным выше определением (см. также [7]).

$$u\,\mathbf{in}_{\sqsupseteq}\,v = \begin{cases} u\,\mathbf{in}_{\mathbb{L}}\,v, u \in \mathbb{T}, v \in \mathbb{L} \\ \bot, \text{в противном случае} \end{cases} \tag{1.2}$$

Само множество $\mathbb{T}$ в данном случае образуется как дизъюнктная сумма $\mathbb{T} = \{\bot\} + \mathbb{B} + \mathbb{N} + \mathbb{S} + \mathbb{R} + \mathbb{L}$ [12].

Для полного определения системы типов остается также определить множество стандартных операций ($\{+, -, *, /\}$) и распространить их на все множество $\mathbb{T}$ способом, приведенным выше в формуле (1.2). Это можно сделать, например, введя набор операций приведения типов $\mathbf{to}(\mathbb{T}_1, \mathbb{T}_2) : \mathbb{T} \to \mathbb{T}_2$, и затем приводя операнды выражения к одному типу. Например, операция + (обозначающая арифметическое сложение или конкатенацию в зависимости от типового контекста) может быть введена следующим образом:

$$u +_{\mathbb{T}} v = \begin{cases} \bot, & u = \bot \text{ или } v = \bot \\ u +_{\mathbb{N}} v, & u \in \mathbb{N}, v \in \mathbb{N} \\ u \& v, & u, v \in \mathbb{S} \text{ или } u, v \in \mathbb{L} \\ u \& \mathbf{toList}(v), & u \in \mathbb{L}, v \in V \\ \mathbf{toList}(u) \& v, & u \in \mathbb{N}, v \in L \\ \mathbf{toStr}(u) \& v, & u \in \mathbb{N}, v \in \mathbb{S} \\ u \& \mathbf{toStr}(v), & u \in \mathbb{S}, v \in \mathbb{N} \\ \bot, & \text{в противном случае} \end{cases}$$

Здесь через & обозначена операция конкатенации (строк или списков), $\mathbf{toList}, \mathbf{toStr}$ операции приведения типов. Заметим, что в зависимости от определения операции могут быть неассоциативными или даже некоммутативными.

Отметим, что введенное таким образом множество $\mathbb{T}$ действительно является решеткой с минимальным элементом. Действительно, на всех вводимых подтипах естественным образом определено отношение порядка, которое распространяется и на дизъюнктную сумму[13], при этом, так как выделенный элемент $\bot$ входит в дизъюнктную сумму отдельным слагаемым, то для него будет выполнено свойство минимальности. При необходимости можно также добавить в дизъюнктную сумму отдельным последним слагаемым максимальный элемент $\top$, который в этом случае будет удовлетворять условию $\forall x \in \mathbb{T} \quad x \sqsubseteq \top$.

1.3.3. Свойства состояний фреймовой модели

1.3.3.1. Множество и фактор-множество состояний

Функция состояния $\mathcal{W}$ описывает не только текущее состояние в процессе логического вывода, но также и множество правил, которое мы в данной работе будем полагать постоянным. Таким образом, возможно выделить функцию $\mathcal{W} : \mathbb{I} \to \mathbb{T} : \quad \mathcal{W}(f, s) = \mathcal{W}(f, s).value \; \forall f \in I, s \in I_f$, которая будет характеризовать чисто статическую составляющую состояния системы, а также комплементарную к ней функцию, которая в свою очередь будет постоянной в процессе логического вывода.

Множество состояний системы $\mathfrak{W}$ можно представить себе в виде бесконечного графа, вершинами которого будут различные состояния $\mathcal{W} \in \mathfrak{W}$, а дуги будут задаваться правилами логического вывода. Бесконечность графа будет, в первую очередь, вызвана

[13] Для дизъюнктной суммы $D_1 + \ldots + D_n$ будем полагать $\langle x_1, d_1 \rangle \sqsubseteq \langle x_2, d_2 \rangle \iff (d_1 < d_2) \vee ((d_1 = d_2) \wedge (x_1 \sqsubseteq x_2))$

потенциальной бесконечностью (в теоретическом плане — континуальностью) множества значений ($\mathbb{T}$) каждого из слотов. Однако, говоря практически, число различимых состояний в каждой базе знаний будет конечно, так как во множестве посылок всех правил базы знаний содержится конечное число сравнений.

Для формализации этого понятия введем в рассмотрение отношение эквивалентности $\cong$, при котором $\mathcal{W}_1 \cong \mathcal{W}_2 \iff \mathcal{W}_1$ и $\mathcal{W}_2$ неразличимы с точки зрения базы знаний, т.е. для всего множества посылок C в левых частях правил базы знаний $\|c\|_{\mathcal{W}_1} = \|c\|_{\mathcal{W}_2}\ \forall c \in C$, где через $\|c\|_{\mathcal{W}}$ обозначено значение посылки c в состоянии $\mathcal{W}$. Тогда фактор-пространство $\mathfrak{W}_{\cong} = \mathfrak{W}/\cong$ пространства $\mathfrak{W}$ относительно отношения $\cong$ будет обладать следующим свойством:

Утверждение 1.1. *Фактор-пространство $\mathfrak{W}_{\cong}$ содержит конечное число элементов.*

Доказательство. Фактор-пространство $\mathfrak{W}_{\cong}$ можно строить следующим образом. Пусть $c \in C$ — посылка некоторого правила, которая может принимать истинное или ложное значение на состоянии $\mathcal{W}$. Тогда она разбивает множество состояний на два подмножества: $\mathfrak{W}_t(c) = \{\mathcal{W} \in \mathfrak{W} |\ \|c\|_{\mathcal{W}} = \mathbf{true}\}$ и $\mathfrak{W}_f(c) = \{\mathcal{W} \in \mathfrak{W} |\ \|c\|_{\mathcal{W}} = \mathbf{false}\}$.

Пусть множество всех посылок C (конечное) имеет вид $C = \{c_1, \ldots, c_n\}$. Будем строить фактор-множество $\mathfrak{W}_{\cong}$ следующим образом: положим $\mathfrak{W}_{\cong}{}^{(1)} = \{\mathfrak{W}_f(c_1), \mathfrak{W}_t(c_1)\}$. Далее определим

$$\mathfrak{W}_{\cong}{}^{(k)} = \bigcup_{x \in \mathfrak{W}_{\cong}{}^{(k-1)}} [(x \cap \mathfrak{W}_f(c_k)) \cup (x \cap \mathfrak{W}_t(c_k))] \tag{1.3}$$

Тогда $\mathfrak{W}_{\cong} = \mathfrak{W}_{\cong}{}^{(n)}$ и будет искомым фактор-пространством. Действительно, для произвольного множества $A \in \mathfrak{W}_{\cong}$, если $\mathcal{W}_1, \mathcal{W}_2 \in A$, то по построению $\forall c \in C\ \|c\|_{\mathcal{W}_1} = \|c\|_{\mathcal{W}_2}$, т.е. $\mathcal{W}_1 \cong \mathcal{W}_2$. □

Очевидно, что каждый задаваемый правилом переход в пространстве $\mathfrak{W}$ индуцирует соответствующий переход в пространстве $\mathfrak{W}_{\cong}$. Таким образом, исследование процесса вывода в системе можно свести к исследованию конечного графа состояний с конечным множеством переходов.

1.3.3.2. Определение отношения порядка на множестве состояний

На множестве состояний $\mathfrak{W}$ может быть определено естественное отношение порядка $\sqsubseteq$ следующим образом:

$$\mathcal{W}_1 \sqsubseteq \mathcal{W}_2 \iff \forall \langle f, s \rangle \in \mathbb{I} \quad \mathcal{W}_1(f, s) \sqsubseteq \mathcal{W}_2(f, s) \tag{1.4}$$

где во втором случае отношение порядка связывает элементы множества системы типов $\mathbb{T}$. То, что такое определение действительно задаст отношение порядка, можно легко убедиться стандартными для теории решеток рассуждениями.

Очевидно, что если не рассматривать метаправила, модифицирующие динамическую часть базы знаний в процессе вывода, то заданное таким образом отношение порядка также индуцирует отношение порядка на множестве $\mathbb{W}$. Кроме того, введенное формулой 1.4 отношение порядка требует постоянства множества $\mathbb{I}$ на протяжении вывода, т.е. имена всех фреймов и слотов должны быть известны заранее, до начала вывода. Эти

ограничения безусловно сужают класс рассматриваемых динамических интеллектуальных систем, и изучение более богатой семантики представляет значительный интерес для дальнейших исследований[14].

1.3.3.3. Отношение и иерархия наследования

В системе фреймов вводится отношение наследования ":" между фреймами, определяющее иерархию понятий предметной области и соответственно используемое для заимствования свойств (правил) от родительских фреймов. Различают [10] одиночное и множественное наследование: в первом случае отношение : является функцией по правому аргументу, т.е. одному дочернему фрейму может соответствовать только один родитель[15]; во втором случае такое ограничение отсутствует. Кроме того, различают статическое наследование (используемое, например, в формализме F-логики [85]), при котором отношение наследования остается постоянным в процессе логического вывода, и динамическое наследование, когда конфигурация иерархии наследования в процессе вывода изменяется.

Будем рассматривать отношение наследования, индуцированное значением слотов `parent` дочерних фреймов, т.е. $F : G \iff \|F.parent\| = G$, где через $\|\cdot\|$ обозначена операция вычисления значения слота, которая будет уточняться далее в соответствии с определяемой семантикой. Более того, если допустить значения слота *parent* некоторого спискового типа, то возможно определить отношение множественного наследования как $F : G \iff \|F.parent\| \ni G$. Определенное таким образом отношение наследования будет динамическим, так как значение слота будет вычисляться в соответствии с правилами определенной ниже семантики, т.е. будет возможно определение родителя посредством продукционных правил, либо операции спецификации.

1.3.4. Операционная семантика логического вывода

Для определения семантики процесса вывода в системе воспользуемся, аналогично [12], отображением $\mathcal{E} : \mathbb{E} \times \mathbb{W} \times \mathbb{C} \to \mathbb{T} \times \mathbb{W}$, которое вычисляет значение произвольного выражения $E \in \mathbb{E}$ в некотором состоянии $\mathcal{W}$ и контексте $\mathbb{C}$, возвращая полученное значение $v \in \mathbb{T}$ и новое состояние $\mathcal{W}'$. Будем также использовать более удобное обозначение $v = \|E\|^{C}_{\mathcal{W}\to\mathcal{W}'} \iff \mathcal{E}(E, \mathcal{W}, C) = (v, \mathcal{W}')$.

Семантика логического вывода подразумевает, что при вычислении значения выражения может быть инициирован вывод значений тех слотов, для которых значение не известно заранее и не было получено ранее в процессе вывода. Определим также отношение $\mathcal{E}' : \mathbb{E} \times \mathbb{W} \times \mathbb{C} \to \mathbb{T} \times \mathbb{W}$, аналогичное $\mathcal{E}$, но не инициирующее процесс вывода. Для этого отображения будем использовать обозначение $v = \|E\|'^{C}_{\mathcal{W}\to\mathcal{W}'} \iff \mathcal{E}'(E, \mathcal{W}, C) = (v, \mathcal{W}')$. В дальнейшем, если это не оговорено отдельно, все приведенные для $\mathcal{E}$ результаты будут аналогичным образом формулироваться для $\mathcal{E}'$.

Понятие контекста вычисления будет необходимо для описания семантики наследования, чтобы корректно применять правила для фрейма-родителя к значениям в дочерних фреймах. В нашем случае достаточно будет положить $\mathbb{C} = I$, хотя в более слож-

[14] Действительно, считается, что способность использовать мета-правила и "рассуждения о рассуждениях" является одной из существенных особенностей интеллектуальных систем и необходимым свойством для семиотического моделирования человеческой интеллектуальности. Исследование математических свойств мета-теорий и семантики мета-правил таким образом может пролить свет на такие вопросы, как самосознание, самовосприятие и др.

[15] Следует заметить, что в этом случае иерархия наследования представляет собой *дерево* [4, 7].

ных моделях в понятие контекста может потребоваться включить и другие конструкции. Фрейм, передаваемый через контекст, будем называть *базовым фреймом*: он будет использоваться при вычислении выражения вместо зарезервированного идентификатора *this*. Так как в большинстве случаев при описании семантики вычисления выражений значение базового фрейма передается в функции вычисления подвыражений без изменения, то мы будем верхний индекс C опускать. Таким образом, если контекст вычисления не указан в некотором равенстве, то имеется в виду, что в обоих частях равенства фигурирует один и тот же контекст.

1.3.4.1. Синтаксис множества выражений

Для корректного определения семантики необходимо сначала задать формальный синтаксис множества выражений $\mathbb{E}$. Для этого используем несколько менее формальный аналог нормальной формы Бэкуса-Науэра (БНФ). Полное же описание синтаксиса выражений, используемого в реально-разработанном по данной модели языке представления знаний содержится в приложении.

Таким образом, выражением $E \in \mathbb{E}$ является:

- Константа из множества типов $\mathbb{T}$.
- Ссылка на слот вида $f.s,\ f \in I \cup \{this\},\ s \in I_f$.
- Арифметическая операция вида $E_1 \circledast E_2$, где $\circledast \in \{+, -, *, /\}$.
- Логическая операция $E_1 \boxdot E_2$, где $\boxdot \in \{\mathbf{or}, \mathbf{and}\}$, или $\mathbf{not}\ E_1$.
- Условная операция $E_0 \to E_1, E_2$ или $E_0 \to E_1$.
- Операция вызова функции-оракула $\mathcal{A}(d),\ d \in \mathbb{D}$ (см. раздел 1.3.4.3).

Строго говоря, арифметические и логические операции в множестве выражений $\mathbb{E}$ могут вводиться в соответствии с операциями, определенными на множестве типов. Приведенная в следущем разделе семантика вычисления выражений сопоставляет каждой операции в синтаксическом множестве $\mathbb{E}$ некоторый денотат, основанный на определении операции в множестве $\mathbb{T}$.

1.3.4.2. Интерпретация констант, арифметических и логических выражений

Весьма очевидно, что интерпретация констант не зависит от состояния $\mathcal{W}$, и в качестве денотата константы возвращается само ее значение, т.е. $\forall x \in \mathbb{T}\ \|x\|_{\mathcal{W}\to\mathcal{W}} = x$.

Значением выражения $E = E_1 \circledast E_2$, являющегося арифметической операцией над парой выражений E_1 и E_2, будет соответствующая арифметическая операция $\circledast_{\mathbb{T}}$ в множестве типов $\mathbb{T}$. Таким образом:

$$\|E_1 \circledast E_2\|_{\mathcal{W}_1\to\mathcal{W}_2} = \begin{cases} \bot, \|E_1\|_{\mathcal{W}_1\to\mathcal{W}_2} = \bot \\ \|E_1\|_{\mathcal{W}_1\to\mathcal{W}'} \circledast_{\mathbb{T}} \|E_2\|_{\mathcal{W}'\to\mathcal{W}_2} \end{cases}$$

В данном описании учитывается т.н. *укороченное* вычисление выражения, при котором в случае, если первое подвыражение возвращает $\bot$, значение второго не вычисляется. Возможен также более простой подход, когда оба подвыражения вычисляются в любом случае:

$$\|E_1 \circledast E_2\|_{\mathcal{W}_1 \to \mathcal{W}_2} = \|E_1\|_{\mathcal{W}_1 \to \mathcal{W}'} \circledast_\square \|E_2\|_{\mathcal{W}' \to \mathcal{W}_2}$$

В случае с вычислением логических выражений вычисление производится аналогично, так как в динамической системе типов явное различие между логическим и другими типами не проводится. Как правило, при вычислении логических выражений *всегда* используется укороченная схема. Аналогично по укороченной схеме вычисляется условное выражение $\to$:

$$\|E \to E_1, E_2\|_{\mathcal{W}_1 \to \mathcal{W}_2} = \begin{cases} \|E_1\|_{\mathcal{W}' \to \mathcal{W}_2}, & \mathrm{isTrue}(\|E\|_{\mathcal{W}_1 \to \mathcal{W}'}) \\ \|E_2\|_{\mathcal{W}' \to \mathcal{W}_2}, & \neg\,\mathrm{isTrue}(\|E\|_{\mathcal{W}_1 \to \mathcal{W}'}) \end{cases}$$

1.3.4.3. Получение значений слотов и обратный вывод

В случае, если выражение E имеет вид $f.s$, для получения его значения требуется извлечь значение соответствующего слота из состояния $\mathcal{W}$, в случае необходимости применив обратный вывод. Интерпретация при помощи функции $\mathcal{E}'$ не инициирует обратный вывод, а лишь возвращает значение слота или $\perp$ (состояние в этом случае не изменяется):

$$\|f.s\|'_{\mathcal{W} \to \mathcal{W}} = \mathcal{W}(f, s).value$$

Применение обратного вывода для вычисления значения слота s состоит в последовательном (в соответствии с линейным порядком $\sqsubseteq_q$) применении правил из $s.rules$. Применение правила состоит в вычислении соответствующего этому правилу условного или безусловного выражения в соответствии с таблицей 1.1.

Продукционное правило	Выражение
`IF` E `THEN` $f.s = F$	$E \to F$
`SET` $f.s = E$	E
`ASK` q $f.s$	$\mathcal{A}(q)$

Таблица 1.1. Соответствие продукционных правил присоединенным выражениям

Пусть $\langle Q, \sqsubseteq \rangle$ (конечное) линейно упорядоченное множество. Обозначим $\hat{Q} = Q \setminus \{\inf Q\}$. Введем функцию $\mu_{\mathcal{W} \to \mathcal{W}'}(Q)$ последовательного вычисления упорядоченного семейства выражений Q в начальном состоянии $\mathcal{W}$:

$$\mu_{\mathcal{W} \to \mathcal{W}'}(Q) = \begin{cases} \|\inf Q\|_{\mathcal{W} \to \mathcal{W}'}, & \|\inf Q\|_{\mathcal{W} \to \mathcal{W}'} \neq \perp \\ \mu_{\mathcal{W}'' \to \mathcal{W}'}(\hat{Q}), & \|\inf Q\|_{\mathcal{W} \to \mathcal{W}''} = \perp \\ \perp, & Q = \varnothing \quad (\mathcal{W}' = \mathcal{W}) \end{cases} \tag{1.5}$$

Утверждение 1.2. *Данное рекурсивное определение действительно корректно определяет функцию μ, которая определена для любого начального состояния и конечного множества Q.*

Идея доказательства. Для доказательства существования функции $\mu : \mathbb{Q} \times \mathbb{W} \to \mathbb{T} \times \mathbb{W}$ введем в рассмотрение некоторую функцию $\mu' : \mathbb{Q} \times \mathbb{W} \to \mathbb{Q} \times \mathbb{W}$, такую, что $\mu_{\mathcal{W} \to \mathcal{W}'}(Q) = \|\inf Q'\|_{\mathcal{W}' \to \mathcal{W}'}$, где $Q' = \mu'_{\mathcal{W} \to \mathcal{W}'}(Q)$ (считая, что $\|\perp\| = \perp$). Определим μ' как неподвижную точку некоторой функции $\mathcal{M} : [\mathbb{Q} \times \mathbb{W} \to \mathbb{Q} \times \mathbb{W}] \to [\mathbb{Q} \times \mathbb{W} \to \mathbb{Q} \times \mathbb{W}]$ более высокого порядка:

$$\mathcal{M} = \lambda f.\lambda(Q, W).\begin{cases} \langle Q, \mathcal{W}' \rangle, & \|\inf Q\|_{\mathcal{W}\to\mathcal{W}'} \neq \bot \\ f_{\mathcal{W}''\to\mathcal{W}'}(\hat{Q}), & \|\inf Q\|_{\mathcal{W}\to\mathcal{W}''} = \bot \\ \langle \varnothing, \mathcal{W} \rangle, & Q = \varnothing \end{cases} \tag{1.6}$$

Следует отметить, что $\mathcal{P}(Q)$ является решеткой относительно операции включения, а на множестве $\mathbb{W}$ также индуцируется отношение порядка (см. раздел 1.3.3.2) таким образом, что оно является решеткой, на котором функция $\|\cdot\|_{\to}$. будет монотонной (см. утверждение 1.8). В соответствии с этим у отображения $\mathcal{M}$ будет существовать неподвижная точка $\mu' = \operatorname{fix}\mathcal{M}$, которая и будет определять искомую функцию μ'.

При этом сама функция μ' будет монотонной на решетке, порожденной $\mathbb{Q}\times\mathbb{W}$ с рассмотренными выше отношениями порядка, и по теореме Тарского о неподвижной точке [5, 7, 102] она будет определять конечное состояние $\langle Q', \mathcal{W}' \rangle = \bigsqcup_{n=1}^{\infty} \mu'^n(Q, \mathcal{W})$. Следует отметить, что ввиду конечности исходного множества Q и строгого характера монотонности данное равенство будет выполняться для некоторого конечного N, т.е.

$$\forall Q \in \mathbb{Q}\ \exists N \leq \#Q:\ \langle Q', \mathcal{W}' \rangle = \bigsqcup_{n=1}^{N} \mu'^n(Q, \mathcal{W})$$

□

При помощи определенной таким образом функции μ можно легко определить функцию $\|\cdot\|$ для слота с учетом обратного вывода:

$$\|f.s\|_{\mathcal{W}\to\mathcal{W}'} = \begin{cases} \mathcal{W}(f,s).value, \mathcal{W}(f,s).value \neq \bot \text{ (при этом } \mathcal{W}' = \mathcal{W}) \\ \bot, \mathcal{W}(f.s).busy = \mathbf{true} \\ x = \mu_{\mathcal{W}[f.s.busy\leftarrow\mathbf{true}]\to\mathcal{W}''}(\langle \mathcal{W}(f,s).rules, \mathcal{W}(f,s).\sqsubseteq_q \rangle), \\ \qquad \mathcal{W}' = \mathcal{W}''[f.s \leftarrow x, f.s.busy \leftarrow \mathbf{false}] \quad (x \neq \bot) \\ y = \mu_{\mathcal{W}'''\to\mathcal{W}''''}(\{p.s | p \in \|f.parent\|_{\mathcal{W}''\to\mathcal{W}'''}\}), \\ \qquad \mathcal{W}' = \mathcal{W}''''[f.s \leftarrow y, f.s.busy \leftarrow \mathbf{false}] \quad (y \neq \bot, x = \bot) \\ \bot, \text{ в противном случае} \end{cases} \tag{1.7}$$

Следует отметить, что приведенная здесь семантика не учитывает возможность применения метаправил, осуществляющих смену стратегии выбора правил в процессе вывода, а также динамическое изменение множества самих правил. Рассмотрение такой возможности привело бы нас к существенным затруднениям, так как при доказательстве утверждения 1.2 было использовано постоянство и конечность множества Q и отношения порядка на нем. Также, вычисление множества родительских фреймов производилось до осуществления процесса вывода в первом родительском фрейме, что не учитывает возможность изменения этого множества в процессе дальнейшего вывода. Рассмотрение более богатой семантики и ее математических свойств безусловно представляет существенный интерес, однако выходит за рамки данной работы (см. также сноску на стр. 30).

Следует также отдельно определить вычисление значения выражения $this.s$ следующим образом:

$$\|this.s\|^{f}_{\mathcal{W}\to\mathcal{W}'} = \|f.s\|^{f}_{\mathcal{W}\to\mathcal{W}'}$$

а также определение вычисления выражения с нулевым контекстом:

$$\|f.s\|^{\perp}_{\mathcal{W}\to\mathcal{W}'} = \|f.s\|^{f}_{\mathcal{W}\to\mathcal{W}'}$$

Для описания возможности динамического получения значений извне в ходе обратного логического вывода можно воспользоваться двумя подходами: введением множества пар "вопрос-ответ" в контекст вычислений [12], либо **функцией-оракулом** (oracle function) [11,97]. В последнем случае мы определяем некоторую внешнюю функцию взаимодействия со средой $\mathcal{A} : \mathbb{D} \to \mathbb{T}$, определенную на некотором множестве дескрипторов $\mathbb{D}$, и соответствующий класс выражений, для которого операция $\|\cdot\|$ вводится следующим образом:

$$\|\mathcal{A}(d)\|_{\mathcal{W}\to\mathcal{W}} = \mathcal{A}(d) \tag{1.8}$$

Использование такого подхода более естественно выражает характер диалога, управляемого процессом вывода (вызовя функции-оракула происходят по мере необходимости в том порядке, в котором вопросы задаются пользователю), однако затрудняет изучение семантики, делая ее недетерминированной (т.е. результат зависит от неформализованного в модели поведения внешних функций). Введение множества ответов пользователя в контекст вывода приводит к детерминированной семантике, которая не отражает динамику задаваемых вопросов (в этом случае приходится фиксировать все множество "ответов" заранее). Заметим, что разновидностью такого подхода является полный отказ от операции задания вопроса и замена ее первоначальным заполнением значениями всех известных слотов.

1.3.4.4. Прямой вывод

Для формализации прямого вывода существует несколько подходов. В классическом алгоритме прямого вывода [75] на каждом шаге формируется конфликтное множество применимых правил, из которых затем применяется одно, что приводит к изменению состояния. В случае монотонного вывода функция применения одного правила будет монотонной, что приводит к весьма естественному описанию семантики с помощью неподвижной точки на основании теоремы Тарского. Похожий подход применительно к логическому программированию описан в [73]. Однако такой "классический" подход подразумевает использование всего множества правил на каждом шаге вывода, и не учитывает привязанность правил к слотам фреймов.

Другой крайностью являлась бы попытка математической формализации используемых алгоритмов прямого вывода семейства Rete ([74,78,90]). Такая формализация по всей видимости оказалась бы излишне сложной, а также слишком привязанной к конкретному алгоритму логического вывода.

Мы используем некоторый промежуточный подход, в котором правила прямого вывода будут присоединены к слотам фреймов как это было описано в разделе 1.3.1.2. Таким образом, при присваивании слоту значения будут срабатывать только правила, связанные с этим слотом, т.е. в левой части которых фигурирует значение этого слота. Применению правил будет соответствовать изменение состояния в соответствии с некоторой функцией. Ниже мы покажем, что такой подход, при правильном построении сети демонов, в некотором смысле является эквивалентным классическому, но при этом он непосредственно соответствует возможному алгоритму реализации прямого вывода.

Каждое присоединенное правило-демон имеет вид $D = \langle E, A \rangle$, где $E \in \mathbb{E}$ — условная часть правила, представляющая собой произвольное выражение, $A : \mathbb{W} \to \mathbb{W}$ — некоторая функция изменения состояния, которая, как правило, будет представлять собой функцию изменения значения слота либо композицию таких функций. В некоторых случаях будет полезно также параметризовать эти функции текущим контекстом вычисления $C \in \mathbb{C}$, тогда $A : \mathbb{C} \to \mathbb{W} \to \mathbb{W}$, или $A(C) : \mathbb{W} \to \mathbb{W}$. Имеет смысл рассматривать композиции следующих элементарных функций:

$$\begin{array}{lcl} [f.s \leftarrow v](C, \mathcal{W}) & = & \mathcal{W}[f.s \leftarrow v] \\ [this.s \leftarrow v](C, \mathcal{W}) & = & \mathcal{W}[C.s \leftarrow v] \end{array} \tag{1.9}$$

Для каждой функции A введем в рассмотрение множество модифицируемых слотов $\Delta(A)$. Очевидно, что $\Delta([f.s \leftarrow v]) = \{f.s\}$, и $\Delta(A_1 \circ \ldots \circ A_n) = \Delta(\{A_1, \ldots, A_n\}) = \cup_{i=1}^{n} \Delta(A_i)$.

Для описания процесса прямого вывода нам понадобится понятие функции активации $\alpha : \mathbb{I} \to \{free, todo, applied\}$, где

$$\alpha(f, s) = \begin{cases} free, & \text{если слот не принимает участие в прямом выводе} \\ todo, & \text{если демоны для слота еще нужно применить} \\ applied, & \text{если демоны для слота уже были применены} \end{cases} \tag{1.10}$$

Множество функций активации обозначим через $\mathcal{A}$.

На множестве $\{free, todo, applied\}$ определим отношение порядка следующим образом:

$$free \sqsubseteq todo \sqsubseteq applied \tag{1.11}$$

Тогда

$$\alpha_1 \sqsubseteq \alpha_2 \iff \forall \langle f, s \rangle \in \mathbb{I} \quad \alpha_1(f, s) \sqsubseteq \alpha_2(f, s) \tag{1.12}$$

Для формализации прямого вывода введем функцию $\Phi^C : \mathcal{A} \times \mathbb{W} \to \mathcal{A} \times \mathbb{W}$, которая по функции активации и состоянию применяет правила-демоны для этого состояния, и помечает в функции активации те слоты, которые были изменены в процессе обработки, а также одноименные слоты для родителей данного фрейма.

Обозначим через $\alpha[f.s] = \lambda\langle f_x, s_x \rangle.(\langle f, s \rangle = \langle f_x, s_x \rangle) \to (todo \vee \alpha(f, s)); \alpha(f_x, s_x)$. Индуктивно можно определить $\alpha[\{f_i.s_i\}_{i=1}^{n}] = \alpha[f_1.s_1] \ldots [f_n.s_n]$.

Для применения множества правил $S \subset \mathbb{D}$ к состоянию определим функцию $\phi_{\sqsubseteq}^{C} : \mathbb{D} \times \mathcal{A} \times \mathbb{W} \to \mathbb{D} \times \mathcal{A} \times \mathbb{W}$ следующим образом[16]:

$$\phi^C(S, \alpha, \mathcal{W}) = \begin{cases} \langle \varnothing, \alpha, \mathcal{W} \rangle, \text{ если } S = \varnothing \\ \langle \hat{S}, \alpha[\inf S], \mathrm{isTrue}\, \|E\|'_{\mathcal{W} \to \mathcal{W}} \to A(C)(\mathcal{W}); \mathcal{W} \rangle \\ \qquad (\text{где } \inf S = \langle E, A \rangle) \end{cases} \tag{1.13}$$

Лемма 1.1. $\forall \alpha \in \mathcal{A} \quad \alpha[f.s] \sqsupseteq \alpha$

Лемма 1.2. $\forall \langle E, A \rangle \in \mathbb{D}, \forall \mathcal{W} \quad A(C)(\mathcal{W}) \sqsupseteq \mathcal{W}$

Следствие 1.3. *Функция ϕ^C является монотонной относительно решетки $\mathbb{D} \times \mathcal{A} \times \mathbb{W}$, с отношением порядка, порожденным естественным отношением $\subseteq$ на $\mathbb{D}$, описанным формулой 1.12 на $\mathcal{A}$, и введенным в разделе 1.3.3.2 отношением порядка на $\mathbb{W}$.*

[16] В данном равенстве операции inf и $\hat{\cdot}$ понимаются в смысле отношения $\sqsubseteq$.

Утверждение 1.4. *Существует функция* $\varphi : \mathbb{C} \times \mathbb{D} \times \mathcal{A} \times \mathbb{W} \to \mathcal{A} \times \mathbb{W}$, *которая по начальному состоянию* $\mathcal{W}$ *и функции активации* α *применяет множество правил* $\{D_i\}_{i=1}^N$ *в контексте* C, *возвращая конечное состояние* $\mathcal{W}'$ *и функцию активации* α', *причем* $\mathcal{W} \sqsubseteq \mathcal{W}'$, $\alpha \sqsubseteq \alpha'$, *т.е.* φ *как функция* α *и* $\mathcal{W}$ *будет монотонной.*

Доказательство. Из следствия 1.3 вытекает, что по теореме Тарского [102] функция ϕ имеет неподвижную точку, которая может быть получена как $\langle S', \alpha', \mathcal{W}' \rangle = \bigsqcup_{i=1}^{\infty} \phi^i(S, \alpha, \mathcal{W})$. Легко видно, что ввиду строгой монотонности по первому аргументу данное равенство принимает вид

$$\bigsqcup_{i=1}^{N} \phi^i(S, \alpha, \mathcal{W}) = \langle \varnothing, \alpha', \mathcal{W}' \rangle \tag{1.14}$$

Можно таким образом определить функцию φ, положив $\varphi^C(S, \alpha, \mathcal{W}) = \langle \alpha', \mathcal{W}' \rangle$. Условия на упорядоченность начального и конечного состояний следует из монотонности ϕ. □

Определим функцию $\mathrm{Anc}_{\mathcal{W} \to \mathcal{W}'}(f)$, возвращающую множество родительских фреймов для f (при этом динамическое вычисление родителя требует предусмотреть возможность изменения состояния $\mathcal{W}$ в процессе вычисления):

$$\mathrm{Anc}_{\mathcal{W} \to \mathcal{W}'}(f) = \begin{cases} \varnothing, \text{ если } \|f.parent\|_{\mathcal{W} \to \mathcal{W}'} = \bot \\ \|f.parent\|_{\mathcal{W} \to \mathcal{W}_1} \cup \left(\bigcup\limits_{x \in \|f.parent\|_{\mathcal{W} \to \mathcal{W}''}} \mathrm{Anc}_{\mathcal{W}_i \to \mathcal{W}_{i+1}}(x) \right), \text{ где } \mathcal{W}' = \mathcal{W}_N \end{cases} \tag{1.15}$$

С учетом введенных обозначений, получим

$$\Phi^C(\alpha, \mathcal{W}) = \begin{cases} \langle \alpha, \mathcal{W} \rangle, & \text{если } \forall \langle f, s \rangle \in \mathbb{I} \quad \alpha(f, s) \neq todo \\ \langle \alpha', \mathcal{W}' \rangle, & \text{где } \langle \alpha'', \mathcal{W}'' \rangle = \varphi^C_{\mathcal{W}(f,s).\sqsubseteq_d}(\mathcal{W}(f, s).daemons, \alpha, \mathcal{W}), \\ & \langle f, s \rangle = \min\{\langle f_x, s_x \rangle | \alpha(f_x, s_x) = todo\} \\ & \alpha''' = \alpha'' \left[\Delta(\mathcal{W}(f, s).daemons) \cup \left(\bigcup\limits_{p \in \mathrm{Anc}_{\mathcal{W}'' \to \mathcal{W}'}(f)} \{p.s\} \right) \right] \\ & \alpha' = \alpha'''[f.s \leftarrow applied] \end{cases} \tag{1.16}$$

Функция Φ находит первый (в смысле лексикографической упорядоченности) слот $f.s$, помеченный в α как *todo*, использует φ для применения всех связанных с ним правил-демонов с соответствующим отношением порядка, помечает все измененные слоты, а также одноименные слоты всех родителей f, как кандидатов на дальнейшую обработку, и, наконец, устанавливает значение $\alpha(f, s)$ как *applied*, что исключает повторное применение демонов для этого слота.

Утверждение 1.5. *Функция* Φ *является монотонной в смысле естественного индуцированного отношения порядка на* $\mathcal{A} \times \mathbb{W}$ *для любого контекста* $C \in \mathbb{C}$.

Доказательство. следует из лемм 1.1 и 1.2 и утверждения 1.4. □

Утверждение 1.6. *Существует функция прямого вывода для слота* $f.s$ $\Phi : \mathbb{I} \to \mathcal{W} \to \mathcal{W}$, *которая для состояния* $\mathcal{W}$ *возвращает конечное состояние* $\mathcal{W}' = \Phi_{f.s}(\mathcal{W})$, *полученное применением всех ассоциированных со слотом правил прямого вывода.*

Доказательство. Из монотонности функции Φ следует существование решетки неподвижных точек, и, следовательно, $\forall \alpha \in \mathcal{A}, \mathcal{W} \in \mathbb{W} \quad \exists \langle \alpha', \mathcal{W}' \rangle = \bigsqcup_{n=1}^{\infty} \Phi^n(\alpha, \mathcal{W})$. Положим $\Phi_\alpha^C(\mathcal{W}) = \mathcal{W}'$. Тогда можно определить функцию прямого вывода для слота $f.s$ следующим образом:

$$\Phi_{f.s}(\mathcal{W}) = \Phi^f_{\alpha_0[f.s \leftarrow todo]}(\mathcal{W}) \tag{1.17}$$

где $\alpha_0(f, s) = free \; \forall \, \langle f, s \rangle \in \mathbb{I}$. □

Функцию $\Phi_{f.s}(\mathcal{W})$ будем называть **функцией направленного прямого вывода**, подчеркивая, что вывод производится начиная от слота $f.s$. Можно аналогичным образом определить функцию **исчерпывающего прямого вывода** $\Phi(\mathcal{W})$, которая будет рассматривать все слоты и все правила на предмет возможного их применения в соответствии с некоторой стратегией разрешения конфликтов. Для ее определения мы можем вообще отказаться от функции активации, а в качестве одношаговой функции вывода рассматривать $\phi' : \mathcal{W} \to \mathcal{W}$, получая $\Phi(\mathcal{W}) = \operatorname{fix} \phi(\mathcal{W})$. Следующее утверждение устанавливает связь между введенными понятиями:

Утверждение 1.7. *Процесс направленного прямого вывода является полным относительно процесса исчерпывающего прямого вывода, т.е. если $\mathcal{W}_0$ — состояние, для которого $\Phi(\mathcal{W}_0) = \mathcal{W}_0$ (такое состояние будем называть* **устойчивым относительно прямого вывода***), то существует такая стратегия разрешения конфликтов исчерпывающего вывода $\sqsubseteq$, что $\Phi_{f.s}(\mathcal{W}_0[f.s \leftarrow v]) = \Phi^{\sqsubseteq}(\mathcal{W}_0[f.s \leftarrow v])$.*

Идея доказательства. Данное утверждение подразумевает корректное построение сети демонов, т.е. такое, при котором все правила, которые зависят от изменения значения слота $f.s$, содержатся среди демонов этого слота, т.е множество правил, применяемых исчерпывающим и направленным выводом на первом шаге будут совпадать. Более строго можно показать по индукции, что если $|\mathcal{W} - \mathcal{W}_0| = \{f.s\}$ и $\mathcal{W}_0 = \operatorname{fix} \Phi$, то $\Phi_{f.s}(\mathcal{W}) = \Phi(\mathcal{W})$. Откуда и следует истинность доказываемого утверждения. □

1.3.4.5. Проверка ограничений

В предложенной модели с каждым слотом ассоциировано множество ограничений $\{C_i\}_{i=1}^N$, каждое из которых представляет собой выражение, которое должно быть истинно для данного слота. В этих выражения используется ссылка на текущий фрейм *this*, позволяющая организовать наследование ограничений для слота рекурсивным вызовом ограничений родительского слота. Опишем такую семантику ограничений, при которой текущее состояние $\mathcal{W}$ не изменяется. Более совершенная семантика, учитывающая возможность обратного вывода необходимых слотов в ограничениях, нами рассматриваться и реализовываться не будет.

Для проверки ограничений введем функцию check : $\mathbb{C} \to \mathcal{W} \to \mathbb{I} \to \{\mathbf{true}, \mathbf{false}\}$, определенную как

$$\text{check}^C_{f.s}(\mathcal{W}) = \text{isTrue}(\|C_1\|^{iC}_{\mathcal{W} \to \mathcal{W}} \, \mathbf{or} \ldots \mathbf{or} \, \|C_N\|^{iC}_{\mathcal{W} \to \mathcal{W}}), \text{ где } \mathcal{W}(f, s).constraints = \{C_i\}_{i=1}^N \tag{1.18}$$

Для проверки ограничений с учетом наследования, подобно тому, как это было сделано выше, введем функцию $\text{Anc}'_{\mathcal{W}}(f)$, возвращающую множество тех родительских

фреймов для f, значения которых уже были получены ранее или заданы явно:

$$\mathrm{Anc}'_{\mathcal{W}}(f) = \begin{cases} \varnothing, \text{ если } \|f.parent\|'_{\mathcal{W}\to\mathcal{W}} = \bot \\ \|f.parent\|'_{\mathcal{W}\to\mathcal{W}} \cup \left(\bigcup\limits_{x\in\|f.parent\|'_{\mathcal{W}\to\mathcal{W}}} \mathrm{Anc}'_{\mathcal{W}}(x) \right) \end{cases} \tag{1.19}$$

Тогда функция полной проверки ограничений

$$\mathrm{ensure}_{f.s}(\mathcal{W}) = \mathrm{check}^{f}_{f.s}(\mathcal{W}) \,\mathbf{or}\, \left(\mathop{\mathbf{or}}_{p\in\mathrm{Anc}'_{\mathcal{W}}(f)} \mathrm{check}^{f}_{p.s}(\mathcal{W}) \right) \tag{1.20}$$

Для определения семантики проверки ограничений в процессе вывода переопределим функцию изменения значения слота следующим образом:

$$write(\langle f, s\rangle, v, \mathcal{W}) = \mathrm{isTrue}\,\mathrm{ensure}_{f.s}(\mathcal{W}[f.s \leftarrow v]) \to \mathcal{W}[f.s \leftarrow v]; \mathcal{W} \tag{1.21}$$

1.3.4.6. Комбинированный вывод

Под комбинированным логическим выводом подразумевается такой механизм вывода, при котором поочередно применяются прямой и обратный вывод. Такой подход сочетает в себе достоинства двух основных методов вывода, а кроме того является более универсальным, так как оставляет возможность применять как прямой и обратный вывод в чистом виде, так и их комбинацию. В этом случае обратный вывод определяет основной маршрут движения по дереву решений, а также отвечает за получение необходимой информации от пользователя в процессе вывода, в то время как прямой вывод позволяет "расширить" множество полученных данных за счет того, что может быть получено из уже имеющегося множества без дополнительной информации извне.

Нами будет рассмотрен подход, при котором после каждого шага обратного вывода, т.е. как только получено значение некоторого слота, будет применяться направленный исчерпывающий прямой вывод. Для описания такой семантики выражение 1.7 следует записать следующим образом:

$$\|f.s\|_{\mathcal{W}\to\mathcal{W}'} = \begin{cases} \mathcal{W}(f,s).value, \mathcal{W}(f,s).value \neq \bot \text{ (при этом } \mathcal{W}' = \mathcal{W}) \\ \bot, \mathcal{W}(f.s).busy = \mathbf{true} \\ x = \mu_{\mathcal{W}[f.s.busy\leftarrow\mathbf{true}]\to\mathcal{W}''}(\langle \mathcal{W}(f,s).rules, \mathcal{W}(f,s).\sqsubseteq_q\rangle), \\ \qquad \mathcal{W}' = \Phi_{f.s}(\mathcal{W}''[f.s \leftarrow x, f.s.busy \leftarrow \mathbf{false}]) \quad (x \neq \bot) \\ y = \mu_{\mathcal{W}'''\to\mathcal{W}''''}(\{p.s | p \in \|f.parent\|_{\mathcal{W}''\to\mathcal{W}'''}\}), \\ \qquad \mathcal{W}' = \Phi_{f.s}(\mathcal{W}''''[f.s \leftarrow y, f.s.busy \leftarrow \mathbf{false}]) \quad (y \neq \bot) \\ \bot, \text{ в противном случае} \end{cases} \tag{1.22}$$

Отличие от предыдущего определения состоит в том, что после обратного вывода к заключительному состоянию применяется функция направленного прямого вывода $\Phi_{f.s}$, которая обеспечивает срабатывание всех релевантных правил прямого вывода для только что обновленного слота, а также для всех слотов, значения которых изменились в процессе прямого вывода. Если в данном определении полагать, что функция $\mathcal{W}[\ldots]$ осуществляет проверку ограничений, то мы получим описание семантики вывода с возможностью отката по невыполнению ограничений. Такой откат, однако, будет приводить к использованию других правил для выяснения значения слота, но не будет нарушать монотонности вывода.

1.3.4.7. Некоторые полезные свойства определенной семантики

При определении отношения $\|\cdot\|$ в доказательстве утверждения 1.2 было использовано свойство монотонности операции вывода, которое может быть сформулировано следующим образом:

Утверждение 1.8. *Для любого контекста вызова $C \in \mathbb{C}$ и выражения $E \in \mathbb{E}$ функция $\|E\|^C_{\rightarrow \cdot} : \mathbb{W} \to \mathbb{W}$ определена и монотонна по $\mathcal{W}$.*

Доказательство. проводится рассмотрением определений $\|\cdot\|$ для различных классов выражений, и следует из сформулированного ниже (лемма 1.3) свойства монотонности элементарной операции над состояниями *write*. Для случаев рекурсивного определения $\|\cdot\|$ корректное определение функции $\|\cdot\|$ было показано, а монотонность следует из представимости фиксированной точки монотонной функции в виде конечной композиции ее применения. □

Лемма 1.3. $\forall\, \mathcal{W} \in \mathbb{W} \quad \mathcal{W}[f.s \leftarrow v] \sqsupseteq \mathcal{W}$

Здесь также следует отметить то, что функция $\|\cdot\|$ определена для любого корректного выражения $E \in \mathbb{E}$, и ее вычисление требует конечного числа переходов в пространстве состояний. Этот факт следует из монотонности вывода и наличия неподвижной точки, которая достигается за конечное число шагов.

1.3.4.8. Комбинированный логический вывод как процесс поиска в графе состояний

Как было показано в утверждении 1.1 в разделе 1.3.3.1, множество состояний системы может быть представлено в виде конечного фактор-множества $\mathfrak{W}_{\cong}$. Это множество можно также рассматривать в виде направленного биграфа $\mathfrak{G} = \langle \mathfrak{W}_{\cong}, \mathfrak{U}, \mathfrak{V} \rangle$, с двумя множествами дуг $\mathfrak{U}$ и $\mathfrak{V}$, соответствующих возможным переходам в процессе прямого и обратного вывода. При этом:

$$\langle \hat{\mathcal{W}}_1 \in \hat{\mathcal{W}}_2 \rangle \in \mathfrak{U} \iff \forall\, \mathcal{W}_1 \in \hat{\mathcal{W}}_1\ \exists \mathcal{W}_2 \in \hat{\mathcal{W}}_2 \ :\ \mathcal{W}_2 = \varphi(\mathcal{W}_1) \tag{1.23}$$

где φ — одношаговая функция исчерпывающего логического вывода, а

$$\langle \hat{\mathcal{W}}_1 \in \hat{\mathcal{W}}_2 \rangle \in \mathfrak{V} \iff \forall\, \mathcal{W}_1 \in \hat{\mathcal{W}}_1\ \exists \mathcal{W}_2 \in \hat{\mathcal{W}}_2 \ :\ \mathcal{W}_2 \lhd \mathcal{W}_1 \tag{1.24}$$

Введенное здесь отношение $\lhd$ означает, что состояние $\mathcal{W}_1$ может быть выведено из $\mathcal{W}_2$ за один шаг обратного вывода, т.е. если $|\mathcal{W}_1 - \mathcal{W}_2| = \langle f, s \rangle$, то значение $f.s$ может быть получено как $\|f.s\|_{\mathcal{W}_1 \to \mathcal{W}_2}$ без промежуточных состояний (это гарантируется одноэлементностью разности двух состояний).

Такое представление в виде графа позволяет наглядно визуализировать процесс логического вывода, а также формализовывать семантику этого процесса как процесса поиска в графе, а не путем введения теоретико-решеточных построений. Говоря нестрого, при комбинированном виде производится один шаг по графу в соответствии с дугой обратного вывода ($\mathfrak{V}$), а затем оттуда максимальное количество шагов по дугам прямого вывода $\mathfrak{U}$, пока мы не окажемся в вершине, из которой нет больше исходящих дуг прямого вывода. Тогда производится еще один шаг обратного вывода и т.д.

Более строгая формализация процесса вывода в графе $\mathfrak{G}$ и на множестве $\mathfrak{W}_{\cong}$, а также доказательство ее эквивалентности приведенной выше семантике здесь не приводятся, хотя исследование полезных свойств графовой модели представляет самостоятельный интерес.

1.3.5. Язык представления знаний и семантика вывода

В отличие от классических языков программирования [12], непосредственное описание семантики для языка представления знаний (ЯПЗ) не имеет смысла, так как конструкции языка не носят процедурного смысла и сами по себе не влияют на состояние выполнения. Конструкции языка представления знаний (по крайней мере большинство из них, называемые **декларативными**) формируют начальное состояние задачи $\mathcal{W}_0$, т.е. те его составляющие, которые отвечают за представление базы знаний ($\mathcal{K}_{ss_1}$ и $\mathcal{K}_d$, представленные структурой фреймовой иерархии и присоединенными к слотам правилами). После того, как начальное состояние сформировано, запрашивается значение некоторого слота (используется соответствующая **процедурная** команда языка либо другие, определяемые реализацией, способы), что инициирует логический вывод из начального состояния $\mathcal{W}_0$ с описанной выше семантикой.

Для полного формального описания языка представления знаний необходимо определить правила трансляции синтаксических конструкций языка в математическую модель, представленную множеством $\mathcal{W}_0$. Формальное описание синтаксиса языка JFMDL (Julia Frame Model Definition Language) в виде синтаксических диаграмм содержится в приложении А. В данной работе мы из-за громоздкости не будем описывать правила формальной трансляции предложений языка в теоретико-множественные конструкции, а ограничимся лишь пояснениями. Далее при описании архитектуры инструментария будет более подробно описан перевод конструкций языка во внутреннее объектное представление, которое по существу является аналогом математической модели.

Получение состояния $\mathcal{W}_0$ по тексту на ЯПЗ по сути представляет собой описание семантики языка, при котором денотатом всего языкового представления базы знаний будет искомое состояние $\mathcal{W}_0$, а денотатами языковых конструкций — функции преобразования состояния. Соответственно, построение формальной модели будет строиться на основе семантической функции $\mathcal{L} : A^* \to (\mathcal{W} \to \mathcal{W})$, где A^* — текст программы[17], $\mathcal{W} \to \mathcal{W}$ — соответствующее отображение состояния программы. Начальное состояние $\mathcal{W}_0$ получается из текста программы P как $\mathcal{W}_0 = \mathcal{L}(P)(0_{\mathbb{W}})$, где $0_{\mathbb{W}}$ — нулевая функция состояния, задаваемая соотношением $0_{\mathbb{W}}(f, s) = \langle \bot, \bot, \varnothing, \varnothing, \varnothing, \sqsubseteq_{def}, \sqsubseteq_{def}, \mathbf{false} \rangle \quad \forall \langle f, s \rangle \in \mathbb{I}$.

База знаний строится как последовательность элементарных конструкций, которые могут быть правилами или описаниями фреймов. Для последовательности конструкций очевидно $\mathcal{L}(P_1; P_2) = \mathcal{L}(P_2) \circ \mathcal{L}(P_1)$.

Конструкция описания фрейма интерпретируется следующим образом:

$$\begin{gathered}\mathcal{L}(\texttt{frame}\ f\ \texttt{parent}\ p\ \{s_1; \ldots; s_n\}) = \mathcal{L}^{(f)}(s_n) \circ \ldots \circ \mathcal{L}^{(f)}(s_1) \circ \\ \circ \mathrm{insert}(f.parent, \langle p, \bot, \varnothing, \varnothing, \varnothing, \sqsubseteq_{def}, \sqsubseteq_{def}, \mathbf{false} \rangle)\end{gathered} \tag{1.25}$$

где s_i — конструкция описания слота, а $\mathrm{insert} : \mathbb{I} \times \mathbb{T} \to (\mathcal{W} \to \mathcal{W})$ — операция добавления слота к функции состояния, определяемая следующим образом:

$$\mathrm{insert}(f.s, X)(\mathcal{W}) = \lambda f_x, s_x . \langle f_x, s_x \rangle = \langle f, s \rangle \to X \star \mathcal{W}(f_x, s_x); \mathcal{W}(f_x, s_x) \tag{1.26}$$

Здесь $\star$ — операция слияния кортежей, представляющих один и тот же слот:

$$\begin{aligned}&(u \star v)_i = (u_i = \bot) \to v_i; u_i, \text{ для } i = 1, 2, 6, 7, 8; \\ &(u \star v)_i = u_i \cup v_i, \text{ для } i = 3, 4, 5\end{aligned} \tag{1.27}$$

[17] Заметим, что A^* не совпадает в областью определения функции $D(\mathcal{L})$, которая задается формальной грамматикой языка.

Описание отдельного слота интерпретируется следующим образом (для слота целого типа):

$$\mathcal{L}^{(f)}(\texttt{scalar}\ n\ \texttt{int def}\ v\ \texttt{ruleselectionstrategy}\ r\ ) = \\ \text{insert}[f.n, \langle \bot, v, \varnothing, \varnothing, \varnothing, r, \sqsubseteq_{def}, \textbf{false}\rangle] \tag{1.28}$$

Более строго следует описать операцию определения слота через составные компоненты определения, как это сделано для описания фрейма, которые будут устанавливать соответствующие компоненты представляющего слот кортежа при помощи отдельных операций insert.

Для формального описания правил придется описывать преобразование выражений во "внешнем" синтаксисе ЯПЗ в синтаксис, описанный в разделе 1.3.4.1, а также преобразование правил прямого и обратного вывода в соответствующие наборы выражений. Для правил обратного вывода `if` *cond* `then` $f.s = expr$ проделывается следующее: вхождения имени фрейма f в обоих частях выражения заменяется на *this* (это необходимо для корректной работы механизма наследования), и к слоту $f.s$ при помощи операции insert добавляется правило-запрос вида $cond' \rightarrow expr'; \bot$, где $cond', expr'$ — модифицированные выражения *cond* и *expr* соответственно. Для правил прямого вывода `when` *cond* `then` *actions* из выражения-посылки *cond* выделяется множество фигурирующих в нем ссылок на слоты $\{\langle f_i, s_i\rangle\}_{i=1}^{n}$, и к каждому из этих слотов присоединяется правило-демон вида $\langle cond', actions'\rangle$. При этом в *actions* операции $f.s = v$ заменяются на соответствующие вызовы функции $[f.s \leftarrow v]$, т.е. происходит преобразование из внешнего языка во внутренние структуры математической модели.

1.3.6. Назначение формального синтаксиса и семантики

В предыдущих разделах на достаточном уровне строгости излагалась двухуровневая модель формальной семантики языка представления знаний, при котором сначала конструкции языка транслируются в теоретико-множественную модель представления знаний, на которой затем определяется семантика логического вывода. Необходимость и полезность такого построения обусловлена следующими соображениями[18]:

- Формальная семантика позволяет математически строго доказать завершимость, а также другие полезные свойства логического вывода во фреймовой системе.
- На основе формального синтаксиса и семантики $\mathcal{L}$ может непосредственно производиться разработка компилятора с ЯПЗ во внутреннее представление, что было проделано автором и описано далее в работе. Более того, формальная семантика может лечь в основу реализации соответствующего программного прототипа на функциональном языке программирования (например, ML или Haskell).
- Формальное описание позволяет однозначно и строго описать синтаксис и семантику языка, исключая тем самым различные толкования как языковых конструкций, так и процесса логического вывода.

[18] Более подробно о назначении формальной семантике языков программирования можно прочитать в [12, 18, 29, 60].

1.4. Архитектура и программная реализация инструментария JULIA

1.4.1. Основные принципы

При проектировании инструментария JULIA в первую очередь принимались во внимание следующие соображения:

- **Расширяемость и открытость** означают возможность расширения инструментария произвольным программным кодом, включая пользовательские функции, реализованные на традиционном языке программирования, дополнительные типы данных, операции, стратегии выбора правил, процедуры-демоны, процедуры запросы и т.д.
- **Многоплатформенность**, достигаемая за счет реализации инструментария на языке Java.
- **Высокая степень интеграции** инструментария с **реляционными хранилищами данных**, с произвольным программным кодом (Java-классами) и **компонентами** корпоративной бизнес-логики (CORBA-объектами, JavaBeans и Enterprise JavaBeans).
- **Гибкость** инструментария, позволяющая настраивать стратегию эвристического поиска при логическом выводе.
- **Встраиваемость** инструментария в состав программных и информационных систем, реализованных как на Java, так и на любом языке программирования с поддержкой CORBA. Инструментарий может также использоваться в интернет-приложениях в составе Java сервлетов, аплетов или специализированных CGI-модулей. Механизм *обратных вызовов* (callbacks) позволяет использовать инструментарий в режиме недиалогового (неинтерактивного) обратного вывода.

Предлагаемая архитектура реализации гибридного инструментария JULIA для построения распределенных интеллектуальных систем приведена на рис. 1.6. Инструментарий представляет собой набор Java-классов с соответствующим программным интерфейсом (API), реализующих также набор CORBA-объектов для удаленного доступа к системе со стороны других JULIA-серверов, а также со стороны программных систем, поддерживающих CORBA.

В основе системы лежит **ядро** (представляющее собой библиотеку Java-классов, распространяемую в виде одного файла `julia_rt.jar`), которое включает в себя набор классов для поддержки фреймов и наследования, библиотеку типов данных, алгоритмы реализации прямого и обратного вывода, ряд дополнительных классов для обеспечения доступа к БД, к внешним Java-классам, компонентам JavaBeans, набор стратегий выбора правил при разрешении конфликтов, библиотеку основных функций и др. В виде отдельных библиотек оформлены классы, обеспечивающие поддержку CORBA для распределенного вывода (`julia_remote.jar`), а также набор утилит (`julia_utils.jar`), в том числе компилятор с языка представления знаний JFMDL во внутреннее сериализованное представление базы знаний[19]. Кроме того, обычно в комплект поставки входит объединенный файл `julia.jar`, включающий в себя все перечисленные компоненты.

[19]Такое разделение библиотек классов позволяет использовать JULIA в качестве библиотеки для обеспечения интеллектуальности локальных приложений, в которых не нужен удаленный доступ. При этом

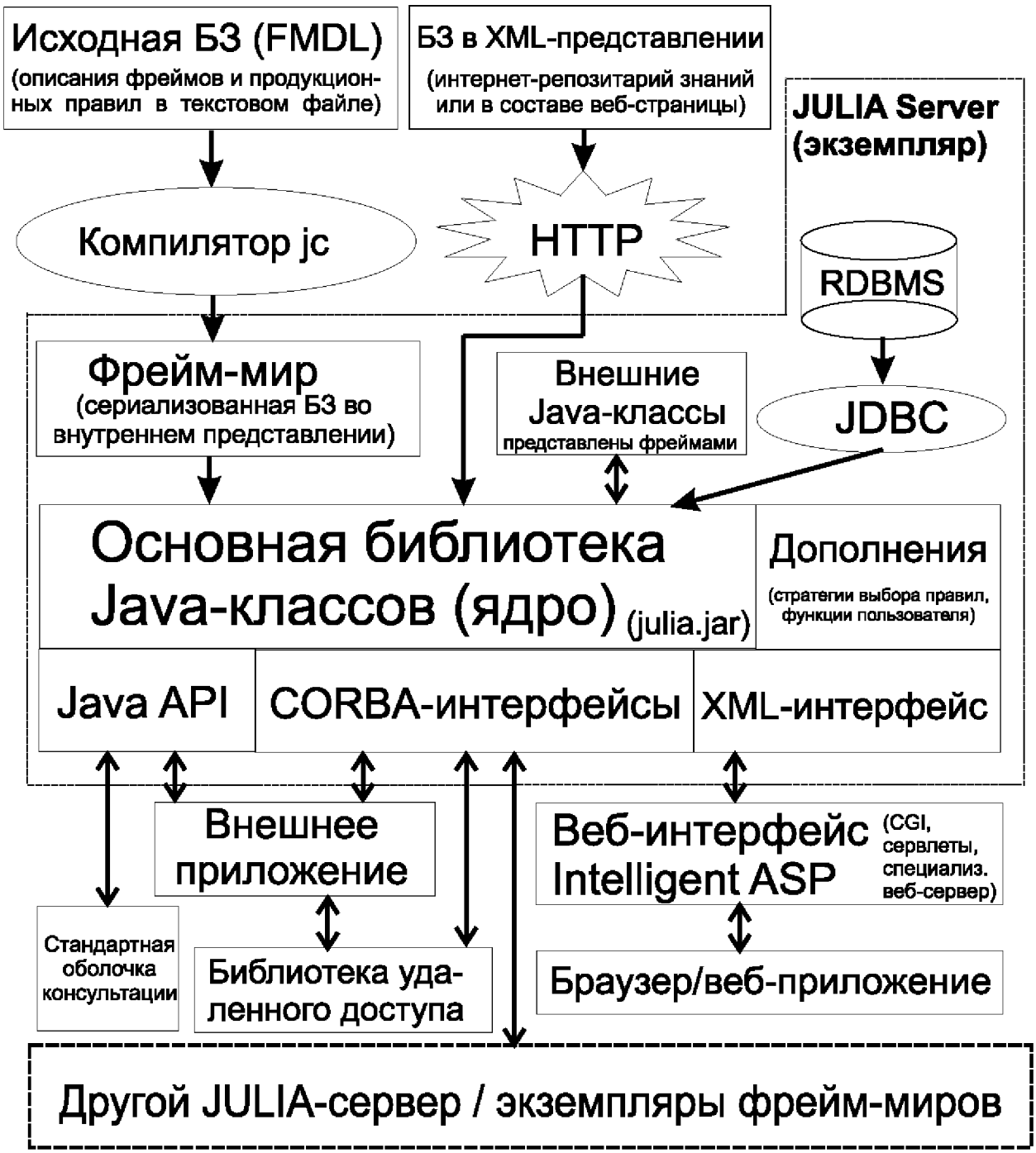


Рис. 1.6. Архитектура системы на базе инструментария JULIA

В верхней части диаграммы показаны различные способы, с помощью которых система получает знания и данные для своей работы; ниже ядра — различные интерфейсы для использования библиотеки в интерактивной консультации или из внешней информационной системы. Из рисунка видно, что инструментарий может использоваться как самостоятельная оболочка экспертных систем для интерактивной консультации, функционирующая в виде Java-апплета в браузере клиента или сервлета на веб-сервере[20], а также как составная часть программных систем через программный интерфейс Java (Julia API) или набор CORBA-интерфейсов.

База знаний обычно описывается на специализированном языке высокого уровня декларативности **JFMDL** (Julia Frame Model Definition Language), который включает в себя средства описания статических закономерностей предметной области (фреймовой иерархии) и динамических знаний в виде продукций прямого и обратного вывода. Это описание затем преобразуется специальным транслятором в набор Java-объектов (так называемый **фрейм-мир**), соответствующих конструкциям языка и представляющих собой начальное состояние $\mathcal{W}_0$, из которого будет начат логический вывод. Это т.н. **внутреннее представление** (см. рис. 1.10 и 1.11) может быть непосредственно использовано для логического вывода, либо в любой момент сериализовано в поток и сохранено в виде файла на диске[21], в виде внешнего XML-представления[22], либо в объектной базе данных. В частности, при построении реальных систем имеет смысл сохранять базу знаний в виде внутреннего сериализованного представления, а для консультации по такой базе знаний применять более компактную библиотеку времени выполнения (`julia_rt.jar`), не включающую в себя достаточно объемные компоненты компилятора.

1.4.2. Основные компоненты инструментария

Инструментарий JULIA состоит из следующих основных компонент (см. рис. 1.7):

Библиотека JULIA — набор классов, обеспечивающих основную функциональность системы, сгруппированных в файл `julia_rt.jar`. Эта библиотека в свою очередь состоит из следующих составных частей:

- **Базовая библиотека** (Base Library) включает в себя родительский класс `JObject`, конфигурационные классы `Defs` и `SysDefs`, поддержку объединения набора классов в фрейм-миры (`World`) и другие основные функции.
- **Библиотека типов** (Type Library) — обеспечивает поддержку динамических типов данных, унаследованных от `TAny`.
- **Библиотека выражений** (Expression Library) — обеспечивает поддержку выражений, представленных в виде деревьев общего вида, на основе библиотеки типов.

объем требуемого файла классов существенно уменьшается и в минимальном варианте составляет около 70 килобайт.

[20]Как будет показано ниже, система допускает также смешанные режимы выполнения, при которой данные с сервера подгружаются динамически. Более подробно сетевая функциональность будет рассмотрена в главе 2.

[21]Таким файлам обычно дается расширение `jsw` — Julia Serialized World.

[22]XML-представление (именуемое также JML, Julia Meta Language) также может быть сохранено в поток или текстовый файл с расширением `.jml`, однако такое представление является менее компактным.

- **Библиотека процедур и демонов** (Actions Library) предоставляет набор стандартных "действий", которые могут выступать в качестве процедур-запросов и процедур-демонов и являются основой для реализации продукционного представления знаний и логического вывода. Включает также поддержку ограничений (Constaints).
- **Фреймовая библиотека** (Frame Library) обеспечивает поддержку фреймов различных типов, в том числе для интеграции с произвольными Java-классами и для поддержки логического вывода. Поддерживает функции создания и добавления фреймов к модели, а также манипуляции со слотами фрейма, включая запрос значения, инициирующий логический вывод.
- **Библиотека доступа к РСУБД** позволяет интегрировать фреймовую модель с реляционными базами данных, доступными по протоколу JDBC.
- **Расширения и дополнения**, в том числе различные стратегии выбора правил, библиотеки функций, типовые фрейм-классы, диалоговые классы и т.д.

Набор утилит, включающих в себя:

- **Транслятор с JFMDL**, позволяющий получать по исходному файлу базы знаний файл во внутреннем представлении, а также выполнять интерактивные запросы и консультации в режиме командной строки.
- **Утилита интерактивной консультации Consult**, функционирующая в режиме апплета или приложения и позволяющая проводить интерактивные консультации в режиме вопрос-ответ.
- **Браузер фреймовой модели**, позволяющий в интерактивном диалоговом режиме просматривать фреймовую модель и вносить в нее изменения.
- **Дополнительные утилиты**: запуска интерактивной консультации, распечатки сериализованного представления, форматтер текста, интерактивная документация и др.

Базовая библиотека ввода-вывода и вспомогательных функций, которая предоставляет набор проблемно-независимых функций, является разделяемой и используется в различных программных проектах автора.

1.4.3. Объектная архитектура инструментария

Инструментарий JULIA построен как набор классов на объектно-ориентированном языке Java, соответственно, при его разработке использовался объектно-ориентированный подход к анализу, проектированию и программированию [3, 10]. Для построения объектной модели использовались элементы языка моделирования UML [65, 103]. Основные элементы иерархии интерфейсов и классов инструментария приведены на рисунке 1.8[23].

Практически все классы унаследованы от общего родителя `JObject`. Основная функция этого объекта — обеспечивать связь всех остальных элементов фреймовой иерархии в рамках одной базы знаний, т.е. поддержание целостности т.н. **фрейм-мира** (Frame

[23] Для упрощения восприятия общей картины на рисунке показаны только основные классы и ключевые методы и атрибуты этих классов.

Рис. 1.7. Основные компоненты инструментария JULIA

World). При создании практически любого объекта библиотки необходимо указание в качестве одного из параметров конструктора объекта `JObject`, т.е. любого другого JULIA-объекта, относящегося к данному фрейм-миру. В одних случаях такой объект передается неявно (например, в конструкторах копирования), в других следует явно предусмотреть передачу какого-нибудь доступного объекта конструктору.

Для исключения конфликта имен, основные классы JULIA определены в пакете `com.shwars.julia`. Для разделения всего множества классов на группы выделяются также подпакеты:

- `com.shwars.julia.DB` для функций работы с базами данных
- `com.shwars.julia.Actions` содержит дополнительные процедуры-демоны и процедуры-запросы
- `com.shwars.julia.Libraries` включает в себя стандартные библиотеки функций
- `com.shwars.julia.JavaClassExt` содержит набор включенных в систему императивных фреймов-расширений, реализованных на Java
- `com.shwars.julia.util` реализует набор стандартных утилит
- `com.shwars.julia.parser` содержит лексический анализатор и транслятор с JFMDL, а также соответствующие дополнительные классы
- `com.shwars.julia.RuleSelectionStrategies` определяет различные стратегии выбора правил при прямом и обратном выводе
- `com.shwars.julia.Askers` содержит набор callback-классов (**Asker**-ов) для обеспечения диалога с пользователем при обратном выводе

Кроме того, в пакете `com.shwars.julia.CORBA` определяются сетевые расширения инструментария JULIA, описанные в главе 2. Компоненты базовой библиотеки ввода-вывода и дополнительных функций находятся в пакетах `com.shwars.io` и `com.shwars.util` соответственно.

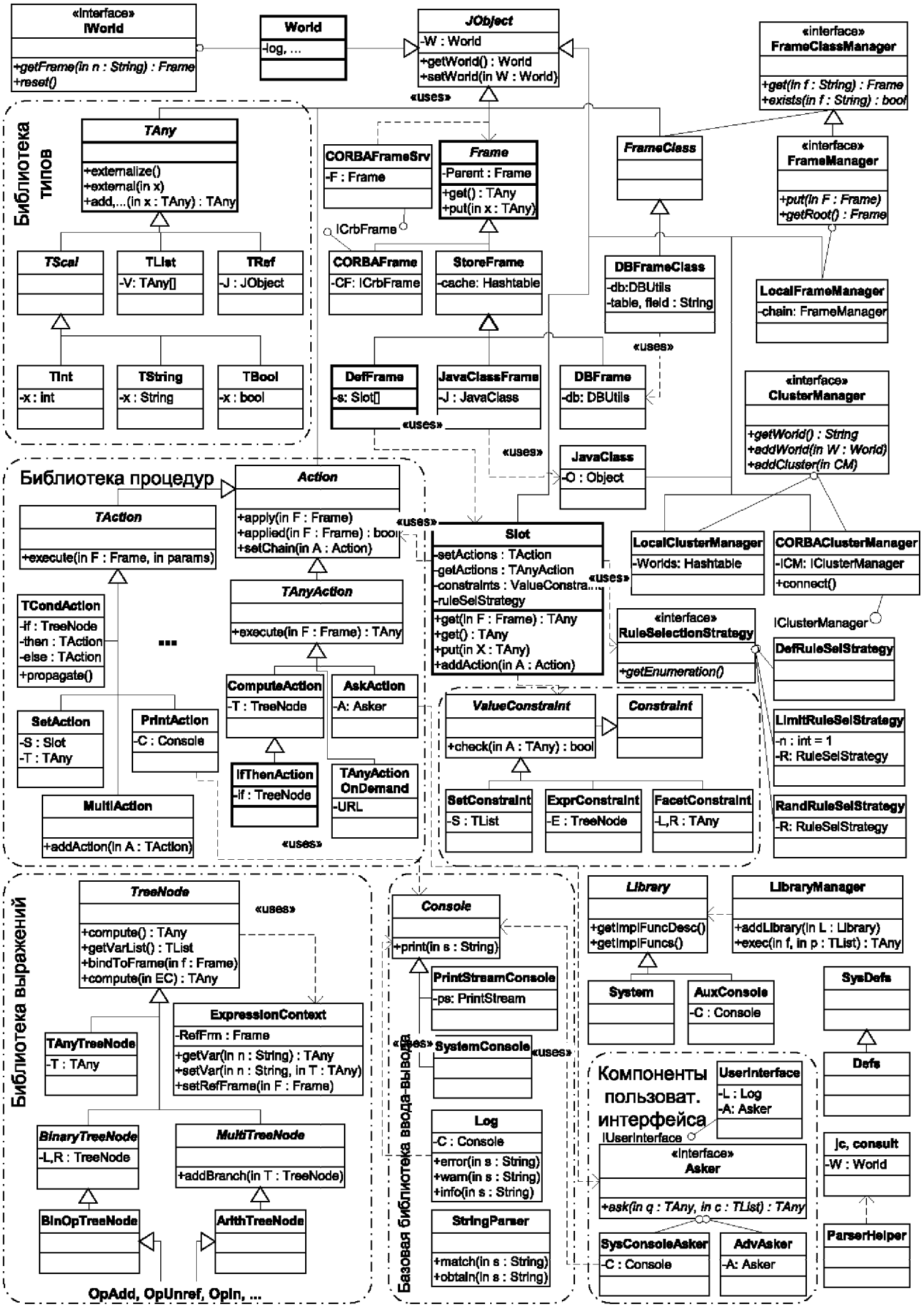


Рис. 1.8. Диаграмма классов инструментария JULIA (UML)

1.4.4. Типы данных и выражения

1.4.4.1. Библиотека типов

Основными типами данных, представленными в системе, являются **скалярный** (реализованный подклассом `TScal`), который подразделяется на **числовой** (целый `TInt` или с плавающей точкой `TFloat`), **логический** (`TBool`) и **строковый** (`TString`) подтипы, **списковый** (`TList`), содержащий произвольное количество элементов любого типа, в том числе спискового, и **ссылочный** (`TRef`), содержащий ссылку на фрейм или слот. Библиотека может расширяться другими типами данных (например, текстовым, шаблонным и др.) путем реализации соответствующих Java-классов, унаследованных от базового родительского класса `TAny` (см. рис. 1.8).

Родительский класс `TAny` определяет типово-независимые операции сложения (`add`), принадлежности (`in`) и другие, операции приведения типов к типам Java (`toString`, `toVector` и др.) и к типам JULIA (`toScal`, `toString` и др.), операции сравнения (`equal`, `lt` и др.) и т.д. Кроме того, определена операция **экстернализации** `externalize()`, т.е. преобразования значения во внешнее представление, основанное на XML. Обратная операция `external` позволяет по внешнему текстовому представлению однозначно восстановить значение одного из подтипов `TAny`. Унаследованные от `TAny` классы должны переопределять часть этих операций в соответствии с реализуемым типом.

1.4.4.2. Представление выражений

При представлении правил, присоединенных процедур и многих других элементов мы сталкиваемся с необходимостью представлять произвольные выражения в соответствии с библиотекой типов. Выражения представляются в виде дерева произвольного вида (см. рис. 1.9). Базовым классом для представления выражений является класс `TreeNode`, определяющий метод `compute()` для вычисления значения выражения в некотором контексте, задаваемом классом `ExpressionContext`.

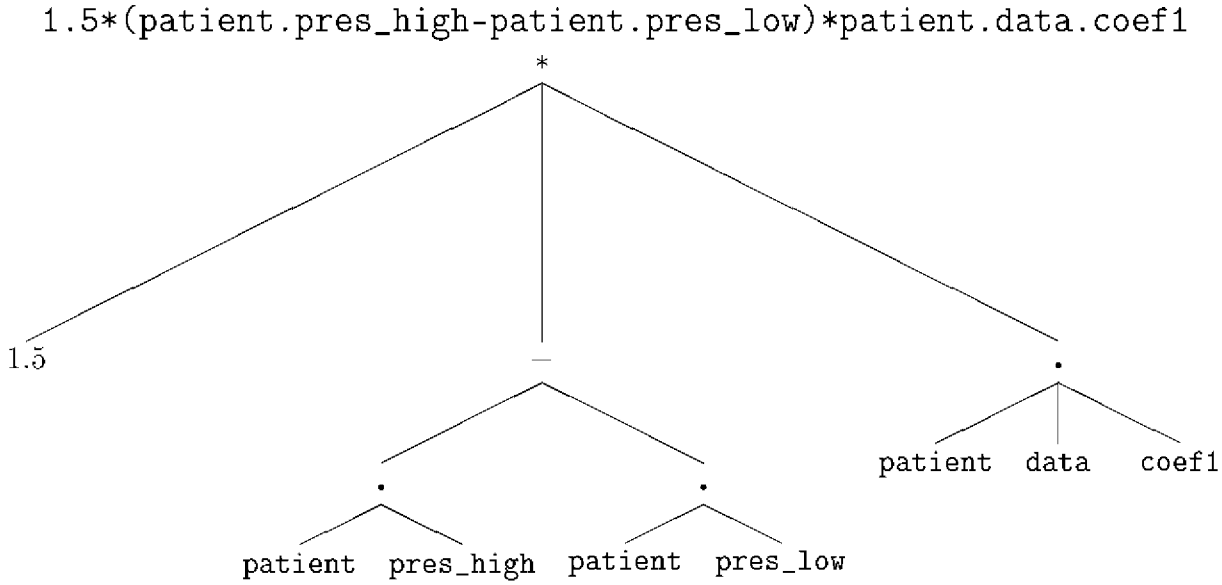


Рис. 1.9. Пример представления выражения деревом общего вида

От `TreeNode` унаследованы классы `BinaryTreeNode` для представления двоичного

дерева и `MultiTreeNode` — реализация дерева общего вида. Наследниками этих классов для представления операций являются `BinOpTreeNode` и `ArithTreeNode` соответственно, от которых в свою очередь могут быть унаследованы классы, реализующие собственно операции: `OpAdd`, `OpUnref` и другие. В текущей реализации функциональность операций содержится также непосредственно в классах `BinOpTreeNode` и `ArithTreeNode`.

1.4.5. Представление знаний и логический вывод

Инструментарий использует продукционно-фреймовое представление знаний, в котором статические знания о предметной области представляются в виде фреймовой иерархии, а в качестве динамических знаний о переходах между состояниями используются продукционные правила прямого и обратного вывода, сгруппированные вокруг соответствующих фреймов и слотов (см. рис. 1.10).

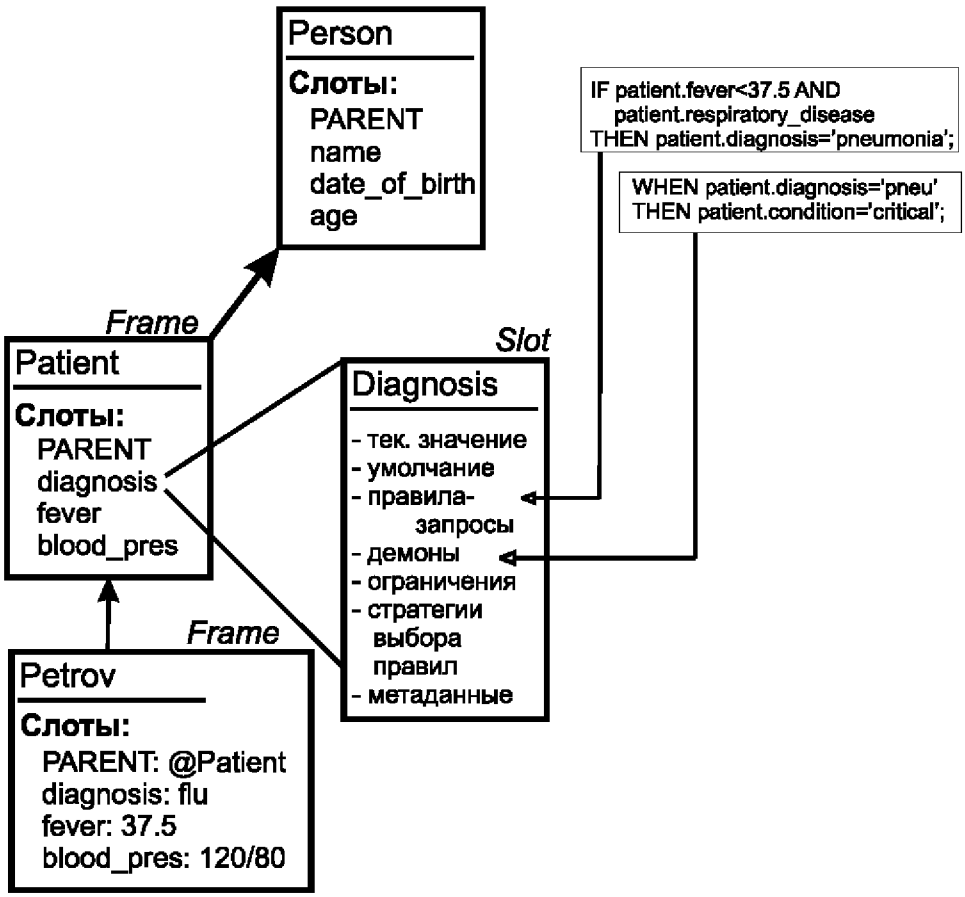


Рис. 1.10. Простейший пример фреймовой иерархии с продукционными правилами, сгруппированными в виде присоединенных процедур

1.4.5.1. Представление статических знаний

Фрейм рассматривается как набор **слотов**, каждый из которых может содержать значение заранее определенного или произвольного **типа**.

Каждый фрейм в процессе выполнения представляется экземпляром некоторого Java-класса, унаследованного от `Frame`. Класс фрейма определяет то, каким образом будет формироваться значения слотов в процессе вывода. Инструментарий содержит спе-

циальные классы для доступа к базам данных (DBFrame и DBFrameClass — подробнее см. раздел 1.5.1), для интерфейса с объектами, реализованными на императивных языках программирования (JavaClassFrame, JavaBeanFrame и CORBAObjFrame — подробнее см. раздел 1.5.2). Библиотека также может быть расширена дополнительными классами фреймов, хотя необходимость в этом редко возникает, так как произвольный Java-код может быть включен в состав фреймовой модели в виде Java-фреймов.

Большинство фреймов обычно представляются классом DefFrame, который обеспечивает хранение значений слотов в памяти и связывание с ними процесса прямого и обратного логического вывода[24]. С каждым слотом связывается тип содержащегося в нем значения, наборы процедур-демонов (выполняемых при присваивании слоту значения) и процедур-запросов (выполняемых, когда требуется узнать значение слота), а также набор ограничений (constraints) на множество допустимых значений (фасетные ограничения на диапазон значений FacetConstraint, на принадлежность определенному множеству значений SetContraint, либо ограничения общего вида, представленные произвольным логическим выражением ExprConstraint).

Все процедуры-демоны наследуются от общего абстрактного класса TAction и могут, например, выполнять присваивание значения некоторому слоту (SetAction), объединять несколько процедур в одну (MultiAction), выполнять некоторую процедуру условно в зависимости от истинности логического выражения (CondAction). Процедуры-запросы наследуются от TAnyAction (который, вместе с TAction, унаследован от общего абстрактного суперкласса Action) и возвращают некоторое значение в качестве результата своего выполнения. Библиотека определяет процедуры для вычисления выражения (ComputeAction), для условного вычисления выражения (IfThenAction), для запроса пользователю (AskAction) и др. Кроме того, возможно определение произвольных процедур-демонов и процедур-запросов на Java путем наследования от соответствующих классов.

Фреймы объединяются в **иерархию наследования**, при этом каждый фрейм имеет одного родителя, на которого указывает значение специального слота parent. В принципе, возможно расширение инструментария для поддержки множественного наследования путем использования списка родителей в качестве значения слота parent. Инструментарий поддерживает как статическое наследование (при котором для фрейма ссылка на родителя известна до начала вывода и представляется Java-ссылкой на соответствующий фрейм-объект), так и динамическое наследование, при котором родитель фрейма определяется в процессе вывода. Родитель может определяться в результате выполнения процедур-запросов, присоединенных к слоту parent (например, в результате обратного вывода путем применения продукционных правил, содержащих в правой части присваивание значения слоту parent), или в результате **фреймовой спецификации**, при которой производится обход некоторой фреймовой субиерархии и определение наименее абстрактного родителя, удовлетворяющего всем ограничениям на слоты, содержащие значения.

При реализации базы знаний ее описание на JFMDL транслируется во внутреннее представление фрейм-мира (пример такой трансляции приведен на рис. 1.11), представляющее собой совокупность описанных выше и взаимосвязанных объектов. Класс World служит для объединения всех объектов одной базы знаний в единое целое, что достигается за счет наследования от JObject, в результате чего каждый объект имеет доступ к

[24]Точнее, основную функциональность хранения значений слотов обеспечивает абстрактный класс StoreFrame, от которого наследуются другие классы, различающиеся способом получения значений слотов, в том числе и DefFrame.

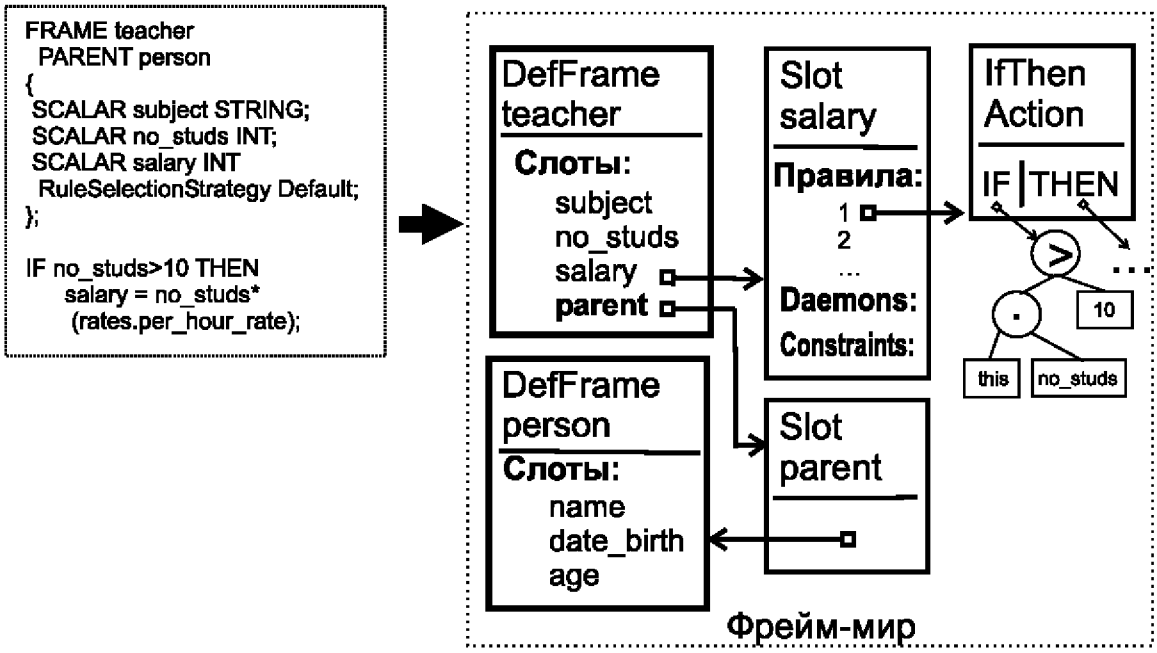


Рис. 1.11. Пример трансляции внешнего представления знаний на JFMDL во внутреннее представление.

породившему его фрейм-миру. В `World` описываются такие общие характеристики базы знаний, как используемый журнал протоколирования вывода (`Log`), callback-класс для запросов пользователю в процессе обратного вывода (`Asker`) и т.д. таким образом становится возможным запускать несколько параллельных или взаимодействующих процессов вывода в рамках одной Java-машины. Более подробно взаимодействие фрейм-миров рассматривается в главе 2.

Другим способом внешнего представления знаний служит язык, основанный на XML [122], который может использоваться для сохранения всей фреймовой иерархии или ее части. Таким образом, часть знаний предметной области может сохраняться в виде XML-репозитария и загружаться по протоколу HTTP в различные фрейм-миры. Использование XML позволяет, в случае необходимости, переходить от одного XML-диалекта представления знаний к другому[25] с помощью XSL, обеспечивая диалог между двумя системами с несколько различным внешним представлением знаний. XML-представление, однако, не является достаточно наглядным, поэтому имеет смысл использовать JFMDL для описания базы знаний, с последующей автоматической генерацией XML-представления для отдельных ее частей.

Другой полезной возможностью для организации репозитариев знаний является удаленное хранение правил и их загрузка по мере необходимости (on-demand rule loading), реализованное классам `TAnyActionOnDemand` и `TActionOnDemand` при этом в основном фрейм-мире содержится только заглушка правила, а в случае необходимости все его тело загружается в сериализованном виде из некоторого внешнего источника. Более подробно такая функциональность также описана в главе 2.

[25]Например, совместимому с форматом единого дескриптора ресурса RDF, разрабатываемого в рамках проекта Semantic Web [64].

1.4.5.2. Продукционные правила и механизм обратного вывода

Вне зависимости от способа внешнего представления знаний, продукционные правила обратного вывода транслируются в соответствующие процедуры-запросы, присоединенные к слотам, содержащимся в правой части правил, а правила прямого вывода — в соответствующие процедуры-демоны, присоединенные к слотам, содержащимся в левой части правил. Слоты, значения которых должны быть запрошены у пользователя в процессе обратного вывода, снабжаются соответствующими экземплярами класса `AskAction`.

Правила обратного вывода представлены одним из подклассов класса `TAnyAction` (реализующего процедуры, возвращающие значение `TAny`): `ComputeAction` для простых присваиваний и вычислений выражений (`ComputeAction` содержит в себе ссылку на дерево выражения, и при активизации вычисляет его, возвращая результат), `IfThenAction` для невырожденных продукционных правил с непустым множеством посылок (в этом случае сначала вычисляется условие, и при его истинности — целевое выражение). Каждая процедура может вдобавок содержать присоединенную процедуру, которая выполняется в случае успешной активизации (эта функциональность предусмотрена классом `Action`, от которого унаследованы процедуры как прямого, так и обратного вывода).

Как следствие такой модели представления правил, продукционные правила в простейшем случае могут содержать лишь одно присваивание в правой части. Правила, содержащие $n > 1$ присваиваний, транслируются в n процедур, каждая из которых содержит $n - 1$ присоединенных процедур-присваиваний. Такое сведение сложных правил к более простым несколько увеличивает расход памяти, но зато упрощает семантику времени выполнения полученной фреймовой модели[26], в то же время отражая смысловую нагрузку правил обратного вывода с множественными присваиваниями в правой части.

При запросе значения слота в процессе вывода рассматриваются все присоединенные к нему процедуры-запросы, и на основе некоторого эвристического правила определяется порядок возможного применения этих процедур $\sqsubseteq_q$. Этот порядок выполняет роль стратегии **разрешения конфликта** (conflict resolution) при обратном выводе, так как на каждом этапе позволяет выбрать из множества правил одно, подлежащее выполнению. За определение порядка применения процедур-запросов отвечает присоединенный к слоту объект, унаследованный от `RuleSelectionStrategy`, который реализует метод, возвращающий объект, реализующий интерфейс `Enumeration`, используемый затем для поочередного извлечения процедур в процессе их выполнения. В инструментарии реализованы стратегии поочередного выбора процедур-запросов (`DefRuleSelectionStrategy`) в соответствии с их порядком следования в исходной базе знаний, случайного выбора (`RandomRuleSelectionStrategy`), ограничения числа выбираемых правил (`LimitRuleSelectionStrategy` — совместно с любой другой стратегией позволяет ограничить число рассматриваемых правил некоторым значением). Несложным программированием Java-класса могут быть также реализованы другие, более сложные стратегии.

Для подлежащего выполнению правила производится **укороченное вычисление** условной части, которое, в свою очередь, может приводить к запросу значений других слотов, т.е. к рекурсивному инициированию процесса обратного вывода. В случае успеха

[26] Следует отметить, что семантика продукционных правил с множественными присваиваниями нами в разделе 1.3.4 не рассматривалась. Поэтому в контексте данной работы можно принять приведенное здесь правило трансляции за определение семантики правил с множественными присваиваниями, оставив более подробное изучение этого вопроса для дальнейшего исследования.

(в результате вычисления выражения получается истинное значение) выполняется присваивание слоту значения, содержащегося в правой части правила (это может привести к дальнейшему логическому выводу, если в выражении в правой части в свою очередь присутствуют ссылки на другие слоты), в случае неуспеха — рассматривается следующее правило.

Помимо правил, в качестве процедур-запросов могут использоваться процедуры запроса значения у пользователя `AskAction`, а также другие определенные пользователем наследники класса `TAnyAction`.

В любом случае, после присваивания слоту значения производится проверка ограничений (constraints), связанных со слотом (**фасетные ограничения**, facet constraints), с фреймом или со всей фреймовой иерархией (**референциальные ограничения**, referential constraints). Ограничения могут накладываться на тип присваиваемого значения, на диапазон или на множество значений, а также в форме произвольных выражений или определенным образом унаследованных Java-классов. Если вновь присвоенное значение не удовлетворяет ограничениям — слоту присваивается неопределенное значение $\perp$ (или `null` в терминах языка программирования), и применение процедур продолжается, таким образом производится частичный откат назад (partial backtracking)[27] и применение других правил. В текущей реализации инструментария роль ограничений сводится к проверке и обеспечению отката назад при поиске решения, а известные методы и алгоритмы **программирования в ограничениях** (constraint programming) [66] не используются. Совмещение этих алгоритмов с реализованным в JULIA комбинированным выводом представляет одно из направлений дальнейшего совершенствования инструментария.

Когда непротиворечащее ограничениям значение слота определено, дальнейшие действия зависят от установленного для данного слота режима выполнения процедур-запросов. По умолчанию, как только одно правило выполнилось, рассмотрение других правил с присваиванием в правой части не производится, а рассматриваются только правила, которые могут добавить значения к текущему значению слота (т.е. содержащие присваивание типа `+=`).

Алгоритм 1.1 показывает процесс вычисления значения слота. Если значение слота не удалось получить с использованием присоединенных процедур — аналогичным образом запрашивается значение соответсвующего слота фрейма–родителя; при этом в процедуру передается ссылка на текущий фрейм, которая называется **базовым фреймом**[28]. Таким образом, для унаследованных фреймов применяются все процедуры-запросы (соответственно, все продукционные правила обратного вывода), определенные для родителей.

1.4.5.3. Процедуры-демоны и механизм прямого вывода

Для реализации прямого вывода в классических продукционных системах семейства OPS5 [110, 117] применяются различные алгоритмы прямого вывода, наиболее распространенными из которых являются алгоритмы семейства Rete [74], по эффективности на порядок превосходящие "наивный" алгоритм, основанный на сопоставлении на каждом шаге всех правил с состоянием рабочей памяти (т.н. *recognize-act cycle* [75]).

[27] В связи со строгой монотонностью вывода при частичном откате не производится восстановление исходных значений переменных на момент отката.

[28] Приведенный алгоритм соответствует одиночному наследованию. Для множественного наследования следует модифицировать строку 12 таким образом, чтобы процедура Запрос применялась поочередно ко всем родительским фреймам, входящим в Parent(F).

Алгоритм 1.1: Алгоритм применения правил-запросов для реализации обратного вывода

Исходные данные: Слот S фрейма F, значение которого необходимо вычислить, базовый фрейм R или $\perp$.
Результат: Вычисленное значение слота. При этом также устанавливаются значения других слотов по мере логического вывода.

Запрос(F,S,[R])
(1) **if** $R = \perp$ **then** $R \leftarrow F$
(2) stage $\leftarrow 1$
(3) **foreach** $r \in \text{GetActions}(S_F)$ **using** $\sqsubseteq_{S_F}$
(4) { *обработать правило* r }
(5) **if** stage > 1 **and** r правило-присваивание **then continue**
(6) $b \leftarrow$ Вычислить($LHS(r), R$)
(7) **if** AsBoolean(b)
(8) $v \leftarrow$ Вычислить($RHS(r), R$)
(9) **if** r +=-правило **then** $x \leftarrow S_R + v$
(10) **else** $x \leftarrow v$
(11) **if** Проверить_ограничения(x) **then** $S_R \leftarrow x$; stage $\leftarrow 2$;
(12) **if** $S_R = \perp$ **then** $S_R \leftarrow$ Запрос(Parent(F), S, R)
(13) **if** $S_R \neq \perp$ **then** Прямой_вывод(R, S)

Rete-подобные алгоритмы, к которым относится предложенный Charles Forgy в работе [74] оригинальный алгоритм Rete, TREAT [90], Gator [78] и другие, основаны на построении по набору правил прямого вывода сети[29] определенной конфигурации, связанной с одной стороны с переменными рабочей памяти, а с другой — с выполняемыми при истинности правила действиями. В промежуточных узлах сети (так называемой α и β-памяти) сохраняется информация об истинности отдельных подвыражений левой части правила — таким образом, при изменении значения переменной происходит обновления только небольшой части связанной с ней сети. Это обновление в свою очередь может привести к активизации нескольких связанных с правилами действий — в этом случае к ним применяется правило разрешения конфликта, и одно из действий срабатывает.

Аналогичным образом алгоритмы семейства Rete могут быть применены к фреймовой системе. В этом случае правила прямого вывода при трансляции преобразуются в сеть Rete , в конечных узлах которой расположены, с одной стороны, процедуры-демоны, присоединенные к соответствующим слотам, входящим в левую часть выражения, а с другой — процедуры, выполняемые при истинности данного выражения (см. рис. 1.12). При присваивании слоту значения срабатывают процедуры-демоны родительских слотов, активизируя также все связанные с ними правила. Для правильной обработки наследования правилам необходимо также знать ссылку на фрейм, вызвавший активацию (базовый фрейм) — чтобы подставлять ее в качестве ссылки на текущий фрейм `this`. Таким образом, вместе с каждой активацией в сети Rete хранится также ссылка на базовый фрейм.

В инструментарии JULIA реализован модифицированный вариант алгоритма прямого вывода, напоминающий предложенный в [21] *быстрый алгоритм Rete* , при котором единая сеть для всех правил не строится. Такая модификация потребовалась для обес-

[29]Название алгоритма как раз происходит от латинского слова, означающего “сеть”.

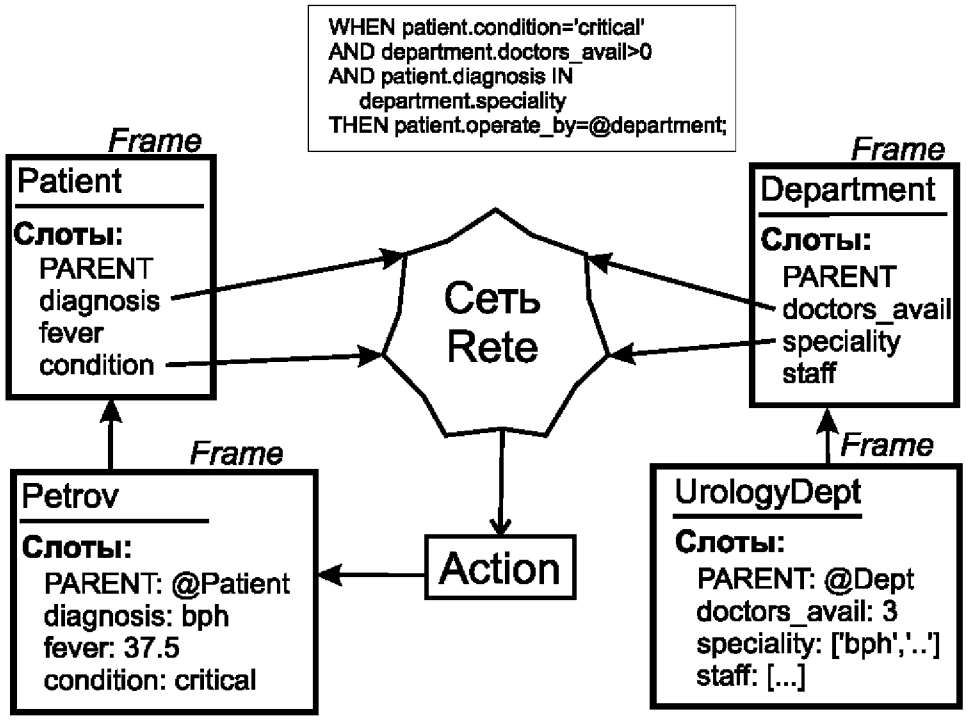


Рис. 1.12. Построение сети процедур-демонов для продукционного правила прямого вывода при использовании алгоритма Rete

печения возможности срабатывания правил в условиях неполной определенности — т.е. когда не все правила известны на момент присваивания. Как мы увидим в главе 2, такое свойство необходимо для распределенного вывода, так как база данных на одном узле может не иметь непосредственного доступа ко всем правилам, находящимся на другом.

В модифицированном алгоритме при изменении значения некоторого слота активизируются все связанные с ним процедуры-демоны, которые непосредственно *пытаются вычислить* выражения, находящиеся в левой части продукционных правил. При этом выражения (представленные соответствующими древовидными структурами в памяти) вычисляются по специальному алгоритму с запоминанием промежуточных результатов в узлах-операциях дерева, которые в данном случае играют роль β-памяти, применяемой в алгоритме Rete . Необходимость в α-памяти в используемом варианте алгоритма отсутствует (в качестве α-памяти используются сами слоты), так как унификация правила со значениями в рабочей памяти как таковая не производится, а заменяется неявной унификацией по наследованию, которая достигается за счет вызова процедур-демонов всех родительских фреймов с передачей текущего фрейма (вызвавшего активацию) в качестве контекста вызова.

Таким образом, сеть неявно образована присоединенными к слотам демонами, связанными с ними правилами и деревьями выражений, в узлах которых запоминаются промежуточные результаты вычислений (см. рис. 1.13).

Другим отличием предлагаемого алгоритма является предварительный выбор правил для срабатывания на основе применения стратегии выбора `RuleSelectionStrategy` к присоединенным процедурам-демонам. После выбора процедуры для применения проверяется на предмет возможности срабатывания связанное с ней правило, и если срабатывание правила ведет к модификации значений других слотов — рекурсивно активизируются соответствующие им процедуры-демоны, и только потом рассматриваются остальные

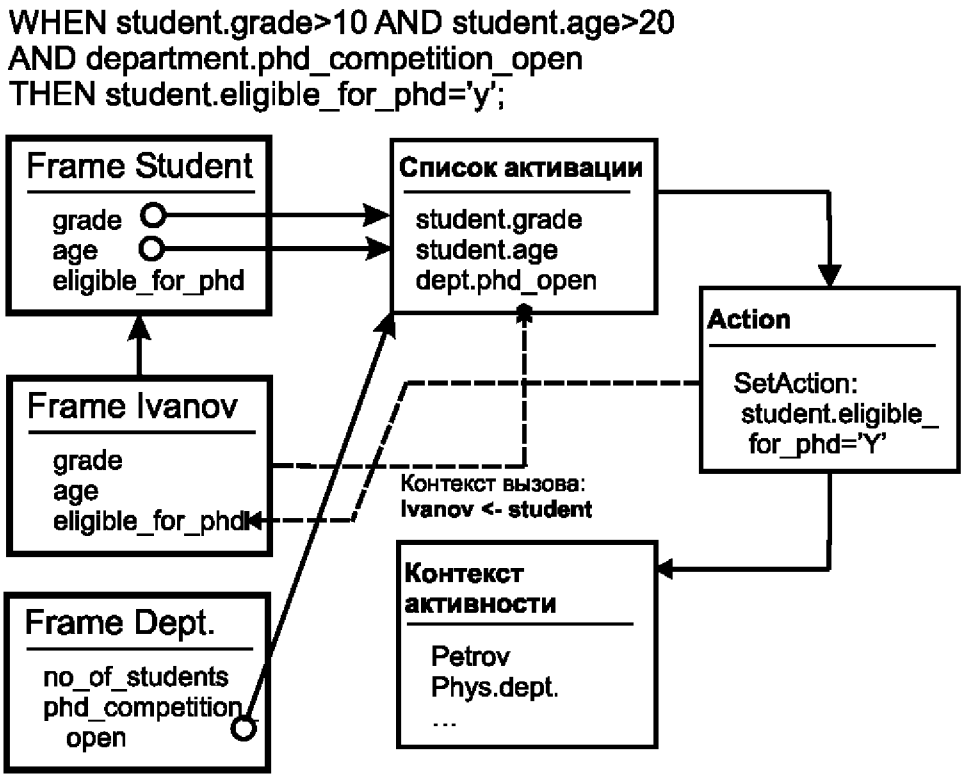


Рис. 1.13. Построение сети процедур-демонов для продукционного правила прямого вывода в библиотеке JULIA

процедуры-демоны исходного слота. Такой порядок срабатывания соответствует описанной выше в разделе 1.3.4.4 семантике. Завершимость рекурсии следует из доказанной корректности функции направленного прямого вывода, поскольку, как было показано при доказательстве утверждения 1.5, функция прямого вывода может быть построена как конечная композиция одношаговых функций. Кроме того, завершимость достаточно очевидна из-за конечности числа слотов и присоединенных процедур, учитывая, что одна и та же процедура не может сработать дважды (это обеспечивается запоминанием базового фрейма в т.н. **контекст активности** присоединенной процедуры).

Алгоритм 1.2 описывает рассмотренные выше действия более строго[30].

1.4.5.4. Комбинированный логический вывод

Основные операции, приводящие к запуску логического вывода — это запрос значения некоторого слота (при этом инициируются процедуры-запросы, реализующие обратный вывод) или присваивание слоту значения (в этом случае работают процедуры-демоны, отвечающие за прямой вывод). В случае, если база знаний состоит только из правил прямого вывода, логический вывод осуществляется по мере присваивания исходных значений слотам; после того, как все исходные значения присвоены, вывод завершается, и значение целевого слота должно быть получено.

Основным в системе является обратный вывод, который инициируется при запросе значения целевого слота и заканчивается получением его значения. В общем случае,

[30]Приведенный алгоритм соответствует одиночному наследованию. Для множественного наследования следует модифицировать строку 13 для рекурсивного вызова демонов для всех родителей, входящих в список parent(F).

Алгоритм 1.2: Алгоритм прямого вывода, основанный на применении процедур-демонов

Исходные данные: Слот S фрейма F, которому было присвоено значение, базовый фрейм R или $\perp$.

Результат: Полный процесс прямого вывода, вызванный данным присваиванием.

Прямой_вывод(F,S,[R])

(1) **if** $R = \perp$ **then** $R \leftarrow F$
(2) **foreach** $r \in \text{SetActions}(S_F)$ **using** $\sqsubseteq_{S_F}$
(3) { *обработать правило* r }
(4) **if** $r \in$ контекст активности слота S_R **then continue**
(5) $U \leftarrow$ множество слотов в $LHS(r)$
(6) $b \leftarrow$ **true**
(7) **foreach** $u \in U$
(8) $t \leftarrow$ Запрос(u,R)
(9) **if** $t = \perp$ **then** $t \leftarrow$ **false** ; **break**
(10) **if not** t **then continue**
(11) $t \leftarrow$ Вычислить($LHS(r)$)
(12) **if** t **then** Выполнить($RHS(r)$)
(13) **if** parent(F) $\neq \perp$ **then** Прямой_вывод(parent(F), S, R)

база знаний может содержать правила как обратного, так и прямого вывода — при этом рассмотренный ранее в разделах 1.4.5.2 и 1.4.5.3 механизм вывода (алгоритмы 1.1 и 1.2) соответствует применению исчерпывающего прямого вывода после каждого шага обратного вывода[31] в соответствии с рассмотренной в разделе 1.3.4 семантикой.

1.5. Интеграция недекларативных структур во фреймовую модель

При создании сложных интеллектуально-информационных систем возникает необходимость совместного использования интеллектуальной составляющей и императивного программного кода, элементов различных компонентных моделей (JavaBeans, COM/DCOM/ActiveX-компонентов, CORBA-объектов), а также реляционных и объектных баз данных. Такие компоненты могут реализовывать корпоративные хранилища данных (data warehouses), либо традиционную корпоративную бизнес-логику, и возможность их прозрачного включения во фреймовую модель была бы чрезвычайно привлекательной.

Среди перечисленных выше можно выделить следующие классы объектов:

- **Пассивные реляционные структуры**, представленные реляционными таблицами баз данных. Хотя такие структуры могут содержать в себе некоторый активный программный код, реализующий бизнес-логику (триггеры, встроенные процедуры), с внешней точки зрения данные всегда представляются реляционным отношением, возможно, динамически вычисляемым.

- **Пассивные** или **активные объектные структуры**, представленные набором (иерархией) именованных объектов или интерфейсов. К этому классу можно от-

[31] Точнее, после каждого присваивания, возникшего в результате обратного вывода

шести объектные базы данных и программные компонентные модели (ActiveX, CORBA, ...).

- **Смешанные структуры**, в которых попытка интеграции объектного и реляционного представления уже была сделана, и результирующая модель данных соединяет в себе черты обоих структур (например, СУБД PostgreSQL).

В следующих разделах мы покажем, что реляционные и объектные структуры имеют с фреймовой моделью много общего, и что несложными "топологическими" превращениями можно унифицировать эти структуры, превратив их в семейства фреймов, доступные обычным образом в процессе логического вывода.

1.5.1. Фреймовая модель и реляционные базы данных

При создании реальных интеллектуальных систем во многих случаях важно интегрировать интеллектуальную функциональность в существующую информационную систему, т.е. использовать уже накопленные данные как исходные для решения задачи. Так как в подавляющем большинстве случаев для хранения больших объемов данных используется реляционная модель, то особый интерес представляет интеграция инструментария с реляционными базами данных.

В области взаимодействия баз данных и знаний в настоящее время существуют два основных направления:

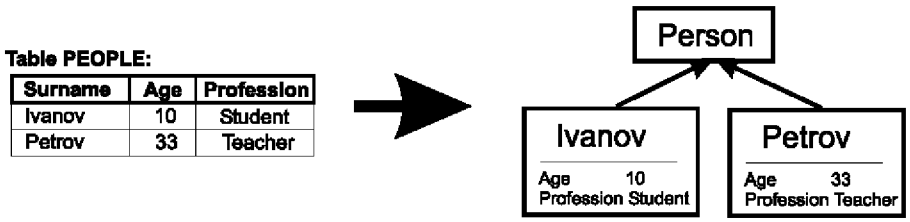
- **Активные базы данных** (active databases) [79], представляющие собой расширение реляционной модели за счет введения триггеров-правил, работающих по принципу прямого вывода.

- **Легковесные дедуктивные базы данных** (lightweight deductive databases) [94], в которых за основу берется логическая модель представления знаний, а реляционная модель данных "моделируется" многоместными предикатами-фактами. Соответственно процесс логического вывода как правило построен на методе резолюции с обратным выводом от цели к фактам.

Активные базы данных удобны для хранения больших объемов данных со сравнительно небольшими "интеллектуальными" вкраплениями, в то время как легковесные дедуктивные базы данных неэффективны на больших объемах данных (так как выборка данных происходит в соответствии с моделью логического программирования, т.е. на основе унификации, а не с использованием специальных методов индексирования), однако они представляют более широкие возможности для программирования интеллектуальной части. Соответственно, для обращения к активным базам данных используется стандартный для реляционных баз данных язык запросов (SQL), а для дедуктивных баз данных язык запросов логического программирования.

1.5.1.1. Реализация доступа к РСУБД в инструментарии JULIA

В данной работе предлагается промежуточный подход, при котором реляционная база данных используется для задания некоторых атрибутов (исходных данных) решаемой задачи в виде значений слотов фреймов и семейств фреймов специального вида, которые затем используются в процессе логического вывода обычным образом, позволяя




```
FRAME PERSON {                    FRAMESET
 SCALAR AGE INT,                   DATABASE people
 SCALAR PROFESSION                 PARENT PERSON;
 };
```

Рис. 1.14. Пример экстенсионала (фрейм-класса) для доступа к реляционной БД

таким образом делать интеллектуальные заключения относительно содержащихся в базе данных сведений. В рамках инструментария JULIA это реализуется одним из следующих способов:

- **Экстенсионал или фрейм-класс реляционной таблицы** позволяет представить реляционное отношение $R = \langle R_1, \ldots, R_n \rangle$ или проекцию в качестве семейства фреймов с общим родителем. Некоторый ключевой атрибут отношения R_i выбирается в качестве **именующего** атрибута[32], в соответствии с которым именуются фреймы, соответствующие отдельным строкам отношения. Слоты фреймов именуются в соответствии с метаданным именами столбцов отношения. Такое преобразование реляционной таблицы в множество фреймов будем называть **горизонтальным срезом** таблицы.

  Пример фрейм-класса приведен на рис.1.14. Фрейм-класс представляется во фрейм-мире одним Java-классом, унаследованным от `FrameClass`, к которому направляются все запросы на значения слотов, принадлежащих одному из фреймов данного класса. При получении запроса либо создается экземпляр соответствующего фрейма (унаследованный от `DBFrame`), и его слоты заполняются значениями из базы данных (т.н. **кеширующий** фрейм-класс), либо все запросы каждый раз направляются к базе данных (в этом случае экономится память, что существенно при обработке больших массивов данных). В любом случае, при присвоении значения слоту фрейма производится модификация соответствующего атрибута в базе данных (при его наличии), либо запоминание этого атрибута в памяти (эта функциональность достигается путем наследования `DBFrame` от `StoreFrame`).

  Хотя фрейм-класс не позволяет напрямую связывать со слотами правила прямого или обратного вывода, такие правила могут определяться для родительского фрейма. В этом случае при запросе незаполненного атрибута фрейм-класса применяются правила (прямого и обратного вывода), определенные для соответствующего слота фрейма-родителя. В зависимости от опций системы, правила прямого вывода могут срабатывать только при явном запросе значений соответствующих

[32] В этом случае предполагается, что у отношения есть единственный ключевой атрибут. Хотя такое ограничение во многих случаях не является слишком сильным, теоретически возможно использовать несколько атрибутов в качестве именующих, формируя имя фреймов конкатенацией значений соответсвующих атрибутов

слотов фрейм-класса, либо при создании фрейм-класса может производиться просмотр всех входящих в него фреймов. Последнее может оказаться весьма длительным процессом, и менее эффективным по сравнению с применяемыми в активных базах данных алгоритмами Gator [78], TREAT [90] или RETE [74], однако неэффективность возникает по той причине, что интеллектуальная составляющая системы в нашем случае работает во времени независимо от реляционной, и такой предварительный просмотр множества данных необходим для того, чтобы обеспечить "синхронизацию" значений слотов во фреймовой модели с данными, т.е. отложенным образом применить все правила прямого вывода, которые возможны для набора данных.

Для выполнения операции выборки, т.е. поиска строк реляционной таблицы, удовлетворяющих определенным ограничениям (в том числе по интеллектуальным атрибутам), используется процедура вычисления выражения на множестве потомков родителя фрейм-класса (`MAP` или `subclass`). С точки зрения эффективности используемый линейный поиск уступает активным базам данных и сравним с перебором, используемым в классическом логическом программировании, но при этом имеется возможность применения обратного вывода в процессе поиска, совместно с прямым выводом.

- **БД-фрейм**, или фрейм-выборка служит для явного представления одной или нескольких строк реляционной таблицы, удовлетворяющих некоторому явно заданному в виде SQL-выражения условию c. При выборке значения определенного слота s результат определяется SQL-запросом `SELECT` s `FROM table WHERE` c (если возвращаются несколько строк таблицы с различными значениями — результатом будет список), а при присваивании выполняется обновления всех строк таблицы с данным условием `UPDATE table SET` $s = v$ `WHERE` c. Полученный таким образом фрейм будем называть **горизонтальным фактор-срезом** таблицы, так как он характеризует множество строк таблицы, выделенных некоторым отношением эквивалентности, порожденным выражением c.

- **Вертикальный срез**, в некотором смысле противоположный горизонтальному, когда фреймы порождаются столбцами таблицы, а значения в строках заполняют слоты в соответствии с некоторым именующим столбцом. Такой вариант доступа ввиду его малой практической применимости в инструментарии не реализован, и рассматриваться не будет.

- **Другие средства доступа к БД**. Процедуры и демоны для доступа к БД позволяют указать в качестве присоединенной процедуры (запроса или демона) некоторое SQL-предложение [33] или (в более простой форме) имя таблицы и атрибута. Кроме того, инструментарий содержит библиотеку функций для доступа к РСУБД, позволяющую получать значения из таблиц базы данных в составе произвольного выражения, например:

```
IMPORT LIBRARY 'com.shwars.julia.Libraries.DB';
SET Person.Children =
```

[33] Аналогично рассмотренному выше, в случае, если SQL-запрос применим к нескольким строкам таблицы, при запросе возвращается список значений, а при присвоении производится полное обновление

```
    &DB_SELECT("SELECT Name FROM Person WHERE Person.ID = " + Person.DB_ID);
WHEN Person.Deceased THEN
    &DB_UPDATE("DELETE FROM PERSON WHERE Person.ID = " + Person.DB_ID);
```

1.5.1.2. Семантика доступа к реляционным структурам

Для определения процесса доступа к внешним реляционным структурам нам потребуется ввести понятие фрейм-класса, а также функции доступа $\mathfrak{R}$ для получения отдельных компонентов реляционного отношения $\mathcal{R}$.

Определение 1.2. Фрейм-классом будем называть функцию $\hat{\mathcal{F}} : \mathbb{I} \to \mathcal{F}$, возвращающую фрейм-определяющую функцию по имени принадлежащего фрейм-классу фрейма, либо $\perp$.

Понятие фрейм-класса можно расширить, допустив применение его к состоянию $\mathcal{W}$ следующим образом:

$$\hat{\mathcal{F}}(\mathcal{W}) = \lambda f. \begin{cases} \hat{\mathcal{F}}(f), & \hat{\mathcal{F}}(f) \neq \perp \\ \mathcal{W}(f), & \text{в противном случае} \end{cases} \tag{1.29}$$

Таким образом, применив некоторый фрейм-класс к текущему состоянию, получим новое состояние, в котором все фреймы данного фрейм-класса будут доступными.

Определение 1.3. Пусть $\mathcal{R} = \{\langle x_0^{(i)}, \ldots, x_n^{(i)} \rangle\}_{i=1}^{N}$ $n+1$-местное реляционное отношение с метаданными (именами столбцов) $\langle \bar{x}_0, \ldots, \bar{x}_n \rangle$. Без ограничения общности будем полагать x_0 ключевым атрибутом. Тогда соответствующую этому отношению функцию доступа определим следующим образом:

$$\mathfrak{R}_{\mathcal{R}}(f, s) = \begin{cases} x_j^{(i)}, & \text{если } \exists j : \bar{x}_j = s, \quad \exists i : x_0^{(i)} = f \\ \perp, & \text{в противном случае} \end{cases} \tag{1.30}$$

Тогда соответствующий горизонтальному срезу отношения $\mathcal{R}$ фрейм-класс можно будет определить как:

$$\hat{\mathcal{F}} = \lambda \langle f, s \rangle . \mathfrak{R}_{\mathcal{R}}(f, s) \tag{1.31}$$

Определение вертикального среза будет производиться аналогично, где вместо функции $\mathfrak{R}$ будет использоваться $\mathfrak{R}'$, определенная как

$$\mathfrak{R}'_{\mathcal{R}}(f, s) = \begin{cases} x_j^{(i)}, & \text{если } \exists j : \bar{x}_j = f, \quad \exists i : x_0^i = s \\ \perp, & \text{в противном случае} \end{cases} \tag{1.32}$$

Для получения фактор-среза по некоторому SQL-выражению c определим функцию-оракул (oracle function) $\mathfrak{R}_{\mathcal{R}}(c)$, применяющую к отношению $\mathcal{R}$ SQL-выражение c и инкапсулирующую в себе всю функциональность РСУБД и семантику соответствующих SQL-предложений. Тогда в качестве БД-фрейма будет выступать фрейм-функция специального вида, интерпретация которой будет задаваться следующим образом:

$$\|f.s\|_{\mathcal{W} \to \mathcal{W}} = \mathfrak{R}_f(\texttt{SELECT } s \texttt{ FROM } f \texttt{ WHERE } \mathcal{W}(f, s).value) \tag{1.33}$$

В данном случае предполагается, что в качестве значения функция БД-фрейма возвращает SQL-подвыражение для фактор-среза.

Операция присваивания значения *write* будет определяться в этом случае как

$$\mathcal{W}[f.s \leftarrow v] = \mathcal{W}, \text{ и } \mathfrak{R}_f(\texttt{UPDATE } f \texttt{ WHERE } \mathcal{W}(f,s).value) \tag{1.34}$$

В этом случае состояния $\mathcal{W}$ и $\mathcal{W}[f.s \leftarrow v]$ будут формально совпадать — изменения коснутся лишь внешнего отношения, инкапсулированного в поведении функции-оракула[34]. Поэтому в случае применения функций-оракулов следует быть осторожным при изучении поведения логического вывода с помощью графов, так как два принципиально различных состояния могут оказаться неразличимыми.

1.5.2. Интеграция компонентных и объектных моделей с фреймовым представлением знаний

Во многих случаях при построении гибридных систем оказывается полезным иметь возможность расширить базу знаний процедурными включениями на императивных языках программирования. Такая возможность предусмотрена во многих существующих системах (CLIPS, JESS), однако, как правило, она представлена возможностью определения пользовательских функций на базовом языке (C—+ или Java).

Фреймовая модель позволяет использовать императивные включения более широко, так как объектные и компонентные модели по своей сути весьма похожи на фреймы. В общем виде объект или класс представляет собой набор атрибутов и методов, при этом обычно делается различие между классом, определяющим только функциональность в виде набора методов и собственно объектом, содержащим конкретные значения атрибутов и использующим функциональность родительского класса [3, 10].

Фреймовый подход унифицирует понятие класса и объекта, а также атрибута и метода, так как слот фрейма может содержать как данные, так и применяемые к ним процедуры. Таким образом, возникает естественное представление классов и компонентов в виде фреймов — надо лишь оговорить правила трансляции имен атрибутов и методов. Поскольку методы во фреймовом подходе не могут быть вызваны явно, то они преобразуются в соответствии с некоторым правилом к слотам, при запросе или присваивании значений которым и срабатывают процедуры внешнего, ассоциированного с фреймом, объекта. Аналогичный метод представления распространяется на компоненты в той или иной компонентной модели (JavaBeans, CORBA и др.), так как они по сути дела являются наборами методов без атрибутов — т.е. частным случаем объектной модели.

1.5.2.1. Расширение фреймовой модели императивным программным кодом

Таким образом, инструментарий JULIA представляет целый спектр возможностей по интеграции императивных включений во фреймовую модель:

- **Представление класса или компонента фреймом** (в соответствии с рассмотренными выше принципами). При этом во фреймовой иерархии такой фрейм представляется заглушкой (stub), которая при обращении к слотам вызывает соответствующие процедуры присоединенного экземпляра класса или компонента. Для определения атрибута с именем `X` в классе или компоненте должны присутствовать функции `getX` и `setX`, либо переменная с именем `theX`. Фрейм автоматически

[34] В случае с горизонтальным срезом таких сложностей не возникает, так как само отношение оказывается неявно сохраненным в состоянии $\mathcal{W}$ после применения фрейм-класса.

снабжается соответствующими слотами, исследуя класс или компонент при помощи Java Reflection API, либо репозитария интерфейсов CORBA. Использование соглашения об именовании, принятого в JavaBeans, позволяет использовать JavaBeans как фреймы без специального дополнительного программирования.

- Определение **функций пользователя** позволяет описывать на Java наборы тематических функций, присоединяемых к инструментарию динамически в виде библиотек. Такая библиотека должна быть унаследована от класса `Library`, и по сути дела представляет собой набор статических функций вида `TAny` *func*`(TList L)`, где в виде единственного аргумента функции передается список аргументов, с которыми была вызвана соответствующая функция во фреймовой модели[35].

- Создание **пользовательских стратегий выбора правил** при прямом и обратном выводе позволяет расширять имеющийся в инструментарии стандартный набор стратегий путем реализации соответствующего класса на базовом языке программирования (Java).

- Реализация произвольных **процедур-демонов и процедур-запросов** на базовом языке программирования путем наследования от соответствующих суперклассов инструментария (`TAction` и `TAnyAction`).

Следует также отметить, что поскольку инструментарий имеет открытую архитектуру и представляет собой набор Java-классов, то в принципе практически любой его компонент может быть расширен и дополнен за счет наследования от соответствующего Java-класса. Однако, рассмотренные выше способы расширения продукционно-фреймовой модели произвольным программным кодом в подавляющем большинстве случаев являются достаточными для приложений. Возможность тесного взаимодействия произвольного программного кода с интеллектуальным компонентом, управляемым знаниями, позволяет строить сложные интеллектуально-информационные системы для решения комплексных задач. Инструментарий, в частности, содержит ряд расширений, предназначенных для выполнения ряда часто встречающихся задач, например получения и анализа информации из Интернет, организации интерфейса с браузером при использовании библиотеки в качестве апплета, динамической генерации вопросов по шаблону и др.

1.5.2.2. Семантика доступа к внешним объектам при логическом выводе

Для представления внешних объектов необходимо также использовать функцию-оракул $\mathfrak{E}$ для моделирования процесса выполнения внешнего программного кода. Определим ее следующим образом:

$$\mathfrak{E}(o.x) = \begin{cases} o.x, \text{ если } x \quad \text{атрибут объекта } o \\ \text{результат выполнения } o.x, \text{ если } x \quad \text{метод объекта } o \\ \bot, \text{ если объект } o \text{ не содержит метода или атрибута } o \\ \quad \text{или если возникла ошибка времени выполнения} \end{cases} \tag{1.35}$$

Тогда некоторому объекту o будем ставить в соответствие фрейм-функцию F_o специального вида $F_o = \lambda s.\mathfrak{E}(o.s)$, для которой правило интерпретации будет иметь вид

$$\|f.x\|_{\mathcal{W}\to\mathcal{W}} = \|F_o(s)\| = \mathfrak{E}(o.s), \text{ где } F_o = \mathcal{W}(f,s) \tag{1.36}$$

[35] Таким образом автоматически осуществляется поддержка функций с переменным числом аргументов

а функция *write* будет определяться следующим образом:

$$\mathcal{W}[f.s \leftarrow v] = \begin{cases} \mathcal{W}[f.s \leftarrow v], \text{ и } \mathfrak{E}(f.s), \text{ если } s \quad \text{метод} \\ \mathcal{W}[f.s \leftarrow v], \text{ и } \mathfrak{E}(f.s := v), \text{ если } s \quad \text{атрибут} \end{cases} \tag{1.37}$$

1.6. Особенности логического вывода в инструментарии

1.6.1. Динамическое наследование

Одной их особенностей реализованной в инструментарии фреймовой модели является возможность динамического вычисления значения слота `parent` наравне с другими слотами фрейма, что позволяет формировать иерархию наследования "на лету", в процессе логического вывода. Такую функциональность будем называть **динамическим наследованием**.

С помощью динамического наследования, в частности, реализуется такая характерная для фреймовых систем операция, как **спецификация** фрейма, т.е. определение места некоторого фрейма в иерархии в соответствии со значениями некоторых его слотов на основании определенных для релевантных фреймов ограничений. Эта операция реализуется встроенной функцией `match(Frame root, Frame X)`, которая производит обход иерархии в ширину, начиная с фрейма `root`, и сравнение фрейма `X` на предмет удовлетворения ограничениям. В качестве результата возвращается ссылка на фрейм, удовлетворяющий ограничениям. Таким образом, добавление правила

SET `f.parent =` **match**`(p, f)`

приводит к тому, что для фрейма `f` (и для всех его потомков) будет автоматически производиться поиск родителя в соответствии с частично-определенными значениями слотов начиная с фрейма `p`. Конструкция `FRAME f DYNAMIC PARENT p AUTO`, используемая при описании фрейма, автоматически добавляет указанное выше правило.

Помимо операции спецификации, возможно явное указание продукционных правил для определения родителя фрейма. В этом случае при описании фрейма надо указать ключевое слово `DYNAMIC PARENT` без ключа `AUTO`, а в теле базы знаний использовать правила вида `IF ... THEN f.parent = g`.

Динамическое определение родителя инициируется тогда, когда возникает потребность применения наследования, т.е. если например в процессе вывода требуется вычисление значения родительского слота, либо сработало некоторое правило прямого вывода. После определения родителя значение ссылки запоминается в слоте `parent` и далее используется без повторного вычисления в соответствии с общими правилами монотонного вывода.

1.6.2. Мета-правила

Обычно под **мета-знаниями** [19] понимают знания относительно манипулирования основными знаниями в интеллектуальной системе. Мета-знания, представленные **мета-правилами**, могут указывать или изменять стратегию выбора правил, изменять работу встроенного арбитра разрешения конфликта [14], критерии эвристического поиска и т.д.

В нашем случае мы будем понимать под мета-правилами действия, которые могут совершаться при наступлении определенных событий в процессе логического вывода и которые могут изменять характеристики этого вывода. Например, мета-правила могут быть определены для следующих ситуаций:

- В процессе обратного вывода все правила для данного слота были исчерпаны.
- В процессе обратного вывода применились все правила для родителя
- Между применениями отдельных правил для слота

Мета-правила представляют собой правила-демоны, унаследованные от `TAction`, и могут изменять значения произвольных слотов и совершать другие дополнительные действия, которые могут изменять процесс вывода:

- Установить/изменить для слота стратегию выбора правил
- Повторно применить логический вывод для некоторого слота
- Повторно применить логический вывод для родительского слота

Таким образом, мета-правила могут нарушать монотонность вывода, и поэтому строгое рассмотрение их функциональности представляет некоторые затруднения. В частности, значительно сложнее сформулировать и доказать критерии завершимости логического вывода для базы знаний, содержащей мета-правила. Однако, использование мета-правил в некоторых случаях позволяет проще решить некоторые практические вопросы, возникающие при проектировании интеллектуальных систем, а также расширить выразительные возможности языка представления знаний.

1.6.2.1. Псевдо-множественное (поочередное) наследование

В инструментарии JULIA может быть реализовано как одиночное, так и множественное наследование. При множественном наследовании слот `parent` может представлять собой список ссылок на множество родительских фреймов, и в случае необходимости вычисления значения слота исходя из наследуемых правил процедура вывода будет запускаться поочередно для всех родительских фреймов, пока требуемое значение не будет получено (см. алгоритм 1.1 и комментарий к нему на стр. 53). Соответственно при прямом выводе и при проверке ограничений каждый раз будет рассматриваться целое множество родителей.

Как альтернативу полноценному множественному наследованию предлагается рассмотреть **псевдомножественное наследование**, при котором в каждый момент времени фрейму будет соответствовать только один родитель, но в процессе вывода ссылка на родителя будет меняться, тем самым обеспечивая при обратном выводе функциональность, эквивалентную множественному наследованию. При этом даже будет обеспечиваться большая гибкость — за счет возможности менять родителя по некоторому более сложному алгоритму, нежели простым перебором значений в списке.

Значение ссылки на родительский фрейм в процессе вывода может меняться различными способами:

- При помощи мета-правил, срабатывающих при завершении исчерпывающего прямого вывода или при неуспешном завершении обратного вывода для данного слота
- Из внешнего Java-приложения или специализированной компоненты инструментария, которая в явном виде создаст фрейм-экземпляр, присоединяет его к различным иерархиям, запускает вывод и проверяет получение результата

- При помощи правил прямого вывода с использованием немонотонного присваивания значения слоту `Parent` фрейма-экземпляра

В общем случае возможны также различные **стратегии** поочередного наследования, например, однократное последовательное наследование, многократное круговое наследование (round-robin) до получения требуемого результата (т.е. пока не будет установлено значение целевого слота, либо за один цикл не будет изменено значение ни одного атрибута), поочередное наследование со статическими (явно заданными) или динамическими (определяемыми в процессе вывода по некоторому критерию) приоритетами и др.

1.7. Программные и пользовательские интерфейсы инструментария

Инструментарий JULIA предоставляет законченный набор инструментов для создания экспертных систем, функционирующих как в диалоговом режиме, так и в составе более крупных программных комплексов. В частности, в инструментарии реализованы следующие виды интерфейсов для доступа к нему со стороны внешних программных систем или пользователей:

- Программный интерфейс **JULIA API** для доступа к функциям библиотеки JULIA из программных систем, реализованных на Java. Использование этого интерфейса открывает доступ ко всем функциям системы, а также позволяет перепрограммировать ее отдельные части путем наследования от классов инструментария (см. рис. 1.8). Простейшая последовательность вызовов для запуска интегрированной консультации из Java-программы выглядит следующим образом:

```
import com.shwars.julia.*;
...
//  Загрузка сериализованного представления БЗ
World W = World.load(new FileInputStream("file.jsw"));

//  Установка объекта для callback-вызова при обратном выводе
W.setAsker(this);

//  Запуск логического вывода
TAny T = W.getClusterManager().get("frame").get("slot");
```

- Мульти-языковый компонентный программный интерфейс **JULIA CORBA API** позволяет программным системам получать доступ к инструментарию, запущенному на некотором узле сети в качестве сервера баз знаний, с помощью набора CORBA-интерфейсов. В текущей реализации CORBA-интерфейс позволяет инициировать логический вывод на JULIA-сервере с возможностью callback CORBA-вызовов для уточнения данных в процессе обратного вывода.

- Пользовательский интерфейс (см. рис. 1.15), позволяющий проводить консультацию, используя JULIA как оболочку экспертной системы. В текущей реализации реализованы пользовательские интерфейсы в виде Java-апплета и Java-приложения, а также Web-интерфейс в виде сервлета, выполняющегося на сервере.

Кроме того, предусмотрен более развитый интерфейс в виде приложения, позволяющий просматривать и модифицировать структуру фреймовой иерархии.

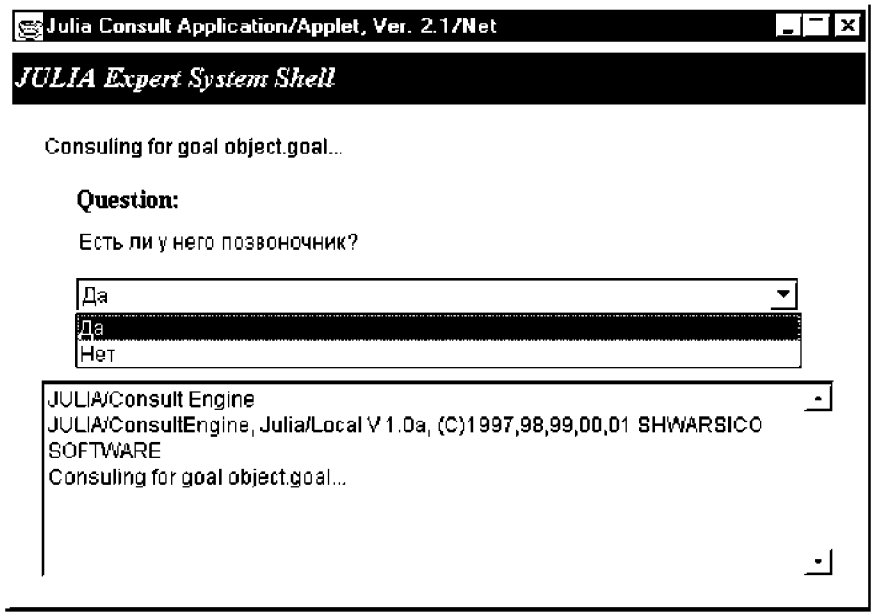


Рис. 1.15. Пример пользовательского интерфейса инструментария JULIA в виде Java-приложения.

Таким образом, возможно использовать инструментарий JULIA в следующих режимах:

- **В диалоговом режиме** с использованием графического интерфейса или интерфейса командной строки (см. рис. 1.6) инструментарий может использоваться как независимая оболочка для построения экспертных систем.

- Реализованный на Java графический интерфейс инструментария может использоваться в виде **апплета** для построения веб-приложений удаленной консультации. При этом часто оказывается удобным использовать загрузку правил по мере необходимости (см. раздел 2.5.4).

- **Вызов инструментария из произвольного Java-приложения** с использованием разработанного программного интерфейса (API). Исходный текст базы знаний, представленный на JFMDL, может быть предварительно преобразован во внутреннее сериализованное представление, которое затем может быть легко загружено приложением по любому потоку (с диска, через Интернет-соединение по протоколу HTTP и др.) и использовано для проведения логического вывода. В простейшем случае для осуществления вывода по сериализованной базе знаний требуется две-три строки Java-кода.

- Сервер или группа серверов на базе JULIA могут вызываться по протоколу **CORBA/IIOP**, для чего разработан специальный интерфейс. Таким образом возможно использование среды распределенного накопления и использования знаний из программных систем, реализованных на различных языках программирования.

- Для языков и систем программирования, в которых нет поддержки CORBA (например C, ранние версии Borland Delphi), на базе Borland Visibroker for C++ разработана DLL и соответствующие библиотеки, реализующие все механизмы удаленного доступа. В частности, могут быть реализованы компоненты (COM, Delphi, ActiveX, JavaBeans и другие) для доступа к JULIA из различных программных сред.

- Доступ к интеллектуальной системе может также осуществляться через **веб-интерфейс**, реализованный в виде сервлета на сервере или в виде апплета в клиентском браузере, который получает доступ к удаленному JULIA-серверу, либо реализует логический вывод локально с загрузкой фреймовой иерархии по протоколу HTTP.

- Наконец, доступ к серверу на базе JULIA может осуществляться со стороны других JULIA-серверов, использующие знания, содержащиеся на данном сервере, путем наследования от определенных фреймов с образованием распределенной фреймовой иерархии.

1.8. Выводы к главе 1

В данной главе была предложена архитектура и рассмотрены вопросы реализации инструментария JULIA для построения интеллектуальных систем на основе продукционно-фреймового представления знаний. Продукционно-фреймовое представление с поддержкой прямого и обратного логического вывода позволяет решать широкий класс задач, так как декомпозиция предметной области в виде иерархии понятий является универсальным и эффективным способом представления знаний о ее структуре и динамике. Для представления знаний разработан специализированный язык JFMDL, синтаксис которого приведен в приложении А.

Реализация инструментария в виде открытого набора классов на языке Java с богатым набором программных и пользовательских интерфейсов привлекательна не только в виде автономной оболочки экспертных систем, но и в составе различных программных и информационно-интеллектуальных систем в качестве встраиваемой интеллектуальной компоненты.

В инструментарии реализованы следующие оригинальные концепции:

- Открытая архитектура, позволяющая надстраивать и расширять функциональность за счет реализации на базовом языке программирования дополнительных библиотек функций, стратегий выбора правил, присоединенных процедур, фреймовых структур, типов данных и других компонентов — таким образом, возможности расширения и приспособления инструментария для решения конкретных задач практически неограничены.

- Интеграция с реляционными базами данных и компонентными моделями CORBA и JavaBeans, позволяющая представлять реляционные таблицы и объекты с произвольным программным кодом в виде фреймов и семейств фреймов.

- Динамическое и псевдо-множественное наследование.

Для используемой в инструментарии модели логического вывода определяется строгая формальная операционная семантика на базе двухуровневой трансляции предложений языка представления знаний в формальную модель, представляемую функцией состояния $\mathcal{W}$, и затем определения для нее функции семантической вычислимости $\|\cdot\|$. В этой модели показывается существование семантики неподвижной точки, однозначно определяющей результат комбинированного логического вывода для начального состояния фреймовой системы и множества входных атрибутов.

2. Построение распределенных интеллектуальных систем на основе фреймового представления знаний

Исследование подходов к созданию **распределенных интеллектуальных систем** в настоящее время особенно актуально, поскольку такие системы открывают путь к распределенному хранению и совместному использованию знаний, что необходимо для эффективной организации больших массивов данных в глобальных компьютерных сетях в эпоху информационного взрыва.

В данном разделе предлагается один из подходов к построению распределенных интеллектуальных систем на основе **распределенной фреймовой иерархии**, рассматривается место такой модели в контексте агентных архитектур. Отмечается ряд отличий предлагаемой архитектуры от общего агентного подхода, акцентирующего внимание на разделении внутреннего и внешнего представления знаний, что делает такую модель привлекательной для решения ряда задач по распределенному хранению и использованию знаний в процессе распределенного вывода. Ключевое для построения распределенной фреймовой иерархии **удаленное наследование** может реализовываться двумя альтернативными способами: перемещением по сети удаленного внутреннего представления фрейм-мира с последующим выводом на локальной ЭВМ (**мобильное наследование**), либо удаленным вызовом (с возможными callback-вызовами) с происходящим на удаленной ЭВМ логическим выводом (**статическое наследование**).

Для формализации рассматриваемой модели на множестве функций состояния вводится ряд дополнительных операций: объединения ($\bigstar$), подчиненного наследования ($\blacktriangleleft/\triangleleft$) и симметричного соединения ($\bowtie$). Для этих операций определяются две различные семантики выполнения в распределенной среде: мобильное наследование, связанное с передачей по сетевым каналам внутреннего представления фрейм-мира (т.е. удаленной функции $\mathcal{W}$), либо с удаленным вызовом для запроса значения.

2.1. Архитектуры распределенных интеллектуальных систем

Значительные исследования в области искусственного интеллекта в последние годы связаны с построением распределенных интеллектуальных систем, т.е. систем, состоящих из взаимодействующего набора более элементарных составляющих $\{p_1, \ldots, p_n\}$, которые совместно служат для решения некоторой комплексной интеллектуальной задачи. Построение распределенных систем особенно актуально в связи с растущей популярностью всемирной сети Интернет и, соответственно, с растущими объемами слабоструктурированной информации, а также с все возрастающими потребностями пользователей применительно к глобальной информационной магистрали. Развиваемый в настоящее время консорциумом W3C проект Semantic Web [64] в значительной степени использует достижения искусственного интеллекта именно в плане эксплицитного представления знаний. Параллельно с ним развиваются многочисленные проекты интеллектуального поиска по текстовым массивам данных [84] и по специальным образом аннотированным ресурсам [33, 46]. Кроме того, есть надежда, что объединение большого числа интеллектуальных систем в единую сеть позволит перейти на качественно новый уровень в процессе

создания истинно интеллектуальных систем (*Strong AI*) по принципу перехода количества в качество[1].

2.1.1. Простейшие клиент-серверные модели удаленного вывода

Первыми попытками построения распределенных интеллектуальных систем следует считать распределенные клиент-серверные архитектуры, реализующие идею удаленной консультации [23], т.е. удаленного вызова сервера баз знаний для решения некоторой задачи пользователя. Удаленная консультация может быть целесообразна в ряде случаев:

- Использование более мощного сервера вывода, нанодобие сервера баз данных, для повышения эффективности вывода[2].
- В качестве способа передачи или тиражирования знаний в машинном представлении в компьютерных сетях.

В последнее время удаленные консультации приобретают особую актуальность благодаря распространению Интернет и, соответственно, появлению задач, требующих передачи по сети или удаленного применения знаний. Как правило, имеет смысл различать два вида удаленных консультаций [99]: с логическим выводом на стороне клиента (в этом случае по сети передается вся база знаний с динамическими правилами, или по крайней мере ее часть, требуемая для вывода) и с логическим выводом на стороне сервера (по сети передаются статические знания о состоянии задачи, которые уточняются в процессе вывода). В первом случае говорят о модели **толстого клиента**, во втором **тонкого**.

Удаленные консультации весьма часто применяются для решения задач, однако соответствующие двухуровневые клиент-серверные модели в настоящее время вытесняются более мощными моделями многоагентного взаимодействия и предлагаемой в данной работе распределенной фреймовой иерархией, в которых удаленная консультация является частным случаем более общего подхода к распределению знаний.

2.1.2. Агентные архитектуры

Среди распределенных интеллектуальных систем наибольшее распространение получили так называемые **многоагентные системы** (иногда называемые **мультиагентными**, multi-agent systems) [24, 80, 91], в которых вся интеллектуальная система строится из составных частей, называемых **агентами**, с определенными свойствами. Понятие многоагентной системы чрезвычайно широко, поэтому зачастую многие совершенно непохожие друг на друга архитектуры включаются в это семейство. Противоположным подходом является предложенная автором в [49, 99] **компонентная архитектура**, в которой составные части одной интеллектуальной системы разделяются между различными узлами сети. Эта архитектура будет рассмотрена более подробно в разделе 2.1.3.

[1] С точки зрения современных философско-кибернетических учений [125] это является весьма возможным на основании принципа метасистемного перехода.

[2] В отличие от серверов баз данных, в серверах баз знаний очевидно будет иметь смысл делать акцент на эффективности выполнения процесса вывода, т.е. основное значение приобретает вычислительная мощность серверной ЭВМ.

2.1.2.1. Классификация агентных архитектур

В классической агентной архитектуре агенты, являющиеся основными взаимодействующими частями системы, удовлетворяют следующим основным свойствам[3] [91]:

- **Автономность** — агент должен быть в состоянии выполнять некоторые осмысленные действия без участия человека или других компонентов среды.
- **Кооперативность** — агент должен быть в состоянии взаимодействовать с другими агентами и работать в составе некоторого **агентного сообщества**.
- **Обучаемость** — агент обычно поддерживает внутри себя некоторую модель внешнего мира, которая совершенствуется в процессе взаимодействия с окружением и с другими агентами.

Рис. 2.1. Классификация многоагентных систем (заимствовано из [91])

Как правило, у каждого агента присутствует как минимум два из приведенных здесь свойств[4]. В зависимости от комбинации этих свойств различают (см. рис. 2.1) **коллаборативные** (collaborative), **интерфейсные** (interface) и **"смышленые"** (smart) агенты.

Агенты в многоагентной системе работают в некоторой среде, которая называется **окружением** (environment). Агенты выполняют некоторые **действия** (actions) в ответ на наблюдаемые изменения в окружении (perceptions), которые включают в себя запросы со стороны других агентов или пользователя.

Не вдаваясь в подробности классификации агентов (которые изложены в [91]), отметим следующие типы агентов:

- В зависимости от уровня представления знаний в каждом агенте различают **интеллектуальные** или **делиберативные** (deliberative) и **реактивные** (reactive) агенты [14]. Интеллектуальный агент содержит в себе некоторое символьное представление знаний об окружении, и его действия основаны на процессе символьного логического вывода (т.е. такой агент представляет собой самостоятельную интеллектуальную систему в терминах, изложенных в разделе 1.1). В реактивных агентах механизм отклика на изменения в окружении упрощен (представляется таблицей или некоторым алгоритмом), а интеллектуальные свойства агентного сообщества достигаются за счет взаимодействия большого количества таких агентов.

[3]Строго говоря, понятие агента у различных исследователей различается. В частности, указанные свойства не являются необходимыми и достаточными, но большинство из них действительно имеют место в агентных системах.

[4]Требование наличия минимум двух свойств может быть положено в определение понятие агента, как это сделано в [91].

- В зависимости от того, выполняется ли программный код агента на одной или различных ЭВМ, различают **мобильные** (mobile) агенты (программный код агента в процессе выполнения может перемещаться между различными узлами сети) и **статические** агенты (кооперативность ограничивается только обменом сообщениями).

2.1.2.2. Онтологическая совместимость агентов и таксономическая концептуализация предметных областей

Для успешного взаимодействия между агентами необходимо установить некоторые общие стандарты взаимодействия, причем как на уровне различных протоколов, так и на семантическом и даже семиотическом уровне. Среди основных уровней взаимодействия, подлежащих согласованию, можно выделить следующие:

- Стандарты **сетевого взаимодействия** не представляют особого интереса, так как имеется достаточно количество распространенных стандартов для построения распределенных систем. Сюда входят семь уровней модели взаимодействия открытых систем OSI [57] и функционирующий на прикладном уровне стандарт взаимодействия открытых компонентных систем, такой как CORBA [20], Java RMI [121], протоколы удаленного вызова RPC [45], DCE и другие.

- Унификация **языкового взаимодействия** обеспечивается стандартизацией языков для обмена знаниями и запросами агентных системах. Стандартизация языков должна обеспечивать единую семантическую интерпретацию языковых конструкций всеми агентами. В эту группу входят достаточно известные стандарты языков запросов и манипулирования знаниями KQML (Knowledge Query and Manipulation Language, [115]), обмена знаниями KIF (Knowledge Interchange Format, [114]), а также основанные на них стандарты ассоциации FIPA ACL (Agent Communication Language, [116]).

- **Семиотическое согласование** в многоагентной системе достигается использованием всеми агентами единой системы понятий и одинаковой их интерпретацией и/или аксиоматизацией. Это достигается за счет разработки единого формального описания **онтологии** предметной области.

Таким образом, одним из ключевых моментов при построении многоагентных систем является онтологическая совместимость семейства агентов, т.е. семиотически согласованное использование ими единой онтологии предметной области.

Онтологию в инженерии знаний обычно определяют как *эксплицитную спецификацию концептуализации* предметной области [14, 76], т.е. некоторое формальное описание основных понятий и аксиоматизированных взаимосвязей между ними, а также правил интерпретации этих понятий и взаимосвязей. **Формальная модель онтологии** определяется [14] как набор $\mathcal{O} = \langle X, \mathcal{R}, \Phi \rangle$, состоящий из множества концептов X, множества отношений между концептами $\mathcal{R} = \{f : f \subseteq X \times X\}$ и множества функций интерпретации Φ, которые в частном случае могут быть заданы на подмножестве концептов $X_1 \subseteq X$, определяя другую часть концептов $X_2 \subseteq X : X_1 \cap X_2 = \varnothing$ через это множество ($\Phi = \{\varphi : X_1^n \to X_2\}$).

Как правило, выделяют [14] следующие разновидности онтологий:

- Простейший **словарь понятий**, включающий в себя явно заданные имена, соответствующие понятиям (концептам) предметной области (в этом случае $\mathcal{R} = \Phi = \varnothing$).
- **Таксономия**, определяющая на словаре понятий отношение наследования $isa \in \mathcal{R}$.
- **Пассивный** или **активный** словарь, включающий при пустом множестве отношений некоторую функцию интерпретации на множестве концептов, определяя одни концепты через другие.

В более общем случае онтология определяется как множество **концептов** предметной области, организованных таксономически отношением наследования `is_a`, и некоторое множество явно представленных знаний относительно этих концептов [76], которые служат для явного или неявного задания функций интерпретации. Словарь обычно специфицирует не только имена понятий, но и их структуру (т.е. набор атрибутов) [5]. Также онтология может содержать активные знания в форме аксиом над определяемыми понятиями.

В рамках проекта Ontolingua разработан интерактивный инструментарий, функционирующий в среде Интернет, для групповой коллаборативной разработки и поддержания онтологий, а также разработан целый ряд прикладных онтологий для различных предметных областей. Для спецификации онтологических знаний в этом проекте используется язык KIF [114].

Введенное в [14] понятие *расширяемой онтологии* и *онтологической системы* позволяет рассматривать семейства распределенных взаимосвязанных онтологий различных уровней абстракции. Как правило, выделяют **метаонтологию** O^{meta}, определяющую набор базовых концептов для построения **онтологий предметных областей** O^{domain} и **онтологий задач** O^{task}. С множеством онтологий также ассоциируется некоторая машина вывода Ξ, задающая правила логического вывода на содержащихся в онтологиях знаниях. Таким образом, модель онтологической системы определяется как $\Sigma = \langle O^{meta}, \{O_i^{domain}\}, \{O_j^{task}\}, \Xi \rangle$.

Однако онтологическое описание не всегда может быть непосредственно положено в основу функционирования агента, так как его внутреннее представление знаний может отличаться от используемого при описании онтологии или совсем отсутствовать (как например у реактивных агентов). В [76] предлагается подход трансляции переносимых онтологий в проблемно-зависимое представление знаний внутри делиберативного агента (см. рис. 2.2, слева). В других случаях онтологические знания могут транслироваться вручную или неявно учитываться при программировании реактивных агентов или разработке интеллектуального наполнения делиберативных агентов.

Таким образом, для многоагентных систем онтология играет роль внешнего описания предметной области, которое связано с внутренней структурой этих знаний внутри агентов весьма опосредовано. В то время как онтологические описания как правило построены по таксономическому принципу, взаимодействие агентов может иметь весьма сложную природу, не обязательно соответствуя структуре представленных онтологической системой знаний. Хотя такая гибкость во многих задачах может быть весьма необходима, для задач распределенного хранения и использования знаний удобно рассматривать

[5] Можно рассматривать понятия и атрибуты как элементы словаря с соответствующим образом заданными отношениями принадлежности между понятием и атрибутом

модели взаимодействия агентов, в которых это взаимодействие подчинено таксономической природе распределенной онтологической системы. Предлагаемая в данной работе архитектура распределенной фреймовой иерархии как раз обладает таким свойством, т.е. является **автоонтологичной**.

2.1.3. Компонентная архитектура

В то время как одной из основных характеристик агентной архитектуры является автономность и атомарность агентов, интерес представляет также альтернативный подход, развитый автором в работах [49,99], при котором распределенная интеллектуальная система строится из неинтеллектуальных компонентов, соответствующих классическим компонентам интеллектуальных систем: процессоров вывода, баз (источников) знаний, доски объявлений (соответствующей рабочей памяти в классической экспертной системе), интерфейса пользователя и др. В этом случае компоненты обмениваются по сети статическими (например, парами атрибут-значение) или динамическими (продукционными правилами) знаниями в некотором внутреннем представлении, которые могут комбинироваться, образуя различные конфигурации обмена знаниями по сети. В то время как в агентной архитектуре внутренняя структура логического вывода каждого агента (и, соответственно, статические знания и накопленный опыт) скрывается от внешнего окружения, в компонентной архитектуре присутствует полный обмен знаниями, благодаря которому реализуется распределенный вывод и распределенное использование знаний.

Основной недостаток компонентной архитектуры в том, что все знания представлены в некотором внутреннем представлении, и тем самым отдельные компоненты системы не обладают автономией и законченностью. Законченная функциональность появляется только при комбинировании всех компонентов — при этом приходится следить за корректностью и непротиворечивостью получившейся в результате комбинирования базы знаний. Поэтому представляет интерес разработка методологии распределенного представления знаний, при которой сохраняется возможность обмена знаниями в статической и динамической форме, но при этом кластеризация знаний основана на некотором естественном принципе, сохраняющем некоторую самостоятельную семантическую ценность отдельных элементов. Результирующая модель таким образом будет представлять собой синтез агентного подхода с традиционными методами построения распределенных гетерогенных систем и будет основана на иерархическом комбинировании онтологических структур в единую распределенную по компьютерной сети иерархию фреймов.

2.2. Архитектура распределенной фреймовой иерархии

В данной работе предлагается гибридная архитектура построения распределенных интеллектуальных систем, обладающая свойствами как компонентной, так и агентной архитектур. Основным недостатком компонентной архитектуры является неавтономность и различная структура компонентов, из-за чего создание интеллектуальной системы требует определенной комбинации и совместной работы всех компонентов. Таким образом, отсутствует возможность сокрытия внутренней структуры и автономного использования некоторого узла системы, что приводит к значительному усложнению процесса разработки и поддержания распределенной базы знаний.

В то же время агентная архитектура является весьма общей и для решения конкретных задач каждый раз должна уточняться специфическими архитектурными реше-

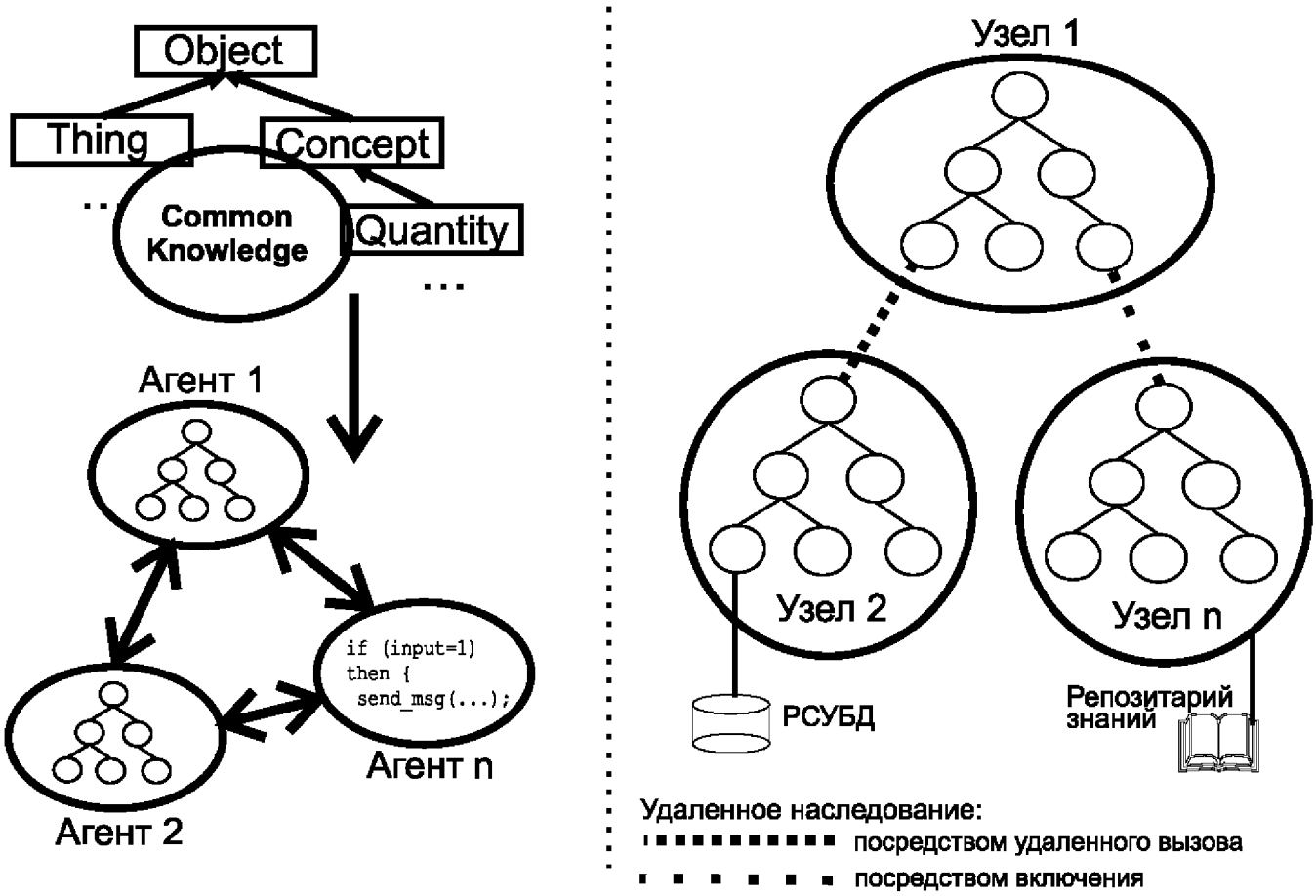


Рис. 2.2. Классическая агентная архитектура (слева) и архитектура распределенной фреймовой иерархии (справа)

циями. При агентном подходе (см. раздел 2.1.2) составные агенты в системе примерно равноправны и взаимодействуют друг с другом либо заранее определенным образом, либо динамически, выстраивая граф взаимодействия в процессе работы. Однако можно привести множество примеров, в основном связанных с распределенным накоплением и использованием знаний, когда взаимодействие сводится к использованию агента для получения ответа на запрос (**удаленная консультация**), либо к использованию знаний агента для их дальнейшего расширения и дополнения с целью получения более полной или специализированной интеллектуальной системы. Такое расширение весьма напоминает наследование в терминах объектно-ориентированного или фреймового подхода, что наводит на мысль об иерархической организации системы в виде **распределенной фреймовой иерархии**. Предлагаемая организация основана на использовании фреймового представления знаний для отображения на фреймовую иерархию множества концептов таксономической онтологии, при этом распределенная иерархия соответствует распределенному характеру расширяемой онтологической системы.

Архитектура **распределенной фреймовой иерархии** строится на основе иерархического комбинирования узлов интеллектуальной системы с фреймовой моделью представления знаний, так что вся распределенная система представляет собой единую иерархию фреймов, связанную отношением наследования, которое может распространяться между узлами системы. Каждый из узлов содержит в себе некоторый участок фреймовой иерархии, т.е. свои базу знаний, процессор вывода и рабочую память, образующие фрейм-

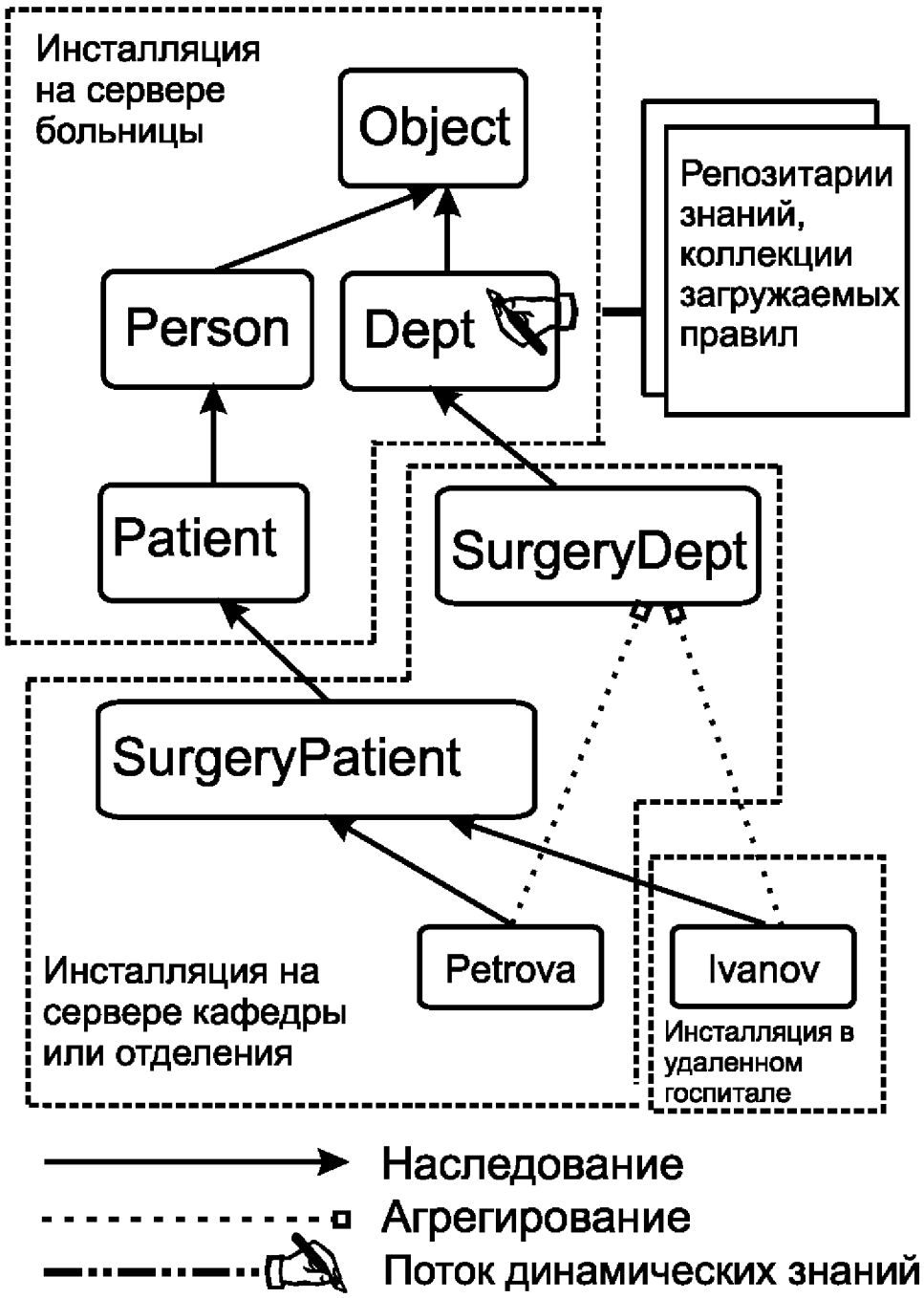


Рис. 2.3. Пример распределенной фреймовой иерархии

мир, но может ссылаться (отношением наследования, агрегации или использования) на фреймы, расположенные на других узлах. Возможны различные модели комбинирования фреймовых субиерархий на распределенных узлах сети: все миры могут объединяться в единую фреймовую иерархию с общим родителем, некоторый фрейм может по очереди присоединяться (наследоваться) к различным иерархиям, образуя аналог модели доски объявлений, или же фрейм-миры могут взаимодействовать более сложным образом, представляя собой сеть с распределенными статическими знаниями. Все фрейм-миры, участвующие в некотором общем процессе вывода, образуют **кластер**. При этом взаимодействие фрейм-миров в рамках кластера сводится к следующим видам взаимодействия:

- Вычисление удаленного слота (статическое или мобильное)
- Удаленное наследование (статическое или мобильное)

Предложенная архитектура дополнена также средствами для обмена динамическими знаниями (в виде продукционных правил прямого и обратного вывода) по сети и использования **репозитария знаний** с загрузкой правил по мере необходимости в процессе вывода (on-demand rule loading).

Логический вывод в системе в целом происходит аналогично выводу в нераспределенной иерархии, с учетом удаленного вызова методов на удаленных фреймах. Такой удаленный вызов позволяет осуществлять обмен значениями слотов, и таким образом построить обмен знаниями на основе обмена статической информацией о состоянии задачи.

Пример распределенной фреймовой иерархии изображен на рис. 2.3.

2.2.1. Вычисление удаленного слота

В простейшем случае фрейм-миры могут объединяться в кластер без образования единой иерархии с общим родителем. Мы будем называть такую структуру **распределенной фреймовой коллецией**, оставляя название **иерархия** для таких способов комбинирования, при которых в иерархии имеется единственный общий родитель, так как именно такие структуры представляют особый интерес для распределенных онтологических описаний.

Для обеспечения распределенной функциональности синтаксис обращения к слоту фрейма расширяется возможностью указания имени удаленной иерархии, к которой относится требуемый фрейм, т.е. ссылка на фрейм может иметь вид `WorldID->FrameID`. При указании в некотором фрейм-мире имени удаленного слота для его вычисления может применяться два подхода:

- **Статическое вычисление путем удаленного вызова** состоит в том, что текущий узел сети производит запрос к удаленному узлу на вычисление значения слота, при этом процесс логического вывода также переносится на удаленный узел, пока значение запрашиваемого слота не будет возвращено назад[6].
- **Мобильное вычисление путем включения**. При этом необходимые для вывода статически и динамические знания из удаленного мира (а возможно и весь удаленный фрейм-мир) переносятся на текущий узел сети, где принимают участие в процессе локального вывода для получения значения требуемого слота.

[6]В процессе удаленного вывода может также происходить удаленное обращение к слотам фреймов исходного фрейм-мира

2.2.2. Удаленное наследование

Удаленное наследование является в некотором смысле частным случаем удаленного вычисления слота и основой для организации распределенной иерархии. Различают мобильное и статическое[7] удаленное наследование.

При **мобильном удаленном наследовании** в случае, когда требуется использовать ссылку на родительский фрейм, расположенный на удаленном узле, производится передача требуемого фрейма (либо всей включающей его иерархии) на локальный узел, где затем ссылка на него производится обычным образом. При этом логический вывод все время производится на одном и том же сетевом узле, а удаленные знания в виде дополнительных фрейм-миров подгружаются по сети по мере необходимости.

Аналогом рассмотренного выше мобильного наследования является **частичное мобильное наследование**, при котором по сети передаются не все знания с удаленного узла, а лишь те правила, которые необходимы для вычисления значения слота локального фрейма. В этом случае функциональность напоминает передачу знаний из репозитария, но организация удаленных знаний в виде правил для слота некоторого фрейма обеспечивает их более прозрачную поддержку.

Статическое удаленное наследование реализуется удаленным вызовом процедур, при этом логический вывод переходит с текущего узла сети на удаленный. Так как знания на удаленном узле должны применяться к значениям слотов текущего базового фрейма, то при удаленном вызове передается также ссылка на базовый фрейм, находящийся на текущем узле, чтобы значения его слотов могли быть определены посредством callback-вызовов.

Более подробно реализация удаленного наследования в инструментарии JULIA рассмотрена в соответствующих разделах ниже. Отметим, что при удаленном наследовании сохраняется поддержка как обратного, так и прямого методов логического вывода. Обсуждение преимуществ использования обратного вывода при удаленном наследовании можно найти в разделе 2.4.1.

2.2.3. Репозитарии знаний и загрузка правил по требованию

В некоторых случаях может оказаться полезным иметь возможность использовать в процессе логического вывода знания из некоторых удаленных источников, организованные некоторым отличным от фреймовой иерархии образом (например, в виде коллекции правил, подчиняющихся некоторой онтологии). Для продукционно-фреймовой системы с преимущественно обратным выводом, рассмотренной в главе 1, может быть легко реализована возможность загрузки правил обратного вывода из внешнего хранилища по мере необходимости, наподобие того, как в реализуется поочередная загрузка правил в компонентной архитектуре [49, 99]. Действительно, алгоритм обратного вывода 1.1 требует лишь знания порядка применения правил до начала вывода (который во многих случаях не требует знания самих правил[8]), а в процессе вывода правила используются поочередно

[7] Термин "статическое" в данном случае является весьма неудачным, так как в предыдущем разделе статическое наследование противопоставлялось динамическому и означало неизменность родителя в процессе вывода. В данном случае термин применяется как антоним мобильному наследованию, по аналогии с общепринятой терминологией для классификации агентов. Предполагается, что из контекста всегда можно определить, о какой "статичности" наследования идет речь.

[8] В тех случаях, когда требуется реализовать сложную стратегую разрешения конфликта, требующую исследования самих правил, такая стратегия может быть реализована на стороне репозитария, предоставив основному инструментарию использовать простой порядок выбора правил по мере их поступления

до тех пор, пока не будет получено значение слота.

2.3. Семантика распределенного вывода

При построении модели распределенной фреймовой иерархии и описании семантики распределенного вывода мы будем действовать по следующей схеме: введем понятие распределенной фреймовой системы, расширим описанный в разделе 1.3.4.1 синтаксис выражений за счет введения удаленных ссылок ($\blacklozenge$ и $\lozenge$) и на базе этого введем операции комбинирования функций состояния ($\bigstar$, $\blacktriangleleft/\triangleleft$, $\bowtie$). При помощи этого мы определим два вида семантики распределенного вывода: эквивалентную семантику для порожденной нераспределенной системы и семантику, расширяющую описанную в разделе 1.3.4, определенную на множестве функций состояния и учитывающую процессы удаленного взаимодействия. Это позволит нам сформулировать утверждения об эквивалентности мобильного и статического взаимодействия в смысле их соответствия семантике порожденной системы. В заключение мы покажем, как при помощи функций-оракулов можно учесть особенности реального сетевого взаимодействия, наподобие того, как это сделано в [97].

2.3.1. Основные понятия и определения

Для описания семантики распределенного вывода нам потребуется рассматривать не только некоторый выделенный фрейм-мир, а всю совокупность входящих во фреймовую коллекцию функций состояния. Состояние всей фреймовой коллекции в этом случае будет описываться набором функций состояния, который мы и назовем распределенной фреймовой системой.

Определение 2.1. Под **распределенной фреймовой системой** $\tilde{\mathcal{W}}$ будем понимать множество функций состояния $\{\mathcal{W}_1, \ldots, \mathcal{W}_n\}$ входящих в систему субиерархий.

Будем рассматривать только такие модели, в которых заранее известны все участвующие в выводе субиерархии. Это ограничение на первый взгляд может показаться слишком существенным, однако ввиду реальной конечности числа возможных фрейм-иерархий по сути дела оно не накладывает никаких ограничений.

При рассмотрении модели может оказаться удобно вместо нумерования фрейм-миров обозначать их именующими индексами. Такое расширение обозначений (по аналогии с [60]) по сути дела служит для повышения наглядности и не меняет проводимых рассуждений.

Будем также без ограничения общности предполагать, что фреймы, входящие в каждый из составляющих фрейм-миров, именованы уникальным образом, т.е.

$$\forall i, j \in 1 \ldots n \quad (i \neq j) \Rightarrow I_i \cap I_j = \varnothing \tag{2.1}$$

где I_j — множество идентификаторов фреймов, соответствующих $\mathcal{W}_j$. Этого можно добиться, например, присоединяя к имени фрейма номер или индекс соответствующего фрейм-мира — будем называть такое именование **расширенным именем фрейма**. Различие в именованиях фреймов будет служить для удобства введения дальнейших понятий, так как непересечение имен позволяет по имени фрейма сразу определить функцию состояния, его описывающую.

Определение 2.2. Объединением двух фрейм-миров $\mathcal{W}_1$ и $\mathcal{W}_2$ будем называть фрейм-мир $\mathcal{W}_1 \bigstar \mathcal{W}_2$, описываемый следующей функцией состояния:

$$\mathcal{W}_1 \bigstar \mathcal{W}_2 = \lambda f, s. \begin{cases} \mathcal{W}_1(f,s), & \text{если } \langle f, s\rangle \in \mathbb{I}_1 \\ \mathcal{W}_2(f,s), & \text{если } \langle f, s\rangle \in \mathbb{I}_2 \\ \bot, & \text{в противном случае} \end{cases} \tag{2.2}$$

Из свойства уникальности именования следует

Лемма 2.1.

$$\forall\, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3 \quad \mathcal{W}_1 \bigstar \mathcal{W}_2 = \mathcal{W}_2 \bigstar \mathcal{W}_1,\ u\ (\mathcal{W}_1 \bigstar \mathcal{W}_2) \bigstar \mathcal{W}_3 = \mathcal{W}_1 \bigstar (\mathcal{W}_2 \bigstar \mathcal{W}_3) \tag{2.3}$$

Таким образом, можно определить операцию $\bigstar$ для любой последовательности функций состояния, причем результат не будет зависеть от порядка вычисления.

Определение 2.3. Назовем

$$\mathcal{W} = \bigstar_{i=1}^{n} \mathcal{W}_i \tag{2.4}$$

функцией состояния распределенной фреймовой системы $\tilde{\mathcal{W}} = \{\mathcal{W}_1, \ldots, \mathcal{W}_n\}$. Будем также использовать обозначение $\mathcal{W} = \bigstar \tilde{\mathcal{W}}$.

Помимо операции $\bigstar$ полезно определить операцию **подчиненного наследования** двух фреймовых иерархий следующим образом:

Определение 2.4.

$$\mathcal{W}_1 \underset{f}{\blacktriangleleft} \mathcal{W}_2 = \mathcal{W}_1 \bigstar \mathcal{W}_2[\mathcal{W}_2.\mathit{object.parent} \leftarrow \mathcal{W}_1.f] \tag{2.5}$$

В этом определении, если $\mathcal{W}_1$ и $\mathcal{W}_2$ являются фреймовыми иерархиями (т.е. иерархиями с единственным общим родителем `object`), то результат также будет являться фреймовой иерархией, в которой общий родитель $\mathcal{W}_2$ будет унаследован от указанного фрейма f иерархии $\mathcal{W}_1$. В случае, если запись $\mathcal{W}_1 \underset{f}{\blacktriangleleft} \mathcal{W}_2$ рассматривается в контексте распределенной фреймовой системы, то подразумевается

$$\mathcal{W}_1 \underset{f}{\blacktriangleleft} \mathcal{W}_2 = \{\mathcal{W}_1, \mathcal{W}_2[\mathit{object.parent} \leftarrow \mathcal{W}_1.f]\} \tag{2.6}$$

Кроме подчиненного наследования, определим также полезную операцию **симметричного соединения** $\bowtie$:

Определение 2.5.

$$\mathcal{W}_1 \bowtie \mathcal{W}_2 = (\mathcal{W}_o \underset{\mathit{object}}{\blacktriangleleft} \mathcal{W}_1) \underset{\mathcal{W}_o.\mathit{object}}{\blacktriangleleft} \mathcal{W}_2 \tag{2.7}$$

где $\mathcal{W}_o$ — пустая фреймовая иерархия, содержащая только единственный фрейм *object* без слотов. Аналогично предыдущему определению, в контексте распределенной фреймовой системы можно показать, что имеет место следующая

Лемма 2.2.

$$\mathcal{W}_1 \bowtie \mathcal{W}_2 = \{\mathcal{W}_o, \mathcal{W}_1[\mathit{object.parent} \leftarrow \mathcal{W}_o.\mathit{object}], \mathcal{W}_2[\mathit{object.parent} \leftarrow \mathcal{W}_o.\mathit{object}]\} \tag{2.8}$$

Следствие 2.1.

$$\forall\, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3 \quad \mathcal{W}_1 \bowtie \mathcal{W}_2 = \mathcal{W}_2 \bowtie \mathcal{W}_1, \quad (\mathcal{W}_1 \bowtie \mathcal{W}_2) \bowtie \mathcal{W}_3 = \mathcal{W}_1 \bowtie (\mathcal{W}_2 \bowtie \mathcal{W}_3) \tag{2.9}$$

Введенные выше операции позволяют строить распределенные фреймовые системы, комбинируя их из существующих функций состояния. Таким образом, некоторое выражение, построенное в терминах операций $\bigstar$, $\blacktriangleleft$ и $\bowtie$, полностью определяет как саму фреймовую систему, так и соответствующую ей функцию состояния[9].

2.3.2. Семантика распределенного вывода в терминах порожденной системы

Так как распределенная система в свою очередь представляет собой единую фреймовую иерархию, то модель распределенного вывода представляет собой синхронный поиск в общем пространстве состояний задачи [87], т.е. общее пространство состояний распределено по фреймовой иерархии, и соответственно поиск управляется поочередно разными процессорами.

Проще всего определить семантику вывода в распределенной системе на основе ее функции состояния. Функция состояния распределенной системы является функцией состояния некоторой нераспределенной фреймовой иерархии, которую мы будем называть **порожденной фреймовой системой** для исходной распределенной фреймовой коллекции. Таким образом, при определении семантики распределенного вывода можно рассматривать порожденную систему, семантика для которой уже является определенной.

Однако, такая семантика не позволяет нам исследовать процессы распределенного взаимодействия, возникающие во фреймовой коллекции. По сути дела порожденная система представляет собой множество всех фреймов распределенной иерархии, собранных локально в единый фрейм-мир. Таким образом, в данной семантике не представляется возможным контролировать множество знаний, сосредоточенных в каждом из узлов сети в каждый момент времени.

2.3.3. Семантика распределенного вывода в терминах состояний исходной распределенной системы

Для рассмотрения распределенных взаимодействий требуется определять семантику на всем множестве состояний распределенной системы $\tilde{\mathcal{W}} = \{\mathcal{W}_1, \ldots, \mathcal{W}_n\}$. По аналогии с семантикой, рассмотренной в разделе 1.3.4, будем рассматривать отображение $\tilde{\mathcal{E}} : \mathbb{E} \times \tilde{\mathbb{W}} \times \mathbb{C} \to \mathbb{T} \times \tilde{\mathbb{W}}$, которую для ясности будем обозначать как $v = [\![E]\!]^{C}_{\tilde{\mathrm{W}} \to \tilde{\mathrm{W}}'} \iff \tilde{\mathcal{E}}(E, \tilde{\mathcal{W}}, C) = (v, \tilde{\mathcal{W}}')$.

Синтаксис выражений расширим за счет введения **удаленных ссылок**, т.е. ссылок на имена слотов, расположенных в отличных от текущего фрейм-мирах. Введем также операции **статического** ($\blacklozenge$) и **мобильного** ($\lozenge$) вычисления удаленной ссылки. С учетом этих операций можно также определить две операции подчиненного наследования статическую ($\blacktriangleleft$) и мобильную ($\triangleleft$) следующим образом:

$$\begin{aligned} \mathcal{W}_1 \underset{f}{\blacktriangleleft} \mathcal{W}_2 &= \{\mathcal{W}_1, \mathcal{W}_2[\mathit{object.parent} \leftarrow \blacklozenge \mathcal{W}_1.f]\} \\ \mathcal{W}_1 \underset{f}{\triangleleft} \mathcal{W}_2 &= \{\mathcal{W}_1, \mathcal{W}_2[\mathit{object.parent} \leftarrow \lozenge \mathcal{W}_1.f]\} \end{aligned} \tag{2.10}$$

[9] Строго говоря, для доказательства этого утверждения следует показать, что для всех операций $\circledast \in \{\bigstar, \blacktriangleleft, \bowtie\}$ верно $\mathcal{W}_1 \circledast \mathcal{W}_2 = \{\mathcal{W}_1', \mathcal{W}_2'\} \Rightarrow \mathcal{W}_1 \circledast \mathcal{W}_2 = \mathcal{W}_1' \bigstar \mathcal{W}_2'$.

Расширим определение фреймовой системы $\tilde{\mathcal{W}}$, дополнив ее понятием "текущего" фрейм-мира, в котором в данный момент производится логический вывод. Для удобства будем записывать это, помечая текущий фрейм-мир чертой, например $\tilde{\mathcal{W}} = \{\mathcal{W}_1, \overline{\mathcal{W}}_2, \ldots, \mathcal{W}_n\}$[10].

Тогда семантика вычисления в распределенной системе произвольного выражения $E \in \mathbb{E}$, не содержащего удаленных ссылок, будет определяться через семантику локального вывода естественным образом:

$$[\![E]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}'_i,\ldots,\mathcal{W}_n\}} = \|E\|_{\mathcal{W}_i\to\mathcal{W}'_i} \tag{2.11}$$

Особый интерес будет представлять семантика обработки удаленных ссылок, которая будет рассмотрена в следующих подразделах.

2.3.3.1. Семантика мобильного удаленного вывода

В случае использования мобильной ссылки ее вычисление сводится к тому, что упомянутый в ней фрейм-мир присоединяется (с помощью операции ★) к текущему, и логический вывод продолжается в текущем фрейм-мире — при этом удаленная ссылка заменяется обычной:

$$[\![\Diamond\mathcal{W}_j(f,s)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = [\![f.s]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}_i\bigstar\mathcal{W}_j},\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} \tag{2.12}$$

В случае, если во всей распределенной фреймовой системе используется только мобильный удаленный вывод, то текущим всегда остается только один фрейм-мир; при этом он монотонно расширяется за счет присоединения к нему других фрейм-миров.

Мобильное наследование с поддержкой прямого и обратного вывода реализуется через вычисление удаленной ссылки естественным образом. При необходимости обращения к удаленному родителю производится вычисление удаленной ссылки по формуле 2.12, что приводит к объединению двух фрейм-миров и проведению логического вывода в полном соответствии с локальной семантикой.

2.3.3.2. Семантика статического удаленного вывода

При статическом удаленном выводе модификация фрейм-миров не производится, но меняется "текущий" фрейм-мир, и дальнейший вывод производится уже с использованием другой функции-состояния (и, соответственно, определенных в ней правил):

$$[\![\blacklozenge\mathcal{W}_j(f,s)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = [\![f.s]\!]_{\{\mathcal{W}_1,\ldots,\mathcal{W}_i,\ldots,\overline{\mathcal{W}}_j,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} \tag{2.13}$$

При удаленном наследовании ситуация обстоит несколько сложнее, чем в случае мобильного вывода. При обратном выводе, когда в результате вычисления значения слота управление переходит к удаленному родителю, и логический вывод продолжается для другой функции состояния, базовый фрейм, подставляемый в правила вместо *this*, должен соответствовать исходному фрейму, который с точки зрения новой функции состояния является удаленным. Таким образом, переход логического вывода на другую функцию состояния приводит к появлению удаленной ссылки в контексте вывода:

$$[\![\blacklozenge\mathcal{W}_j(f,s)]\!]^F_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = [\![f.s]\!]^{\blacklozenge\mathcal{W}_i(F)}_{\{\mathcal{W}_1,\ldots,\mathcal{W}_i,\ldots,\overline{\mathcal{W}}_j,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} \tag{2.14}$$

[10]Строго говоря, фреймовая система в данном случае будет представлять собой упорядоченный набор из множества фрейм-миров и идентификатора текущего фрейм-мира.

При этом в процессе вывода для функции состояния $\mathcal{W}_j$ при каждом вычислении *this* будет производиться повторное вычисление удаленной ссылки и перенос текущего фрейма назад к $\mathcal{W}_i$ — при этом операция вычисления удаленной ссылки будет снята, в том числе и с контекста вызова, в соответствии с правилом

$$[\![\blacklozenge\mathcal{W}_i(F)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}} = \|F\|_{\mathcal{W}_i\to\mathcal{W}_i} = \mathcal{W}_i(F) \tag{2.15}$$

2.3.4. Свойства эквивалентности различных семантик распределенного вывода

Формальное описание семантики для распределенного мобильного и статического вывода позволяет сформулировать некоторые утверждения об эквивалентности распределенного вывода и вывода в эквивалентной фреймовой иерархии. Эти утверждения оказываются истинными в некотором классе систем, которые мы назовем **нормальными**. К этому классу относятся системы, построенные на одном виде взаимодействия (удаленный вызов или включение), **строго полиморфные** системы (в которых все правила являются внутрифреймовыми, и все ссылки на слоты в правилах полиморфны), а также системы, в которых доступ к каждой отдельно взятой фреймовой иерархии осуществляется только на основе одного вида взаимодействия (**сепаратные**). Следует отметить, что большинство возникающих в реальных задачах распределенного накопления и использования знаний систем является нормальными.

Утверждение 2.2 (о вычислении мобильной ссылки).

$$[\![\Diamond\mathcal{W}_j(f,s)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = \|\mathcal{W}_j(f,s)\|_{\mathcal{W}\to\mathcal{W}'} \tag{2.16}$$

где $\mathcal{W} = \mathcal{W}_1 \bigstar \ldots \bigstar \mathcal{W}_n$ — *общее состояние фреймовой системы, при этом* $\mathcal{W}' = \bigstar\tilde{\mathcal{W}}'$.

Идея доказательства (для систем на основе исключительно мобильного наследования). При вычислении мобильная ссылка преобразуется к обычной путем объединения двух соответствующих функций состояния, таким образом текущая функция состояния распределенной фреймовой системы постепенно включает в себя все участвующие в выводе фрейм-миры, играя по сути дела роль общей функции состояния фреймовой системы $\bigstar\mathcal{W}$. □

Утверждение 2.3 (о вычислении статической ссылки).

$$[\![\blacklozenge\mathcal{W}_j(f,s)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}}_i,\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = \|\mathcal{W}_j(f,s)\|_{\mathcal{W}\to\mathcal{W}'} \tag{2.17}$$

где $\mathcal{W} = \mathcal{W}_1 \bigstar \ldots \bigstar \mathcal{W}_n$ — *общее состояние фреймовой системы, при этом* $\mathcal{W}' = \bigstar\tilde{\mathcal{W}}'$.

Идея доказательства. Следует для начала рассмотреть процесс вычисления ссылки в том случае, когда дальнейший распределенный вывод не инициируется, и затем перейти к индукции по длине цепочки удаленного вывода. Следует отметить, что дальнейшие удаленные ссылки не возникают в том случае, когда все необходимые для вычисления выражения слоты находятся в рамках текущей фрейм-иерархии $\mathcal{W}_i$, а в этом случае семантика вывода в ней будет соответствовать семантике вывода в иерархии $\mathcal{W} \supseteq \mathcal{W}_i$[11]. □

[11] В данном случае включение понимается в терминах множества слотов, т.е. $\mathcal{W}_1 \subseteq \mathcal{W}_2 \iff \mathbb{I}_1 \subseteq \mathbb{I}_2$, и $\forall\,\langle f,s\rangle \in \mathbb{I}_1\, \mathcal{W}_1(f,s\rangle = \mathcal{W}_2(f,s)$, где $\mathbb{I}_1 = \mathcal{D}\mathcal{W}_1$, $\mathbb{I}_2 = \mathcal{D}\mathcal{W}_2$.

Утверждение 2.4 (Об эквивалентности семантики распределенного и локального вывода). *Для любого выражения $E \in \mathbb{E}$ в нормальной системе $\tilde{\mathcal{W}}$ имеет место* $[\![E]\!]_{\tilde{\mathcal{W}}\to\tilde{\mathcal{W}}'} = \|E\|_{\mathcal{W}\to\mathcal{W}'}$, *где* $\mathcal{W} = \bigstar\tilde{\mathcal{W}}$ *и* $\mathcal{W}' = \bigstar\tilde{\mathcal{W}}'$.

Это утверждение означает, что нормальная распределенная фреймовая иерархия с мобильным, статическим или смешанным наследованием в смысле семантики будет эквивалентна некоторой локальной фреймовой иерархии. Кроме того, отсюда следует эквивалентность статического и мобильного удаленного наследования в терминах определяемой ими семантики вычислений — разница проявляется лишь в том, как общие знания в системе будут распределены между ее составными частями. При мобильном наследовании все знания (статические и динамические) по мере вывода аккумулируются в одной текущей субиерархии, в то время как при статическом наследовании они распределены по сети, и обмен статическими знаниями происходит каждый раз по мере необходимости.

2.3.5. Учет особенностей реального сетевого взаимодействия в семантике вывода

При проведении удаленного вызова или при передаче фрейм-мира по сети в реальной распределенной системе могут возникать различного рода проблемы сетевого взаимодействия. Для учета такого рода проблем в [97] предлагается воспользоваться функцией-оракулом *download*(P), моделирующей процесс сетевых взаимодействий. В нашем случае роль функции-оракула будут выполнять операции $\blacklozenge$ и $\lozenge$, которые в зависимости от успешности сетевых операций будут при интерпретации возвращать требуемое значение или $\bot$. С учетом этого формулы 2.12 и 2.13 преобразуются следующим образом:

$$[\![\lozenge\mathcal{W}_j(f,s)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}_i},\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = \begin{cases} [\![f.s]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}_i\bigstar\mathcal{W}_j},\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'}, & \text{если передача данных от узла} \\ \quad j \text{ к узлу } i \text{ прошла успешно} \\ \bot, \text{ если возникла ошибка сетевого взаимодействия} \\ \quad \text{между узлами } i \text{ и } j \end{cases} \tag{2.18}$$

$$[\![\blacklozenge\mathcal{W}_j(f,s)]\!]_{\{\mathcal{W}_1,\ldots,\overline{\mathcal{W}_i},\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'} = \begin{cases} [\![f.s]\!]_{\{\mathcal{W}_1,\ldots,\mathcal{W}_i,\ldots,\overline{\mathcal{W}_j},\ldots,\mathcal{W}_n\}\to\tilde{\mathcal{W}}'}, & \text{если удаленный} \\ \quad \text{вызов узла } j \text{ узлом } i \text{ прошел успешно} \\ \bot, \text{ если возникла ошибка сетевого взаимодействия} \\ \quad \text{между узлами } i \text{ и } j \end{cases} \tag{2.19}$$

В случае такого определения вычисления удаленной ссылки семантика перестает быть детерминированной, так как результат вычисления выражения начинает зависеть от поведения внешних функций-оракулов. Однако, такое определение семантики позволяет учитывать особенности реальных процессов передачи данных в компьютерных сетях и моделировать поведение процессов логического вывода в условиях неудачных сетевых взаимодействий.

2.3.6. Синхронный и параллельный вывод

Из-за синхронного характера вывода в рассмотренной семантике (т.е. управление передается поочередно от процессора к процессору, в то время как все другие процессоры находятся в состоянии ожидания или готовности к ответу на запрос) не возникает

проблем с синхронизацией и распараллеливанием вывода. Это также упрощает реализацию инструментария, но лишает распределенный вывод некоторой привлекательности с точки зрения эффективности, так как возможность распараллеливания процесса вывода была бы чрезвычайно привлекательна с точки зрения решения задач, в которых разные подзадачи могли бы решаться разными процессорами вывода. Заметим, что данный дефект не является следствием модели распределенной фреймовой иерархии — напротив, в принципе вполне допустима параллельная работа процессоров вывода в различных инсталляциях. В этом случае подразумевается, что если в некоторое выражение входят ссылки на слоты удаленных фреймов, то начинать их вычисление следует параллельно.

В данной работе подробное рассмотрение возможности распараллеливания вывода не проводится, так как оно может стать предметом самостоятельного исследования. Отметим, что существуют два принципиально различных подхода к осуществлению параллельного вывода:

- **Автоматическое распараллеливание вывода** подразумевает, что процесс распараллеливания будет планироваться системой и осуществляться автоматически. Это потребует существенной модификации семантики вывода, основанной на укороченном вычислении выражений и строгой монотонности операторов, а также разработки специальных алгоритмов и методов планирования.

- **Введение эксплицитных распараллеливающих операций**, для которых аргументы будут вычисляться параллельно на различных процессорах вывода. В этом случае для таких операций будет использоваться полное (неукороченное) вычисление с явно определенной семантикой.

Следует понимать, что автоматическое распараллеливание вывода нарушает традиционную семантику обратного вывода, при которой значения, входящие в выражение, вычисляются по укороченной схеме слева направо. Зачастую создатели баз знаний имеют в виду именно такую семантику выполнения, и параллелизм может некоторым образом нарушить предполагаемый процесс вывода. Однако, в ряде случаев параллельное вычисление весма желательно — например, если цель нашей системы — сравнить значения некоторого атрибута, полученные различными независимыми экспертными системами. Поэтому включение в язык конструкции для указания возможности параллельного вычисления некоторого выражения или операции представляется наиболее разумным, так как это позволит использовать параллелизм совместно с полным вычислением выражения в том случае, где это не противоречит требуемой семантике.

В качестве промежуточного подхода можно отметить **явное предварительное планирование вычислений** с последующей синхронизацией, при котором в начале вывода явно указывается необходимость в дальнейшем знать значение некоторого слота или выражения. Это выражение независимо вычисляется пока его значение не будет востребовано: в этом случае при необходимости происходит ожидание окончания процесса вычисления и используется полученное значение.

2.4. Реализация распределенной фреймовой иерархии в инструментарии JULIA

В данном разделе мы рассмотрим компоненты инструментария JULIA, отвечающие за реализацию распределенного вывода. Этот набор компонент является надстройкой над

описанной в главе 1.4 функциональностью и в то же время является ее составной частью.

Будем называть ядро JULIA (поддерживающее сетевые расширения) с загруженным фрейм-миром, работающее на некотором узле сети **инсталляцией**. Инсталляция включает в себя некоторую фреймовую иерархию (субиерархию), т.е. набор фреймов с уникальным именованием, связанных отношением наследования. С фреймами связаны процедуры-демоны и процедуры-запросы, соответствующие некоторой базе знаний. Пример распределенной фреймовой иерархии приведен на рис.2.3.

В каждой из локальных субиерархий возможно использование двух видов ссылок на **удаленные фреймы** (т.е. на фреймы, принадлежащие фрейм-мирам других инсталляций): **статической** и **мобильной**. Эти виды ссылок соответствуют описанным в разделе 2.3.3 операциям ♦ / ◊ и определяют, будет ли для вычисления ссылки применяться удаленный вызов или включение.

2.4.1. Распределенный вывод на основе удаленного вызова (invokation)

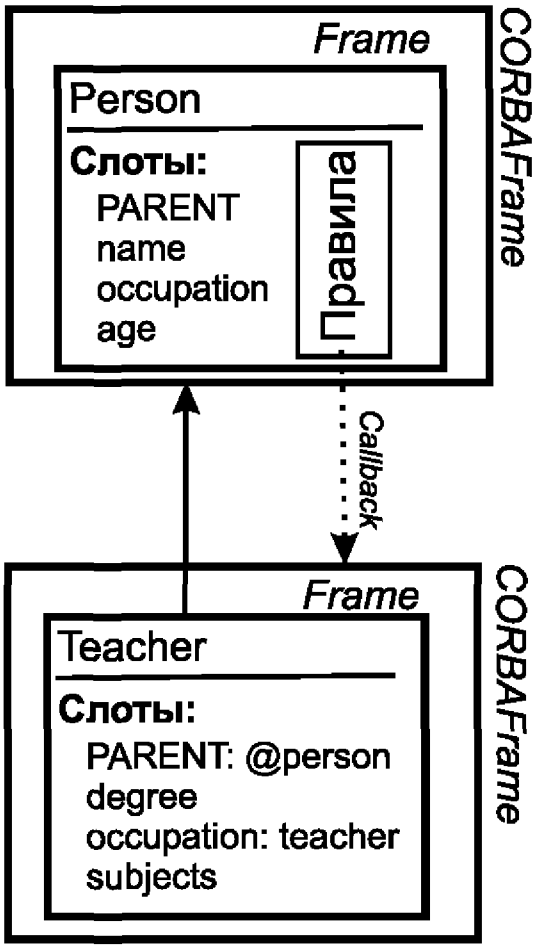


Рис. 2.4. Удаленный вывод с использованием фрейма-посредника

Когда в некотором фрейм-мире необходимо обратиться к удаленному фрейму по статической ссылке, создается т.н. **фрейм-посредник** (proxy frame, см. рис. 2.5), унаследованный от `StoreFrame`, который дублирует атрибуты соответствующего удаленного фрейма, а в случае необходимости получения значения некоторого атрибута производит удаленный вызов соответствующих методов (см. алгоритм 2.1), реализованных интерфейсом удаленного фрейма, что может приводить в процессу обратного логического удаленного вывода на удаленном узле (см. рис. 2.4). Аналогично при присваивании значения слоту производится удаленный вызов, который ведет к прямому выводу на удаленной машине. Дублирование и локальное хранение значений атрибутов позволяет обеспечить кэширование значений во избежание их повторной передачи по сети.

Библиотека поддерживает два вида статических ссылок на удаленные фреймы: **ранние статические ссылки** и **динамические ссылки**. В первом случае (т.н. **раннее связывание**) еще на этапе процессирования исходного текста на JFMDL создается фрейм-посредник, который жестко связывается со ссылкой на удаленный фрейм посредством ссылок используемого механизма удаленного взаимодействия (CORBA, RMI или др.). В случае динамической ссылки (т.н. **позднее связывание**) фрейм-посредник создается только в процессе выполнения и динамически находит удаленный фрейм используя доступные средства именования (например, COS Naming Services).

Алгоритм 2.1: Алгоритм статического вычисления удаленной ссылки

Исходные данные: Имя удаленного фрейм-мира W, имя фрейма в рамках этого мира F.
Результат: Вычисленное значение ссылки типа `TRef` или $\perp$, если фрейм или фрейм-мир не найден.
Вычислить_Статически(W,F)

```
(1)   if W  укороченное имя then W ← соответствующее W глобальное имя
      { Произвести поиск W.F в таблице фреймов-посредников }
(3)   if W.F ∈ таблице созданных фреймов-посредников
(4)     R ← ссылка на фрейм-посредник для W.F
(5)   else
(6)     ref ←Получить_глобальную_ссылку(W,F)
        { Для получения глобальной ссылки производится (возможно удаленный) запрос к ClusterManager }
(8)     if ref = ⊥ then return ⊥
(9)     R ← new CORBAProxyFrame(ref)
(10)    Добавить R в таблицу созданных фреймов-посредников
(11)  return R
```

В распределенной фреймовой иерархии различают два случая использования удаленных ссылок: **наследование** от удаленного фрейма и **агрегация** удаленного фрейма. В свою очередь в случае агрегации удаленный фрейм может использоваться в посылках правила прямого вывода либо в выражении. Последний случай не представляет особого интереса, так как мало чем отличается от удаленного вызова с целью получения значения слота — следует однако иметь в виду, что такой удаленный вызов зачастую приводит к процессу обратного вывода. Если в случае обратного вывода встречаются правила, запрашивающие значения у пользователя — осуществляется обратный вызов (callback) вызывавшей инсталляции, которая и осуществляет интерфейс с пользователем. Таким образом, в стандартном случае исходные данные запрашиваются на одной ЭВМ (что обычно и требуется); однако, всегда возможно использовать процедуры, которые будут осуществлять распределенный сбор данных, что характерно, например, для задач распределенного интеллектуального контроля и управления [24, 107]

В случае наследования от удаленного фрейма обычно происходит переход процесса обратного вывода через границы инсталляции. В этом случае в алгоритме обратного вывода 1.1 производится вызов процедуры `Запрос` для фрейма-посредника, который, в свою очередь, производит удаленный вызов соответствующей процедуры на другой инсталляции, передавая в качестве базового фрейма ссылку на текущий фрейм (алгоритм 2.2 описывает механизм запроса для фрейма-посредника более подробно). В этом случае в удаленной инсталляции также создается фрейм-посредник, который связывается с базовым фреймом для данного вывода. Соответственно, процессом вывода управляют динамические правила, расположенные на удаленной ЭВМ, но значения атрибутов присваиваются исходному базовому фрейму в процессе обратного удаленного вызова.

Таким образом, обмен по сети производится только значениями слотов — статическими знаниями о состоянии задачи — что тем не менее приводит к передаче знаний,

Алгоритм 2.2: Алгоритм запроса значения слота для фрейма-посредника

Исходные данные: Слот S фрейма-посредника F, значение которого необходимо вычислить, базовый фрейм R
Результат: Значение слота, полученное в результате удаленного вызова.
ProxyFrame.Запрос(F,S,R)
{ *Проверяем, нет ли кэшированного значения* }
(2) **if** $F.S \in$ HT **then return** HT.get(F,S)
{ *Получаем удаленную ссылку на базовый фрейм* }
(4) **if** $R = \perp$
(5) $ref \leftarrow \perp$
(6) **else**
(7) **if** $R \in$*множеству фреймов, для которых созданы фрейм-серверы*
(8) $ref \leftarrow$Найти_сервер(R)
(9) **else**
(10) $ref \leftarrow$`new CORBAFrameSrv(`R`)`
(11) Добавить ref в таблицу созданных фрейм-серверов
(12) $rf \leftarrow$*соответствующая F удаленная ссылка*
{ *Произвести CORBA-вызов в соответствии с idl-интерфейсом* }
(14) $res \leftarrow rf$.get(S,ref)
(15) **if** $res \neq \perp$ **then** HT.put(F,S,res)
(16) **return** res

точнее, дистанционному применению знаний или удаленной консультации. Отличие от удаленной консультации в чистом виде состоит в том, что возможно несколько уровней удаленного наследования, а также расширение модели дополнительными знаниями на каждом из уровней. Простая удаленная консультация реализуется наследованием одного фрейма от некоторого удаленного родителя (например, фрейм `Ivanov` на рис. 2.3) с последующим запросом целевого слота, что приводит к процессу обратного вывода на удаленной ЭВМ с заполнением слотов исходного базового фрейма полученными в процессе вывода значениями.

В данном случае важную роль играет тот факт, что фрейм-наследник не должен обладать никакими данными относительно правил, соответствующих родительскому фрейму — достаточно лишь удаленной ссылки. Это свойство мы назовем **прозрачностью удаленного наследования**.

Распределенный прямой вывод реализуется несколько сложнее и накладывает некоторые ограничения, из-за чего практическая его ценность весьма ограничивается. В частности, в посылках правил прямого вывода допускаются только рашие статические ссылки на удаленные фреймы, что обусловлено необходимостью построения сети ссылок до начала вывода. При добавлении правила прямого вывода с удаленным слотом в посылке во фрейм-мир некоторой инсталляции производится запрос к удаленной инсталляции на добавление процедуры-демона к соответствующему удаленному фрейму. При присвоении этому фрейму значения эта процедура осуществляет удаленный вызов, что приводит к попытке применения правила, осуществляемой описанным выше (раздел 1.4.5.3) образом.

Рассмотренный в разделе 1.2 модифицированный алгоритм прямого вывода оказывается чрезвычайно удобным для применения в распределенном случае, так как в нем не

производится построения полной сети Rete, и тем самым не требуется централизованного построения единой сети, включающей в себя ссылки на все используемые в кластере правила вывода. Основным преимуществом алгоритма является возможность распределенного хранения правил прямого вывода, что достигается за счет некоторого снижения производительности. Тем не менее в данном случае ключевой является возможность обеспечения прозрачности удаленного наследования, которая достигается как раз за счет распределенного хранения правил.

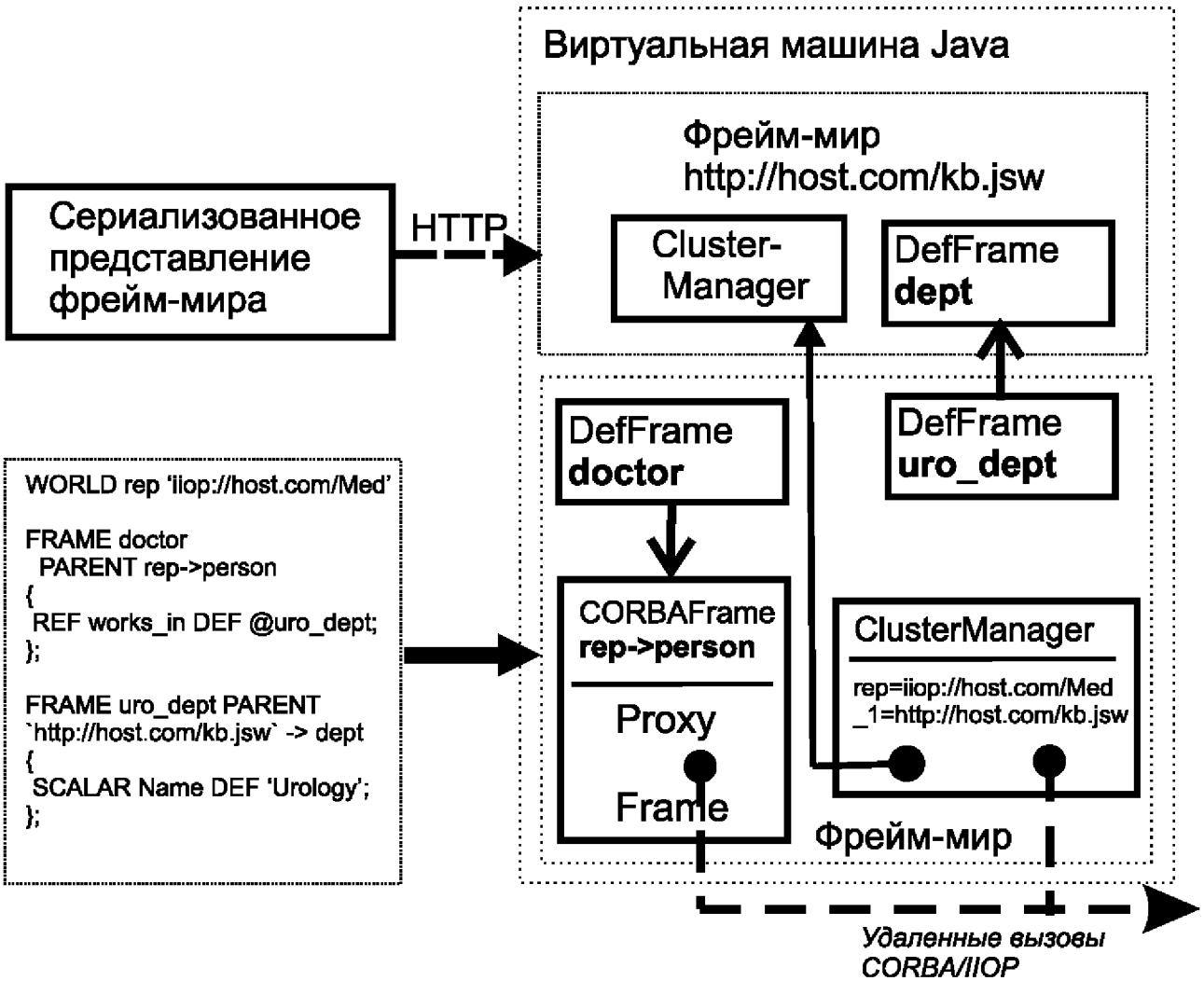


Рис. 2.5. Пример взаимодействия компонентов инструментария при использовании статического и мобильного удаленного взаимодействия

2.4.2. Распределенный вывод на основе включения (inclusion)

Удаленный вызов на основе включения сводится к локальному вызову в рамках одной виртуальной машины Java за счет создания копии удаленного фрейм-мира на локальной ЭВМ. Как только в процессе вывода возникает необходимость вычисления мобильной удаленной ссылки, инструментарий, в случае необходимости, загружает соответствующий фрейм-мир с удаленного узла (как правило, в виде внутреннего сериализованного представления) и создает его локальную копию, после чего работа с экземплярами фреймов и слотов этого фрейм-мира ведется в рамках витруальной машины как с локальными

объектами. Как показано на рис. 2.5, два и более фрейм-миров могут сосуществовать в одной виртуальной машине Java, каждый имея свой набор управляющих объектов (`World`, `ClusterManager` и др.). При присоединении удаленного фрейм-мира выполняется набор вспомогательных операций, корректирующих ряд внутренних структур соответствующего объекта `World`, в частности, для присоединенного фрейм-мира устанавливается тот же объект журналирования `Log`, интерфейс для диалога с пользователем `Asker` и другие свойства, что и для исходного фрейм-мира.

Более подробно процесс вычисления мобильной ссылки иллюстрирует алгоритм 2.3.

Алгоритм 2.3: Алгоритм мобильного вычисления удаленной ссылки

Исходные данные: Имя удаленного фрейм-мира W, имя фрейма в рамках этого мира F.
Результат: Вычисленное значение ссылки типа `TRef` или $\perp$, если фрейм или фрейм-мир не найден.
Вычислить_Мобильно(W,F)
(1) **if** W укороченное имя **then** $W \leftarrow$ соответствующее W глобальное имя
(2) **if** $W \in$ *множеству инстанцированных фрейм-миров*
(3) $R \leftarrow$ ClusterManager.get(W)
(4) **else**
(5) { *Произвести загрузку и инстанцирование* W }
(6) $R \leftarrow$ Загрузить(W)
(7) **if** $R = \perp$ **then return** $\perp$
(8) R.SetWorld(this.getWorld())
(9) **return** R.getFrameManager().get(F)

2.4.3. Статические и мобильные ссылки в языке представления знаний

В языке JFMDL для ссылки на удаленные объекты используется оператор `->`, используемый перед именем фрейма для указания принадлежности его к фрейм-миру, например `world->frame.slot`. Соответственно все ссылки внутри инструментария могут делаться не только на локальные, но и на удаленные объекты.

Механизм вычисления ссылки (удаленный вызов или включение) определяется не синтаксисом ссылки, а свойствами фрейм-мира. Действительно, это весьма целесообразно, так как сложно представить себе ситуацию, когда с одним и тем же фрейм-миром потребуется взаимодействовать двумя разными способами (более того, такое взаимодействие с точки зрения реализации было бы практически невозможно корректно реализовать).

Однако с точки зрения практической реализации важно не только обеспечить возможность взаимодействия двух фрейм-миров посредством ссылок, но и нахождения удаленных фрейм-миров по некоторому имени (т.е. возможность **именования** фрейм-миров в рамках кластера, или даже в рамках глобальной сети Интернет). За именование отвечает компонент, называемый `ClusterManager` (и унаследованный от одноименного абстрактного класса). Инструментарий поддерживает два режима именования:

Локальное именование подразумевает, что каждый фрейм-мир использует свой компонент именования, и ссылки на удаленные фрейм-миры должны быть сообщены в исходном JFMDL-тексте в виде **глобальных имен**.

Кластерное именование означает, что кластер использует общий сервис именования, запущенный на одной из инсталляций, а другие инсталляции, имея ссылку на этот сервис, могут пользоваться услугами **сокращенного именования**.

Для глобального именования некоторого фрейм-мира в рамках глобальной сети используется универсальный локатор ресурса (URL) сериализованного файла фрейм-мира либо доступного по какому-либо протоколу удаленного вызова сервера, например `http://host.com/example.jsw` и `ftp://user:password@host.com/example.jsw` допустимые глобальные имена фрейм-миров, обращение к которым будет производиться посредством включения по протоколам HTTP и FTP соответственно, а `iiop://host.com/Example` и `ior:00110F...` имена, используемые при обращении посредством удаленного вызова по протоколу CORBA (в последнем случае используется внешнее представление уникальной ссылки на CORBA-объект).

Укороченные имена удаленных фрейм-миров могут обеспечиваться сервисом именования кластера, либо вводиться в явном виде при помощи команд `WORLD` *имя* `AS` *глобальное-имя*. Кроме того, в любом контексте могут непосредственно использоваться глобальные имена, например `'http://host.com/ex.jsw'->person`. Во всех случаях способ взаимодействия определяется локатором ресурса, точнее, его протокольной частью: для протоколов HTTP и FTP используется включение, для CORBA/IIOP, RMI и др. удаленный вызов.

2.4.4. Выбор протокола удаленного взаимодействия

Технически, описанный метод построения распределенной фреймовой иерархии на базе инструментария может быть основан на любом транспортном протоколе удаленного вызова, поддерживающем компонентную модель и достаточно развитые интерфейсы вызовов. К таким протоколам относятся Java RMI [121], Microsoft DCOM, DCE, CORBA [20] и др. В текущей реализации использовался механизм CORBA, так как он является стандартом, а также позволяет реализовывать отдельные элементы распределенной фреймовой иерархии на других языках программирования. В частности, расширение инструментария такими средствами, как `CORBAFrame` (доступ к произвольному CORBA-объекту как к фрейму), процедурами-запросами и процедурами-демонами, вызывающими произвольный метод некоторого CORBA-объекта, позволяет получить на базе инструментария полноценную среду программирования, интероперабельную со многими современными распределенными системами, построенными на базе CORBA. Внедрение такого инструментария на CORBA-магистраль позволяет дополнять ее интеллектуальными функциями, используя при этом единый механизм удаленного вызова.

Например, в задачах промышленного контроля за состоянием распределенных объектов зачастую необходим распределенный сбор данных и прозрачный обмен собранными и выведенными данными между узлами системы. Такие задачи как правило решаются применением централизованной модели распределения компонентов системы, в качестве которой как правило выбирают именно CORBA [107].

Использование CORBA не противоречит применению некоторого языка для обмена знаниями между фреймами. В многоагентных системах часто используется стандартный

язык обмена знаниями KQML (Knowledge Query and Manipulation Language) [115][12] или его подмножество, позволяющий строить гибкий обмен сообщениями между агентами. В реализации инструментария проще и эффективнее не использовать специализированный язык внешнего представления, так как в качестве него неявно используется некоторое представление, генерируемое транспортным уровнем CORBA. Однако, для обеспечения совместимости инструментария с другими агентными системами возможна поддержка подмножества KQML, по крайней мере для обеспечения ответов на запросы о значениях слотов. Работа над таким развитием инструментария планируется и позволит использовать предложенную методику распределенной фреймовой иерархии совместно с другими агентными архитектурами. Также, представляет интерес использование для удаленного вызова языков, основанных на XML [122], совместно с протоколом XML-RPC [123], в первую очередь из-за возможности гибкого сопряжения основанных на XML языков друг с другом посредством трансляции с использованием таблиц стилей XSLT.

2.4.5. Дополнительные средства обеспечения онтологической прозрачности и инкапсуляции

При создании базы знаний, лежащей в основе некоторой субиерархии, во многих случаях ставят задачу ее многократного использования со стороны различных удаленных наследников. Для обеспечения удобства использования базы знаний необходимо применять в ней принципы, схожие с принципами модульного и объектно-ориентированного программирования. Среди наиболее важных принципов хотелось бы отметить следующие:

Онтологическая прозрачность, позволяющая пользователю получить максимум информации об онтологии, реализуемой рассматриваемой базой знаний.

Онтологическая инкапсуляция, обеспечивающая внешнюю целостность концептов онтологии за счет сокрытия от внешнего по отношению к базе знаний наблюдателя несущественных деталей реализации и атрибутов и концентрации его внимания на существенных.

Рутинность, означающая эквивалентность онтологического описания вне зависимости от предыстории, т.е. от конкретного вызова соответствующей этому описанию инсталляции.

Для повышения **онтологической прозрачности** инструментарий JULIA содержит средства поддержания более богатого онтологического описания модели. В частности, каждый фрейм, слот, правило или ограничение могут аннотироваться на естественном языке, чтобы при последующем их просмотре было понятно назначение соответствующих элементов модели. Такие текстовые описания вводятся при помощи команды `DESC`, и также могут использоваться для работы механизма *как-* и *почему*-объяснений.

Каждый фрейм или слот фрейма могут быть помечены как `private` в том случае, если создатель модели предполагает, что данный слот/фрейм не представляет интереса как атрибут онтологии, а служит для некоторых внутренних целей. Браузеры онтологий могут учитывать этот флаг для сокрытия несущественных атрибутов модели, тем самым реализуя принцип **онтологической инкапсуляции**.

[12] Часто используются также разновидности KQML, ставшие независимыми стандартами, например FIPA ACL.

Текущая реализация инструментария не содержит браузера онтологии, однако программные интерфейсы JULIA (как CORBA-интерфейсы, так и Java API) позволяют получать доступ к структуре фреймов для просмотра онтологической информации удаленных фреймовых иерархий.

Рутинность применительно к интеллектуальным системам заслуживает особого внимания, так как одной из характеристик интеллектуальной системы является способность к обучению. Поэтому, с точки зрения таких систем может быть желательным, чтобы некоторая база знаний изменяла свое поведение по мере ее использования, таким образом обучаясь по мере своего существования. В этом случае нельзя гарантировать эквивалентность поведения системы с точки зрения формальной семантики, равно как и эквивалентность содержащихся в базе знаний сведений, так как обучение подразумевает модификацию базы знаний. Однако, по-прежнему хотелось бы иметь свойство, аналогичное рутинности (которое можно назвать **семиотической рутинностью**), означающее, что база знаний по прежнему *корректно* (с точки зрения некоторого скорее неформального критерия) отражает предметную область.

В инструментарии JULIA при использовании мобильного наследования достигается полная независимость от предыстории, так как каждый раз используется одно и то же внешнее сериализованное представление. Что касается удаленного вызова, то тут возможно нарушение принципа рутинности за счет потенциальной возможности модификации всех открытых слотов. Более того, в некоторых случаях такая функциональность является полезной, так как позволяет упростить реализацию базы знаний на стороне клиента и минимизировать трафик за счет хранения определенных данных в инсталляции на стороне сервера. Тем не менее, в таких случаях хотелось бы изолировать других клиентов от эффекта изменения базы знаний. Обеспечение этого реализуется в инструментарии двумя путями:

- Введением и использованием еще одного уровня защиты атрибутов (`protected`), запрещающего модификацию их значений со стороны внешних запросов. Это позволяет минимизировать расходы памяти на стороне сервера (для всех внешних запросов используется один и тот же фрейм-мир), а также обеспечить полную рутинность, запретив возможность модификации родительской иерархии.

- Созданием для каждого клиента удаленной инсталляции своей копии фрейм-мира на удаленном сервере. Такое создание может быть привязано к команде именования `WORLD ... AS ... SEPARATE`. Такое решение требует хранения на сервере соответствующего числу клиентов количества фрейм-миров.

2.5. Особенности распределенной фреймовой иерархии

2.5.1. Распределенная фреймовая иерархия и онтологические системы

Ранее в разделах 2.1.2.2, 2.2 и других рассматривалась связь распределенной фреймовой иерархии с понятием онтологии. В данном разделе нам хотелось бы еще раз остановиться на нескольких аспектах этой связи.

При создании любой распределенной интеллектуальной системы (основанной на многоагентной, компонентной или какой-либо другой архитектуре) необходимо, чтобы отдельные компоненты и базы знаний были *согласованы*, т.е. чтобы они использовали

общий словарь предметной области и подразумевали некоторые общие свойства используемых понятий. В классических агентных архитектурах онтология как раз и является внешним соглашением об "общем терминологическом языке", используемом для обмена сообщениями.

Одной из наиболее используемых моделей онтологии является таксономическая структура понятий предметной области с описаниями атрибутов и отношений между понятиями, а также некоторых знаний (аксиом) относительно этих понятий. По сути дела, фреймовая иерархия как раз и определяет такую таксономическую структуру, в которой фреймы соответствуют понятиям предметной области, а знания представляются множеством правил и ограничений. Таким образом, архитектура фреймовой иерархии является **автоонтологичной** в том смысле, что при создании фреймовой субиерархии нет необходимости во внешнем явном описании терминов предметной области, так как сама иерархия является самодокументированной в онтологическом смысле. С точки зрения создания прикладных систем это означает, что при создании базы знаний, наследуемой от некоторой родительской иерархии, достаточно посмотреть на структуру фреймовой модели и использовать соответствующие атрибуты.

С другой стороны можно сказать, что распределенная фреймовая иерархия представляет собой удобную технологию для реализации и использования онтологических описаний, альтернативную подходу с переносимой трансляцией онтологий (portable ontology translation), предложенному в [76]. В последнем подходе онтологическое описание составляется на архитектурно-независимом стандартном языке представления знаний (например, KIF [114]) и затем при помощи определенных правил трансляции преобразуется во внутреннее представление каждого конкретного агента. Распределенная фреймовая иерархия представляет унифицированный подход, при котором используется единое представление знаний во всех узлах распределенной системы, что позволяет также использовать внутреннее представление знаний для обмена между узлами, вместо введения еще одного уровня стандартизации взаимодействия наподобие языка KQML [115]. Это несколько ограничивает круг решаемых задач за счет введения вполне определенной семантики применения знаний, основанной на классическом прямом и обратном логическом выводе, однако позволяет в этом классе решать задачи более эффективно за счет наличия готовых архитектурных и программных решений поддержки онтологической системы.

2.5.2. Множественное или псевдомножественное наследование как модель доски объявлений

В задачах распределенного накопления и использования знаний часто возникает ситуация, когда для решения некоторой задачи необходимо применить знания, содержащиеся в нескольких базах знаний. Классической моделью в этом случае является модель **доски объявлений** [93], в которой статические знания хранятся в единой рабочей памяти, разделяемой несколькими процессорами вывода, использующими свои базы знаний.

В модели распределенной иерархии применение базы знаний для решения некоторой задачи со своим набором атрубутов производится наследованием фрейма-экземпляра от некоторого удаленного фрейма. Таким образом, если система допускает множественное наследование, то такое использование различных баз знаний для одного набора значений атрибутов будет достигаться множественным наследованием от удаленных фреймов. При использовании обратного вывода в этом случае для некоторого атрибута будут по очереди применяться правила из различных удаленных фреймовых иерархий, пока значение

атрибута не будет определено.

В случае множественного наследования достаточно жестко определен порядок применения правил фреймов-родителей. Для более гибкого управления процессом применения знаний из различных источников можно воспользоваться псевдомножественным наследованием (см. раздел 1.6.2.1), присоединяя целевой фрейм по очереди к различным родителям (в том числе многократно в соответствии с некоторым критерием до тех пор, пока не будет получено требуемое решение, или очередной круг не будет вызывать новых изменений в состоянии слотов).

Отметим, что такое множественное и псевдомножественное наследование может быть как статическим, так и мобильным. Особенный интерес представляет мобильное псевдомножественное наследование, при котором производится **последовательное множественное включение** различных удаленных фрейм-иерархий для поочередного применения содержащихся в них знаний к решению задачи.

2.5.3. Распределенное решение проблем и синтез решения

В общем случае под распределенным решением проблем подразумевается разбиение исходной задачи на подзадачи, с последующим распределением поздазач по различным процессорам и синтезом решения на основании полученных результатов. Распределенная фреймовая иерархия является частным случаем распределенного решения задач, при котором часть логического вывода делегируется другому процессору в соответствии с содержащимися в нем знаниями и описываемым участком распределенной фреймовой иерархии.

При рассмотрении распределенного решения задач накопления и использования знаний можно выделить два ортогональных случая [99]:

- Задачи с **непересекающимися проблемными доменами**. В этом случае задача разбивается на набор подзадач из разных предметных областей, каждая из которых может решаться своей экспертной системой.
- Задачи с **одинаковыми проблемными доменами**. В этом случае одна и та же задача может решаться несколькими экспертными системами на основе общего множества входных атрибутов, с последующим сравнительным анализом результатов.

В более общем случае задача может разбиваться на произвольную комбинацию описанных выше подходов.

Описываемая в данной работе модель может эффективно применяться в обоих случаях. В первом случае как правило достаточно использовать модель (псевдо)множественного наследования для имитации доски объявлений (см. раздел 2.5.2). Так как атрибуты в базах знаний с непересекающимися доменами не пересекаются (или почти не пересекаются), то применение логического вывода с очередной родительской иерархией будет приводить к решению очередной подзадачи. Последним шагом в выводе будет сравнительный анализ полученных результатов, который также может проводиться путем наследования от некоторой родительской иерархии с соответствующей базой знаний.

Альтернативным способом решения может быть разбиение исходных данных по набору независимых фреймов, каждый из которых наследуется от своей иерархии и затем подвергается логическому выводу для получения результатов. В случае обратного

вывода такой подход может оказаться удобнее, так как исходные данные отсутствуют, и необходимо просто собрать интересующие атрибуты у набора фреймов, унаследованных от соответствующих иерархий.

В случаях с совпадающими доменами требуется получить решения одной и той же задачи с использованием различных баз знаний (т.е. провести логический вывод для одного и того же набора исходных данных), с целью дальнейшего сравнения полученных результатов на основе простой процедуры или логического вывода. В этом случае требуется создать несколько копий исходного фрейма и использовать различные копии в качестве наследников нескольких родительских иерархий. При использовании обратного вывода этот случай будет мало отличаться от предыдущего — лишь на этапе анализа данных нужно будет учитывать возможные противоречия, возникшие из-за различных решений исходной задачи различными экспертными системами. Проблема синтеза решения на основе набора возможно противоречивых решений независимых экспертных систем представляет самостоятельный интерес, и может решаться математическими или вероятностными методами [108] (особенно в случае, если результаты независимых систем представляют собой нечеткие значения), либо на основе явного представления знаний и анализа результатов эксплицитно-интеллектуальными методами. В последнем случае использование распределенной фреймовой иерархии оказывается чрезвычайно удобным, так как проблема синтеза решения укладывается в ту же самую модель, что и поиск набора решений.

2.5.4. Репозитарии знаний

В некоторых случаях может оказаться полезным иметь возможность обмениваться по сети динамическими знаниями в форме отдельных правил, хранящихся в некотором внешнем представлении. Статический обмен подразумевает наличие в двух или более узлах сети активных JULIA-инсталляций, в то время как динамические знания могут быть пассивно объединены в репозитарий знаний, состоящий из набора кластеризованных продукционных правил (точнее, процедур-демонов и процедур-запросов), которые могут загружаться некоторым процессором вывода JULIA по мере необходимости в процессе вывода (т.н. **загрузка правил по мере необходимости**, on-demand rule loading) .

Использование репозитария знаний в некотором смысле напоминает включение, но имеется и ряд существенных отличий:

- Правила могут хранится в репозитарии самостоятельно, не обязательно в виде фреймовой иерархии. Таким образом репозитарий не включает (в некоторых случаях излишних) знаний о статике предметной области.

- Правила могут хранится в некотором внешнем представлении, например, в виде KIF-предложений или на XML-подобном языке, что допускает использование одного репозитария различными системами.

Правила могут храниться в репозитарии в одном из следующих видов:

- **В сериализованной форме** в виде файла, доступного по протоколу HTTP или FTP. В этом случае во фреймовой модели на месте правила находится процедура-заглушка, которая, в случае активизации правила загружает его тело по сети для последующего выполнения. Установкой опции загрузки правил по мере надобности (`OPTION RuleOnDemandLoading='`*url*`'`) перед компиляцией исходного JFMDL-текста

можно автоматически формировать модель с заглушками и наборы правил для помещения на удаленный сервер.

- **В виде пассивного XML-репозитария**, который загружается во фреймовую модель до начала логического вывода.

- **В виде активного сервера**, который по запросу выдает последовательность правил, применимых к указанному слоту. При запросе значения слота выполняется запрос к серверу знаний, который по очереди возвращает правила в некотором внутреннем представлении (XML или сериализованный поток байтов). Такой репозитарий приближает предлагаемую модель к компонентной архитектуре с активными источниками знаний [49, 99].

- **В виде набора сериализованных файлов с индексацией**. Этот способ позволяет избежать активного сервера, но тем не менее использовать его функциональность за счет использования индексного файла, указывающего соответствие сериализованных файлов и слотов модели.

Первый способ удобно использовать для пассивного хранения знаний на удаленном сервере, который обеспечивает доступ только по стандартному протоколу HTTP, однако в этом случае необходимо заранее знать соответствие имен сериализованных файлов и правил-заглушек. Обычно этот способ позволяет эффективно минимизировать трафик при удаленной консультации, так как в начале консультации загружается только "скелет" модели (структура фреймов с заглушками вместо правил), а правила загружаются по мере необходимости в процессе вывода. Так как обычно при логическом выводе в реальных системах используются не все правила, а только сравнительно небольшое их количество, то такой подход позволяет уменьшить объем передаваемых по сети знаний, а также ускорить процесс начала консультации (так как нет необходимости ждать загрузки всех правил). Однако, объем всей базы знаний в этом случае незначительно повышается (примерно на 10%), за счет дополнительных правил-заглушек.

Предпоследний способ (с использованием активного сервера) является наиболее мощным и служит для создания таких репозитариев динамических знаний, которые по той или иной причине не могут быть интегрированы в распределенную фреймовую иерархию. Основными причинами могут являться возможность применения знаний к различным фреймам, для которых по некоторым соображениям неэффективно использовать наследование от общего родителя. Кроме того, репозитарий может использоваться совместно с основными правилами, присоединенными к слоту, таким образом дополняя основной способ распределения знаний. Более того, сервер динамических знаний не осуществляет процесс логического вывода — его роль сводится к выдаче набора правил по запросу, таким образом сервер не берет на себя излишней "вычислительной ответственности".

Хранение знаний в виде набора сериализованных файлов с индексацией в инструментарии пока не реализовано. В этом случае вместо активного сервера используется индексный файл, указывающий соответствие имен сериализованных файлов и слотов фреймовой модели. Этот индексный файл загружается до начала вывода, и затем используется каждый раз для запроса соответствующих файлов с удаленного сервера по протоколу HTTP. Тем не менее, в отличие от активного сервера, возможность для динамического многокритериального поиска правил отсутствует.

2.6. Выводы к главе 2

В данной главе была предложена архитектура распределенной фреймовой иерархии для построения интеллектуальных систем распределенных накопления и использования знаний. Эта архитектура является синтезом многоагентного подхода, классического фреймового представления знаний, компонентной архитектуры, классических распределенных гетерогенных систем и теории онтологических систем. Являясь по сути дела коллаборативной делиберативной агентной архитектурой с двумя типами взаимодействия (мобильное и статическое), предложенная архитектура представляет собой удобное архитектурное и инструментальное средство построения распределенных онтологических описаний, позволяя применить имеющийся в теории онтологий и объектно-ориентированном подходе теоретический аппарат для построения реальных распределенных систем, основанных на знаниях.

Предложенная архитектура основывается на распределении иерархии по различным узлам сети за счет введения двух типов удаленных ссылок между различными фреймовыми субиерархиями: статических и мобильных. Статическая ссылка вычисляется путем удаленного вызова, мобильная — посредством включения. На основе ссылок строятся удаленные агрегация и наследование, являющееся ключевым средством наследования и повторного использования знаний.

Для удаленной системы определяется два типа семантики: основанная на эквивалентном логическом выводе в порожденной распределенной системой единой фреймовой иерархии, и распределенная семантика на множестве функций состояния индивидуальных подсистем, позволяющая определять тонкости мобильного и статического взаимодействия и отслеживать процессы передачи знаний между узлами системы. Демонстрируется эквивалентность двух семантик, что позволяет сделать утверждение об эквивалентности мобильного и статического методов вычисления удаленной ссылки с точки зрения процессов вывода.

Практическая реализация предложенной архитектуры на базе инструментария JULIA осуществляется с использованием CORBA путем введения фреймов-посредников для удаленного взаимодействия. Кроме того, в качестве альтернативы включению предлагается использование репозитариев знаний, что позволяет включить компонентную архитектуру в реализованное многообразие решений. Также показано, что архитектура распределенной фреймовой иерархии может быть эффективно использована для моделирования классической архитектуры доски объявлений, а также в задачах распределенного решения задач и синтеза решения.

3. Применение инструментария для решения практических задач

Инструментарий JULIA представляет богатый набор средств для построения интеллектуальных систем различных конфигураций. Ниже кратко рассмотрены некоторые возможности для практического применения инструментария, некоторые из которых уже реализованы в виде готовых программных систем.

3.1. Удаленные консультации. Экспертная система продвижения Интернет-ресурсов PromoWeb

Удаленная консультация представляет собой простейшую форму удаленного использования знаний из некоторой базы знаний для решения задачи. Особенностью удаленной консультации является необходимость передачи по сети динамических знаний (в модели толстого клиента), либо удаленного применения знаний для решения задачи (в модели тонкого клиента). Более подробно вопросы, связанные с удаленной консультацией в модели толстого и тонкого клиента были рассмотрены в [49, 99].

В предложенной терминологии удаленное применение знаний в модели тонкого клиента соответствует статическому, а передача знаний — мобильному удаленному выводу. Таким образом, в инструментарии JULIA изначально заложена поддержка обоих моделей удаленной консультации. Кроме того, инструментарий может использоваться в локальном режиме, а для обеспечения клиент-серверной функциональности могут использоваться традиционные Java-ориентированные Интернет-технологии: апплеты и сервлеты. Благодаря относительно небольшому размеру (объем локальной библиотеки времени выполнения составляет менее 100 Кб) инструментарий может эффективно использоваться не только на стороне сервера как Java-сервлет, но и в виде Java-апплета в клиентском браузере.

Одним из примером использования JULIA для построения удаленной интеллектуальной системы является экспертная система по продвижению веб-ресурсов **PromoWeb** [27], разработанная автором совместно с Крастелевой И.Е. в рамках гранта Российского государственного университета инновационных технологий и предпринимательства. Эта система представляет собой Java-апплет (см. рис. 3.1), функционирующий на клиентском компьютере, включающий в себя среду времени выполнения JULIA и некоторые специализированные расширения на языке Java. При запуске апплет загружает основную фреймовую модель, при этом для минимизации траффика и исходного времени ожидания используется загрузка правил по необходимости.

База знаний основана на обратном выводе и рассматривает применительно к задаче пользователя целый спектр различных методов интернет-рекламы. На рисунке 3.2 показан фрагмент фреймовой иерархии, отвечающий за различные методы продвижения веб-ресурса. Все соответствующие этим методам фреймы унаследованы от одного родителя, который определяет общие для всех методов рекламы свойства: количество затраченных на рекламу ресурсов (в том числе денежных), эффективность применения данного вида рекламы для различных категорий веб-ресурсов, формируемые в результате

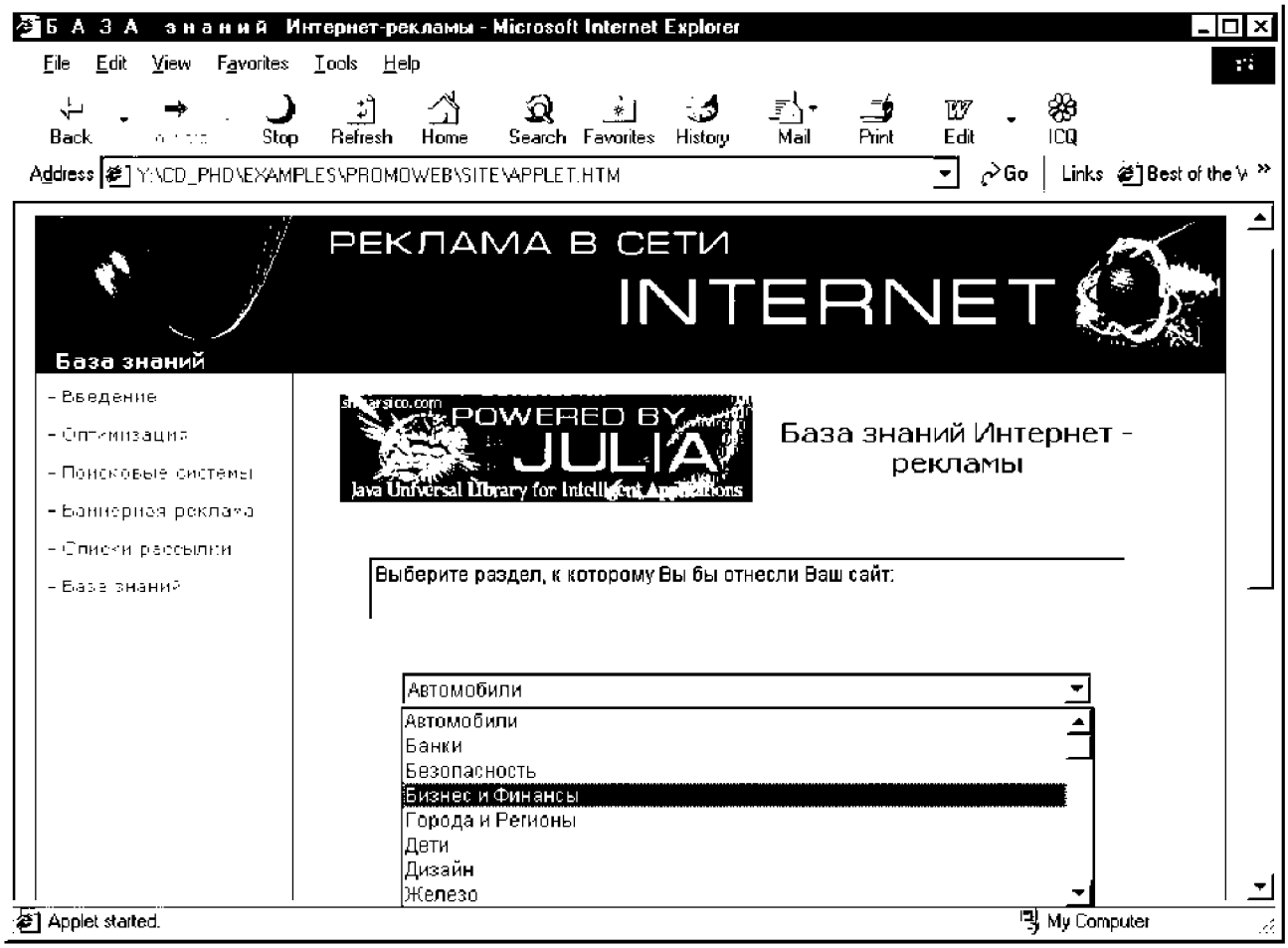


Рис. 3.1. Пользовательский интерфейс системы **PROMOWEB** в виде апплета в браузере

вывода рекомендации и т.д.

В процессе вывода пользователю задается набор начальных вопросов относительно рекламируемого ресурса, в том числе его тематическая направленность, размер отведенных на рекламную кампанию средств и т.д. Информация о ресурсе заносится в соответствующий ресурсу фрейм, который затем классифицируется с помощью динамического наследования в одну из предусмотренных категорий ресурсов в соответствии с показанной на рисунке 3.3 концептуализацией[1]. Затем с фреймом-экземпляром ресурса поочередно соотносятся различные методы интернет-рекламы, в результате чего формируются списки рекомендаций по каждому из методов, а также сравнительный рейтинг эффективности применения, в соответствии с которым на заключительном этапе производится распределение денежных средств и других ресурсов.

Таким образом, результатом консультации является набор рекомендаций по оптимизации сайта, список конкретных действий, направленных на повышение посещаемости, а также распределение исходной суммы по различным методам платной интернет-рекламы. В процессе консультации специально разработанная на Java программа подбора ключевых слов (см. приложение В.2.2) в реальном времени анализирует ключевые слова 25 самых популярных сайтов в указанной категории и на основании этого формирует рекомендуемый список ключевых слов сайта.

[1]Говоря более точно, вопросы задаются в процессе обратного вывода, целью которого является классификация фрейма-экземпляра рассматриваемого ресурса.

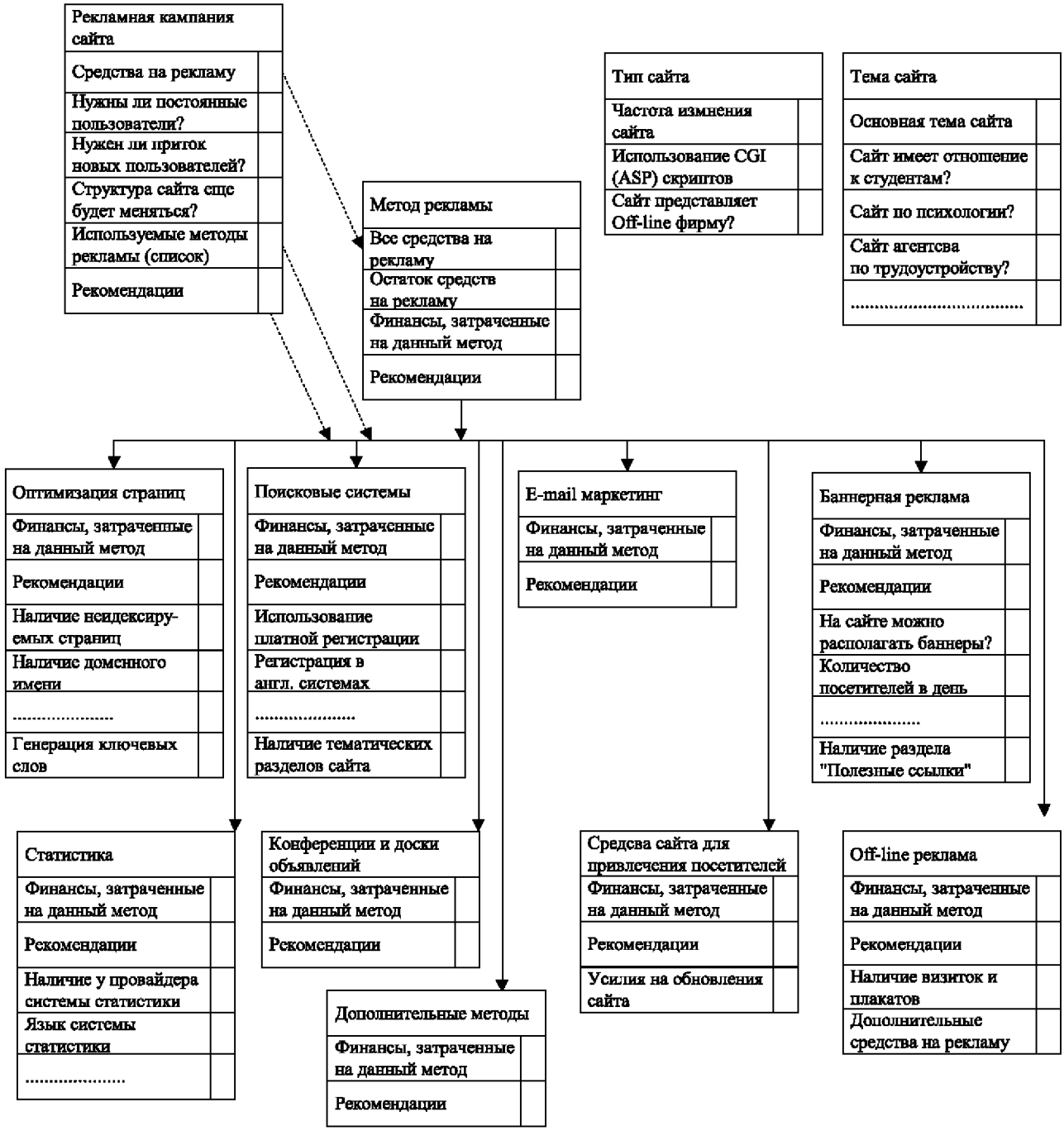


Рис. 3.2. Структура фрагмента фреймовой иерархии базы знаний системы PromoWeb, концептуализирующей различные методы Интернет-рекламы.

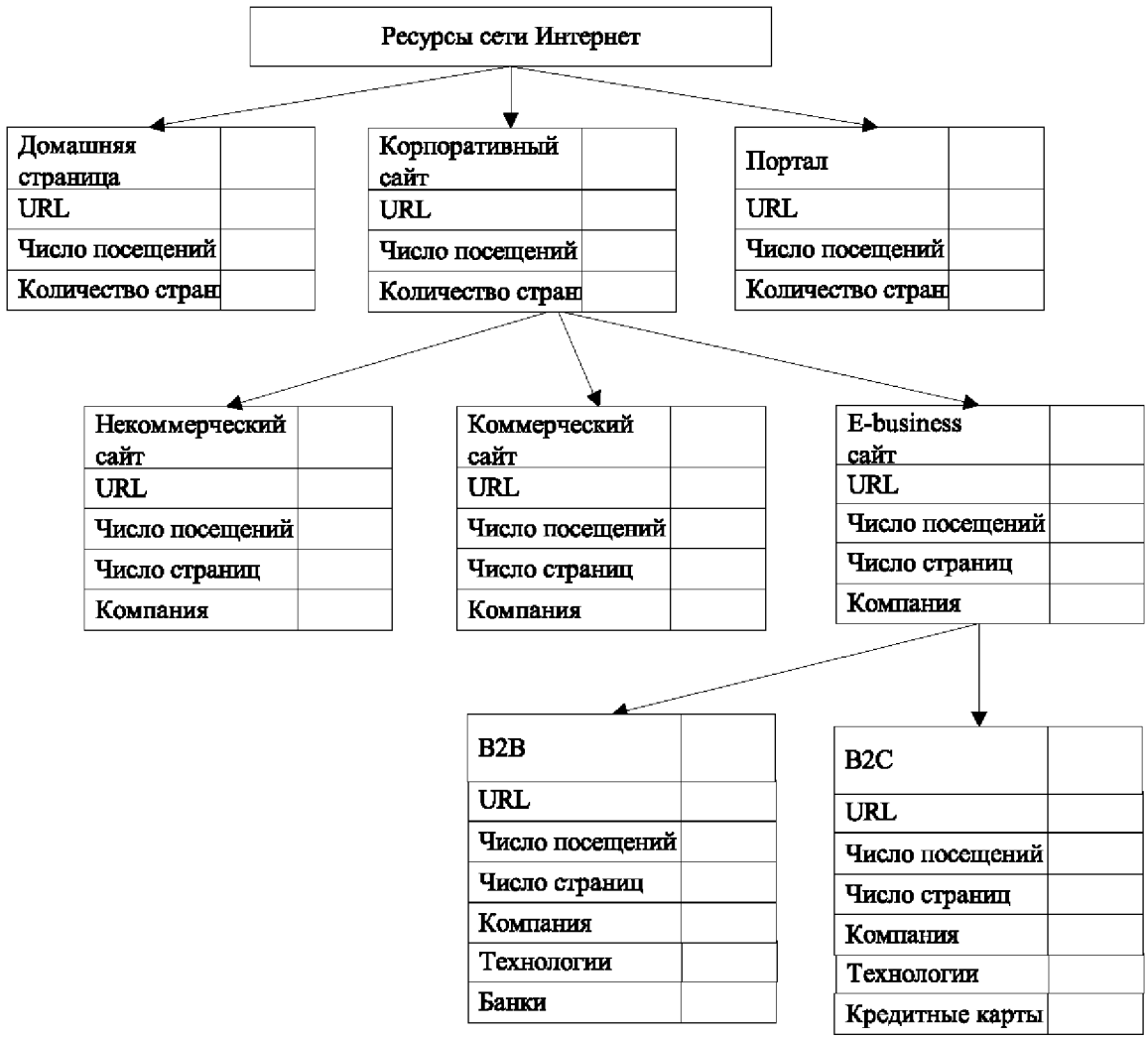


Рис. 3.3. Структура фрагмента фреймовой иерархии системы PromoWeb, концептуализирующей классификацию веб-сайтов

С использованием системы PromoWeb была проведена рекламная кампания различных сайтов русскоязычного сегмента Интернет, в том числе сайта-сообщества `http://www.expat.ru`, проекта "Русское зарубежье" `http://www.zarub.db.irex.ru`, портала IT-специалистов `http://www.kworker.com`, а также кафедры 502 МАИ `http://www.mai.ru/ k502`. В качестве примера, на рис. 3.4 показаны графики повышения посещаемости сайта кафедры 502 МАИ после проведения рекламной кампании, спланированной при помощи системы. Более подробно структура системы и результаты ее применения описаны в [27, 28].

3.2. Системы дистанционного обучения

Идея применения интеллектуальных систем в компьютерном обучении не нова [21, 47]. Экспертные системы представляют готовые способы представления модели обучаемого и преподавателя в виде того или иного способа представления знаний; в этом случае процесс обучения может представляться логическим выводом по некоторой базе знаний, в ходе которого происходит либо собственно обучение (управляемая выводом демонстрация динамически создаваемого материала), либо в том или ином виде контроль знаний.

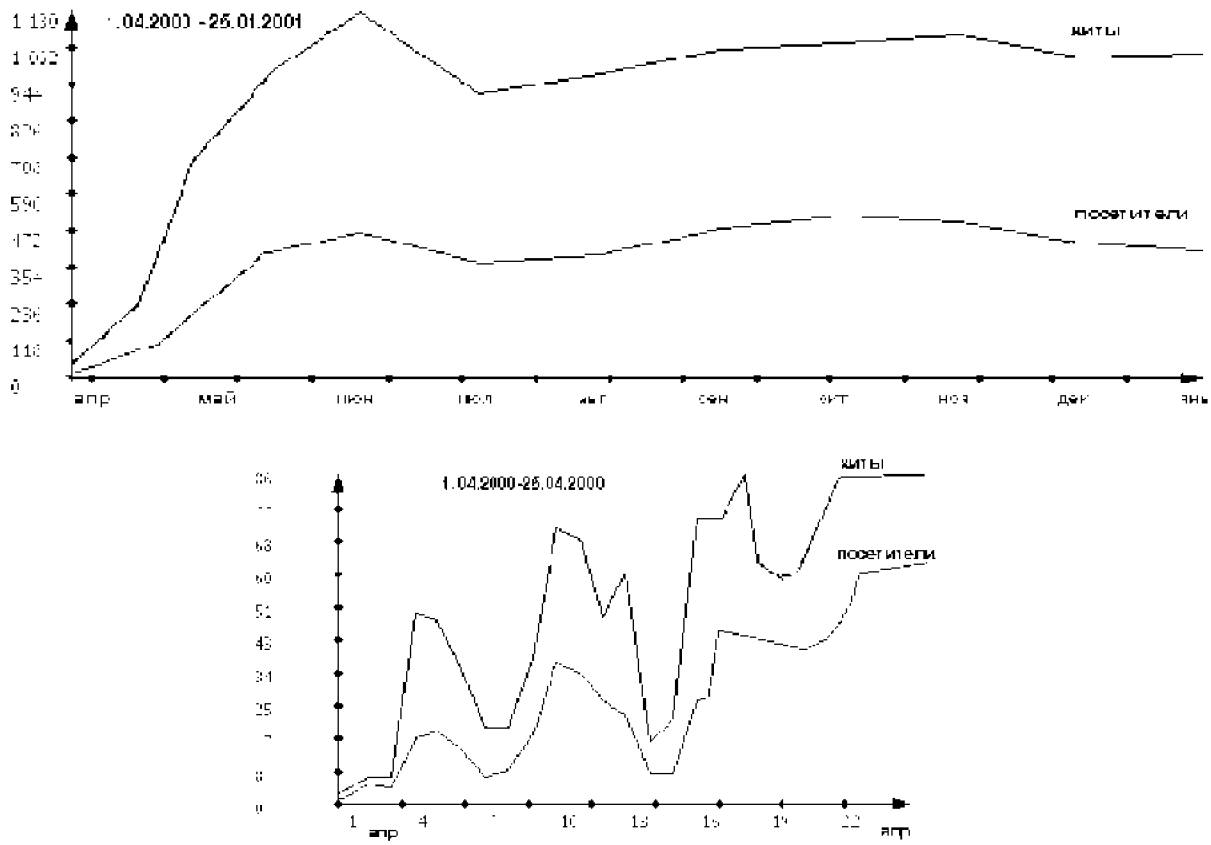


Рис. 3.4. Статистика посещений сайта кафедры 502 МАИ после проведения рекламной кампании, спланированной системой **PromoWeb**.

Продукционно-фреймовое представление знаний является идеальным для использования в задачах обучения. При этом предлагается использовать следующие основные положения:

- Предметная область концептуализируется в виде таксономической онтологии (или онтологической системы) и представляется в виде иерархии (возможно, распределенной) фреймов, образующей совместно с продукционными правилами **модель преподавателя**. При этом фреймовая иерархия с набором отношений наследования, агрегации, а также некоторыми дополнительными специализированными отношениями (следования, семантической связности, релевантности, зависимости и др.) задают структуру курса, а продукционные правила отражают возможную динамику изучения предметной области, т.е. так или иначе неявно определяют последовательность подачи материала, а также способы оценивания знаний студента и другие навыки преподавателя, не относящиеся к структуре изучаемого курса. В динамической составляющей модели преподавателя также можно выделить предметно-независимую часть знаний об основных методах обучения и контроля знаний, которая может быть вынесена в отдельную мета-онтологию.

  Разработка формальной онтологической модели курса целесообразна также для построения гипертекстового наполнения [2]. Как показано в [14], структурирование

[2]Уместно также упомянуть о понятии **гипертекстовой базы знаний** [41], в котором гипертекст

и стратификация курса является самостоятельной задачей, которая решается на первых этапах формирования модели предметной области, которая затем находит отражение как в гипертекстовом наполнении, так и в активной базе знаний, управляющей процессом обучения.

- **Модель знаний обучаемого** формируется в виде набора значений слотов фреймов, отвечающих за различные разделы курса (или унаследованных от них фреймов-экземпляров). Эти слоты могут заполняться в результате предварительного тестирования, либо в процессе обучения при ответе на контрольные вопросы.

- Тестирования с целью формирования модели знаний обучаемого может производиться на основании **базы вопросов**, выбираемых из некоторого репозитария, построенного по закрытой схеме (т.е. с фиксированным набором вариантов ответа), либо генерируемых динамически на основании параметризуемых шаблонов или специализированных алгоритмов.

3.2.1. Адаптивное тестирование, управляемое логическим выводом

Адаптивное тестирование — это широкий класс методик тестирования, предусматривающих изменение последовательности предъявления заданий в самом процессе тестирования с учетом ответов испытуемого на уже предъявленные задания. В узком смысле к адаптивному тестированию обычно относят особые, главным образом компьютерные, алгоритмы предъявления заданий, построенные для пунктов теста, предварительно отобранных с помощью соответствующих моделей и методов *анализа пунктов* и других процедур, основанных на психометрической теории "*вопрос-ответ*" (*item-response theory*) [86]. В широком смысле под адаптивным тестированием можно понимать любую оперативную модификацию теста, приспосабливающую его к продемонстрированному уровню знаний испытуемого; при этом можно сказать, что последующий вопрос формируется на основе предыдущих.

Основное назначение адаптивного тестирования — сокращение числа предъявляемых заданий за счет исключения тех, ответ на которые уже можно однозначно предсказать за счет предыдущих ответов данного испытуемого. В тестах с адаптивным алгоритмом выделяется 4 базовых элемента:

1. набор заданий,
2. механизм выбора заданий из набора,
3. процедура подсчета показателей,
4. правило окончания теста.

Например, так называемое **гибкоуровневое** тестирование использует установленный набор заданий, расположенных по трудности, с предъявлением вначале заданий средней трудности, а затем с повышением или понижением уровня трудности в зависимости от правильности или неправильности ответа. В **страдаптивном** тестировании задания располагаются в виде **страт** различной степени сложности (обычно 9 страт, или уровней).

определенной структуры может самостоятельно рассматриваться как способ хранения и передачи знаний, ориентированных на восприятие человеком

Когда на задание получен неверный ответ, следующее берется из низшего страта; когда ответ правилен — из верхнего.

Как видно из краткого обзора методов адаптивного тестирования, большинство из них основано на достаточно четко определенных алгоритмах выбора вопросов и рейтингования. В то же время известно, что человек-преподаватель как никто другой хорошо справляется с задачей оценки уровня и распределения знаний учащегося, что наводит на мысль о возможности формализации знаний преподавателя в виде экспертной системы, управляющей тестированием [47]. Сам ход адаптивного тестирования — вопрос-ответ, при котором каждый последующий вопрос зависит от предыдущего — также является характерным режимом работы для систем обратного вывода. Экспертная система позволяет легко реализовать не только упомянутые выше подходы к адаптивному тестированию, но и более сложные механизмы, управляемые знанием преподавателя о предметной области и ее структурой.

В [34, 35] описана созданная на базе JULIA система адаптивного интеллектуального компьютерного тестирования **IntelliTest**[3], функционирующая в среде Интернет. Система функционирует на стороне сервера в виде Java-сервлета, и представляет собой структурированную базу знаний, в ходе обратного вывода по которой тестируемому задаются вопросы, выбираемые из большого множества вопросов, или генерируемых динамически по шаблону с помощью специально разработанного модуля. В зависимости от результатов ответа на предыдущие вопросы формируется стратегия дальнейшего тестирования: тестируемому задаются те вопросы, ответы на которые система не может спрогнозировать на основе уже полученных ответов. Разработана оригинальная база знаний и наборы вопросов для тестирования в области веб-технологий.

3.2.2. Система обучения логическому программированию LPTutor на основе гипертекстового курса с интеллектуальной навигацией

Адаптивное управляемое выводом тестирование позволяет создать модель знаний обучаемого в виде множества значений слотов различных уровней — от индивидуальных ответов на вопросы, до уровня знаний по тем или иным разделам курса и по всему курсу в целом. Вполне естественно использовать эти знания для планирования дальнейшей индивидуальной стратегии обучения студента, которое может производиться также на основе логического вывода в той же самой базе знаний, которая использовалась для тестирования. Более того, возможно таким образом организовывать различные схемы обучения, при котором навигация по разделам осуществляется в процессе прохождения тестирования, по главам или каким-либо другим образом (см. рис. 3.8).

Рассмотренная идея легла в основу системы обучения логическому программированию **LPTutor**[4]. В качестве гипертекстового наполнения курса использовался незначительно переработанный курс логического программирования Морозова М.Н. [38], разработанный в лаборатории систем мультимедиа Марийского государственного технического университета. Гипертекстовый курс все время доступен обучаемому в верхней половине экрана (см. рис. 3.5), а в нижней половине расположен управляющий прохождением курса аплет с загруженной копией инструментария JULIA и управляющей базой знаний, который, в зависимости от состояния, задает вопрос, отображает результаты тестиро-

[3]Система разработана под руководством автора студенткой-дипломницей МАИ Малкиной О.И.

[4]Система разработана под руководством автора студенткой-дипломницей МАИ Сикачевой А.Ю.

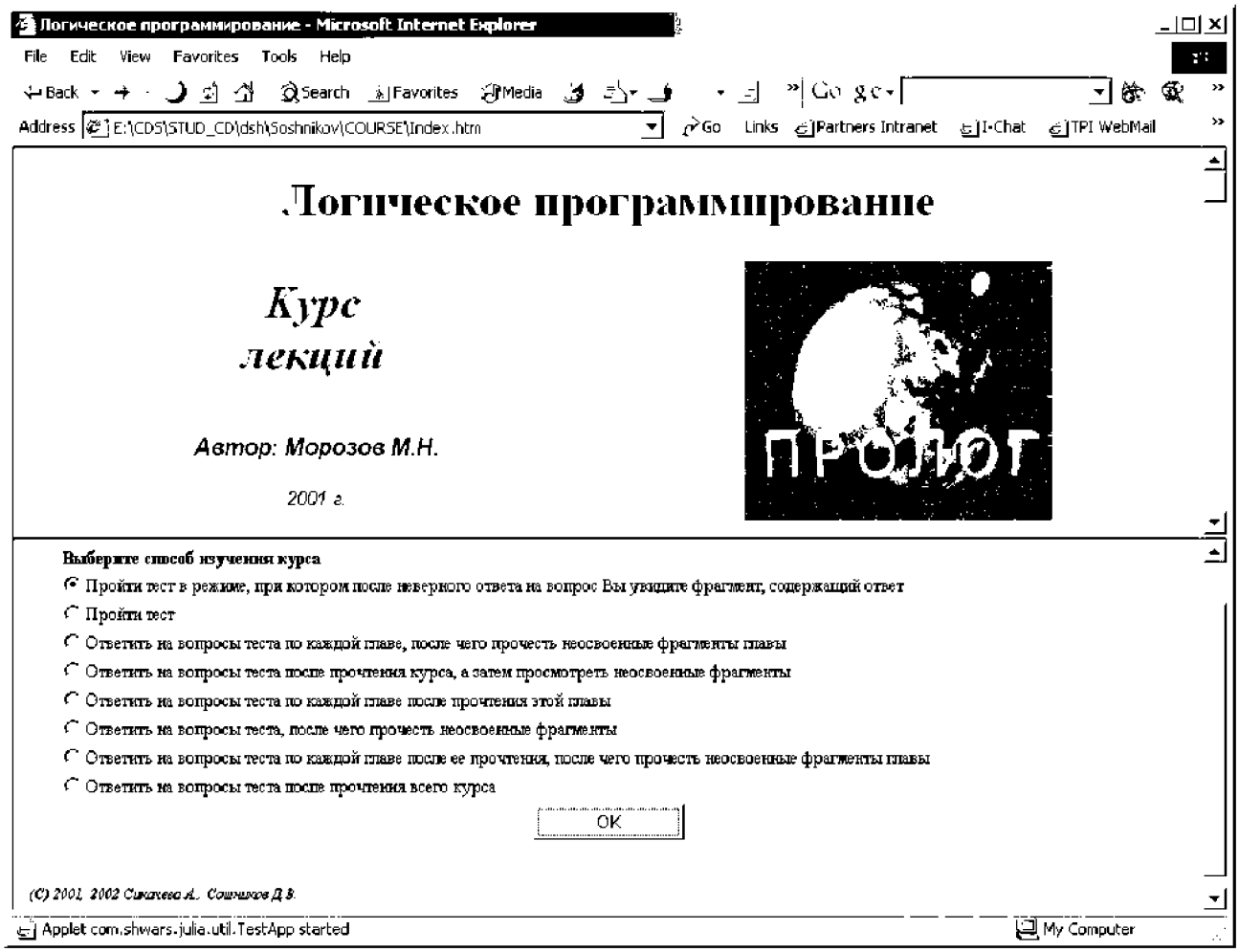


Рис. 3.5. Пользовательский интерфейс интеллектуальной системы обучения логическому программированию **LPTutor**.

вания по всему курсу или подразделам, выдает рекомендации либо ожидает нажатия клавиши для продолжения интеллектуальной навигации. Реализация интеллектуальной составляющей в виде апплета позволит использовать систему в индивидуальном режиме в качестве отдельно распространяемого курса (на CD-ROM), а также в среде Интернет, без существенных требований к веб-серверу.

Структура курса логического программирования представлена статической фреймовой моделью, показанной на рисунках 3.7 и 3.6. Основные категории фреймов представлены протофреймами **курс**, **раздел курса**, **страница курса** (подраздел) и **вопрос**, которые в перечисленном порядке унаследованы друг от друга, и от которых в свою очередь унаследованы реальные фреймы-экземпляры (отношение наследования показано на рис.3.7 сплошными стрелками). Кроме того, каждый верхний уровень содержит в себе список, агрегирующий набор экземпляров фреймов нижнего уровня (показано на рис.3.7 пунктирными стрелками). Конкретные экземпляры фреймов в процессе тестирования заполняются значениями, соответствующими уровню знаний обучаемого по соответствующему разделу, рекомендациями по обучению, а также списком из набора страниц, которые должны быть просмотрены для устранения недочетов.

В зависимости от выбранного в начале курса режима обучения (см. рис. 3.5), эти страницы могут демонстрироваться сразу после неверного ответа на вопрос, в конце гла-

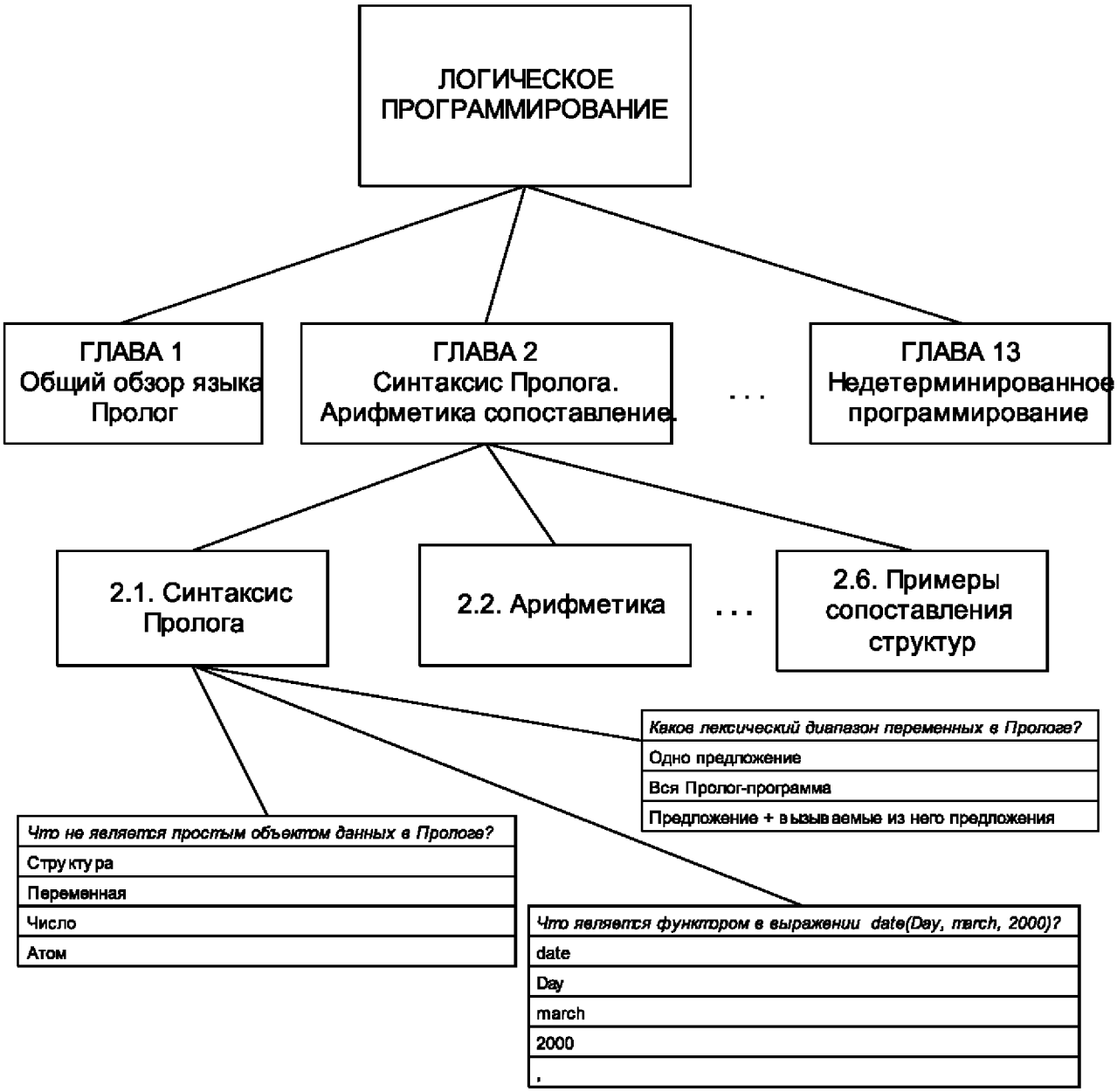


Рис. 3.6. Структура курса логического программирования и ее отображение в виде фрейм-мовой модели

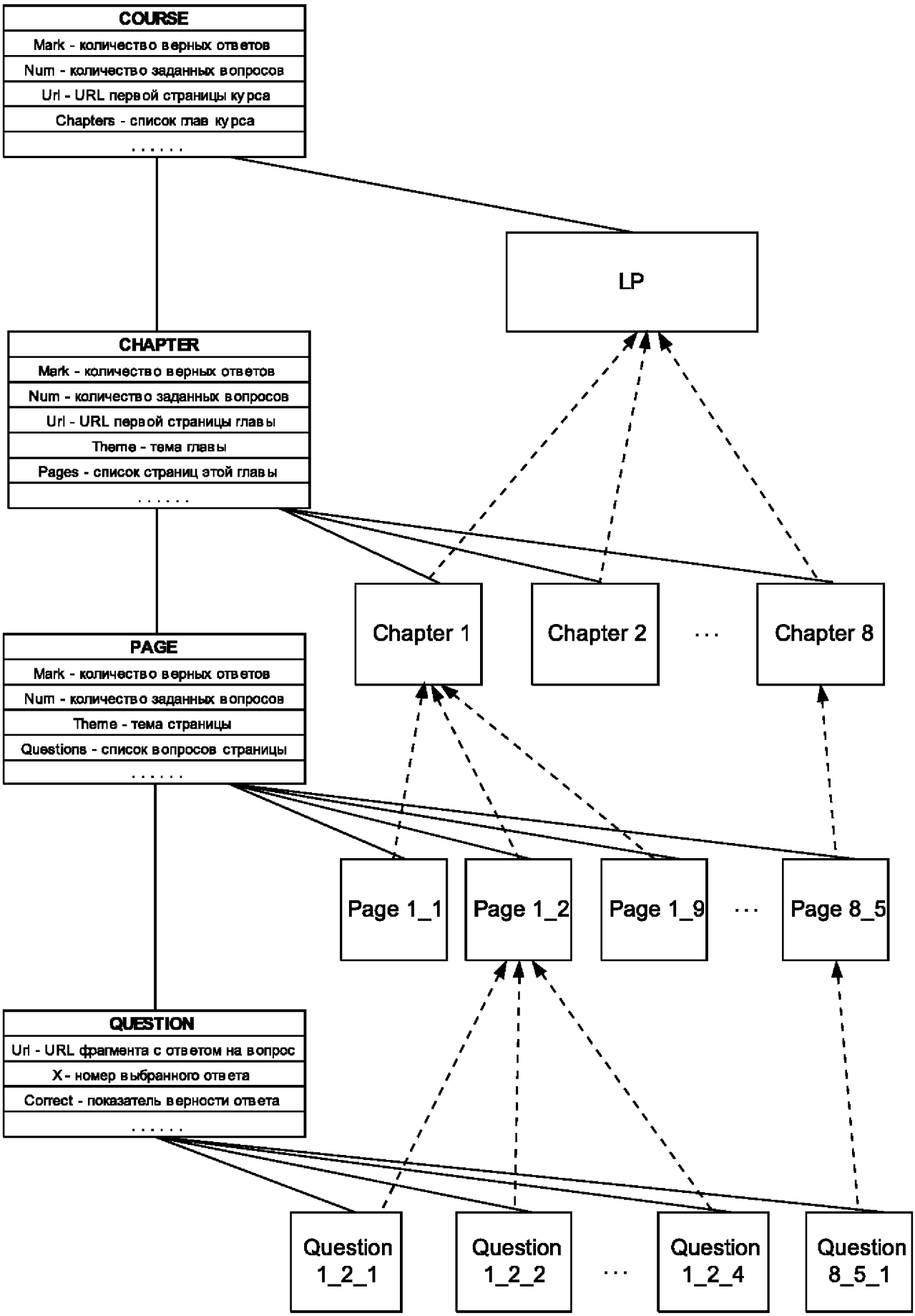


Рис. 3.7. Структура фреймовой иерархии интеллектуальной системы обучения логическому программированию LPTutor.

вы, в конце всего тестирования и т.д. Всего предусмотрено 8 различных вариантов прохождения курса, два из которых показаны на рис. 3.8.

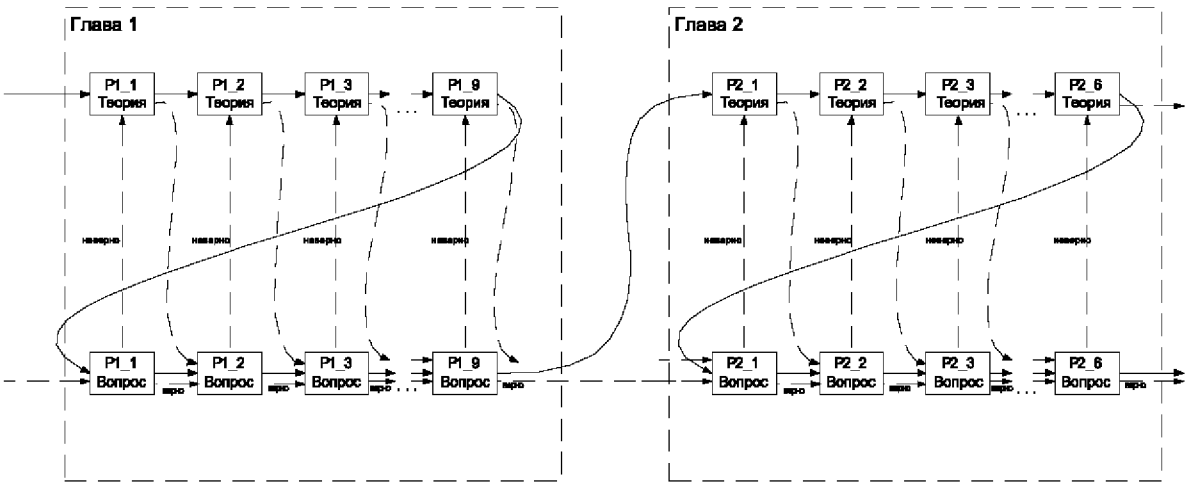


Рис. 3.8. Различные варианты прохождения учебного курса в системе **LPTutor**.

Параллельно ту же самую базу знаний с минимальными доработками можно использовать для реализации обучения группы студентов в дисплейном классе, либо профессиональной системы дистанционного обучения — в этом случае предпочтительной окажется реализация на базе Java-сервлетов, так как она позволит отслеживать успеваемость студентов на сервере, запоминать и учитывать время ответа, не будет предъявлять дополнительных требований к клиентскому программному обеспечению. Архитектура такой системы обучения показана на рис. 3.9.

3.3. Распределенные интеллектуально-информационные системы. Система UroExpert для учета и диагностики больных заболеваниями предстательной железы.

Одним из применений инструментария является использование его как локальной интеллектуальной системы, функционирующей на одной ЭВМ в диалоговом режиме или в составе некоторого программного комплекса. При этом интеллектуальная компонента, реализованная инструментарием, остается независимой от информационной, таким образом позволяя в любой момент расширить базу знаний, либо даже перейти к распределенному накоплению и использованию знаний без модификации информационной компоненты программного комплекса.

По такому принципу автором[5] была реализована интеллектуально-информационная учетно-диагностическая система **UroExpert** по информационной поддержке процесса лечения больных заболеваниями предстательной железы [31, 32, 50], которая в данный момент внедрена в лечебную практику отделения урологии ГКБ им. С.П.Боткина (см. рис. 3.10) и стала предметом диссертации на соискание ученой степени кандидата медицинских наук по специальности 14.00.40 "Урология".

Информационная компонента системы реализована в системе программирования Borland Delphi с использованием таблиц в формате Paradox для хранения данных о паци-

[5]В рамках хоздоговорной НИР на основании экспертных знаний Лукьянова И.В., ведущего специалиста кафедры урологии РМАПО, сотрудника урологического отделения ГКБ им. С.П.Боткина.

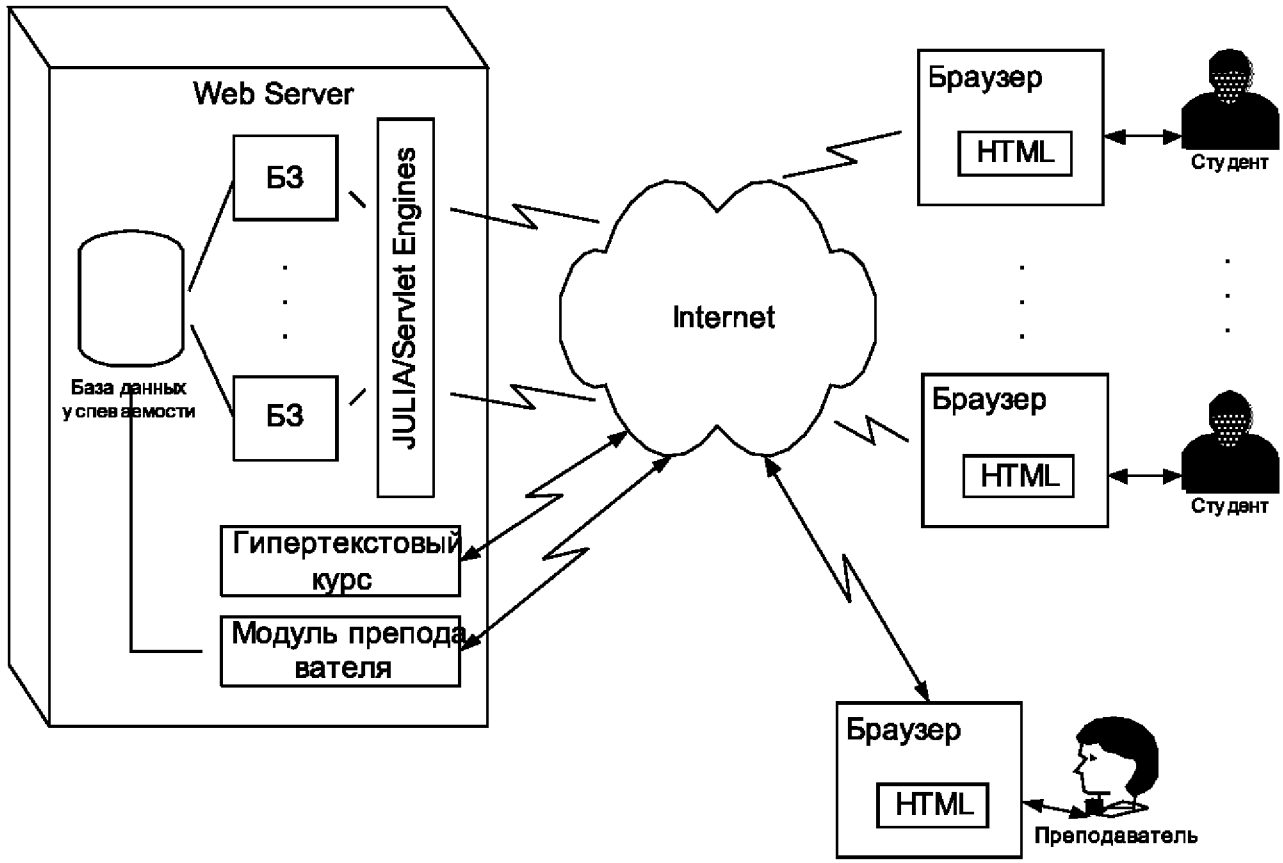


Рис. 3.9. Архитектура комплексной системы дистанционного тестирования и обучения на основе интеллектуальных технологий с явным представлением знаний

сетах. В качестве интеллектальной компоненты используется либо встроенная в систему база знаний на основе высокоуровневой кодогенерации [48], что позволяет легко начать эксплуатацию системы вне сетевой среды без инсталляции JULIA-сервера, либо внешний JULIA-сервер (на рис. 3.11 показана информационная компонента системы, обращающаяся к JULIA-серверу, расположенному на той же ЭВМ) с локальной или распределенной фреймовой иерархией, допускающей модификацию локальной базы знаний, либо использование удаленной базы знаний в рамках электронной коммерции. Для доступа к JULIA из Delphi была разработана DLL, реализованная в среде Microsoft Visual C++ с использованием Visibroker, а также соответствующая библиотека и компонент для Delphi. Несмотря на поддержку CORBA непосредственно последними версиями Delphi Enterprise Edition (5.0 и выше), это позволяет добавить еще один уровень абстракции, который может быть полезен при переходе на другой протокол удаленного взаимодействия (Java RMI, XML-RPC и т.д.).

На базе локальной версии системы **UroExpert** возможно развертывание полномасштабного комплекса диагностики в рамках крупной больницы в соответствии со схемой 3.12. Основные принципы, положенные в основу комплекса, состоят в возможности распределения по сети как баз данных, содержащих информацию о пациентах, так и баз знаний, содержащих онтологические описания методов диагностики и рекомендаций по лечению на различных уровнях абстракции, с учетом особенностей локальных лечебных процессов. Особенность архитектуры распределенной иерархии такова, что данная модель является сравнительно легко масштабируемой на сети больниц и лечебных учреждений,

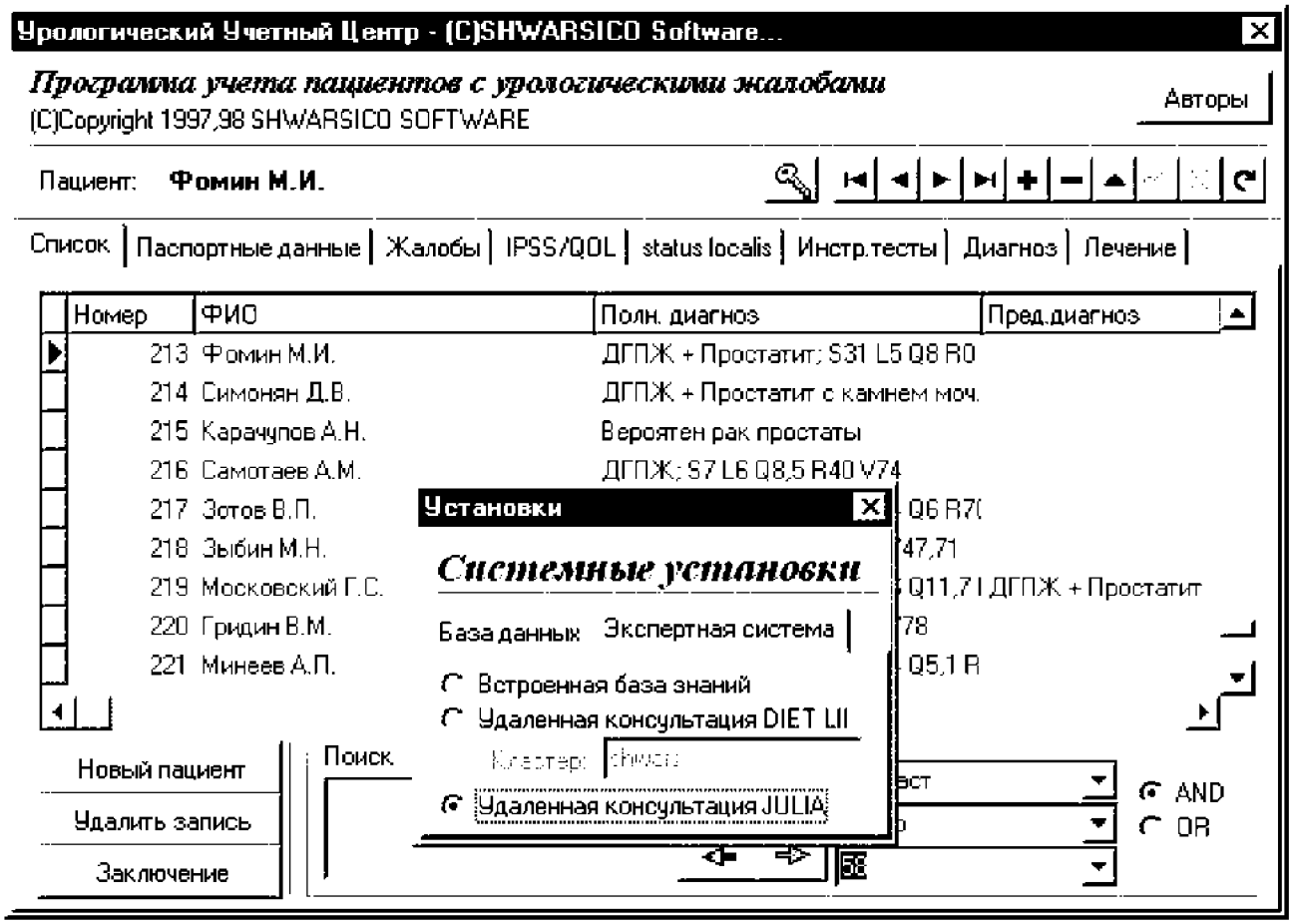


Рис. 3.10. Пользовательский интерфейс системы **UroExpert**

обеспечивая в том числе доступ к централизованным ресурсам баз знаний со стороны мелких региональных больниц через Интернет, в том числе на основе электронной коммерции по принципу подписки на услуги (с оплатой за определенный временной период, за консультацию, за число использованных правил, за число загруженных модулей баз знаний и по другим схемам, см. также раздел 3.4).

Основные принципы, положенные в основу масштабируемой многоуровневой модели диагностики, следующие:

- Наличие в системе **локальной встроенной базы знаний** начального уровня, позволяющей приступить к использованию системы немедленно, без сетевой поддержки. Таким образом возможно построение некоторой информационной инфраструктуры на местах, с последующей интеграцией в единую интеллектуально-информационную сеть.

- Возможность использования **локальных модифицируемых баз знаний** наряду с удаленными серверами, а также возможность наследования от баз знаний удаленных серверов и модификации процесса вывода для его адаптации к локальному лечебному процессу, принятому в данном отделении или лечебном учреждении.

- Возможность использования **централизованного сервера БЗ**, обслуживающего несколько клиентских мест. В этом случае предполагается также использовать централизованную СУБД для хранения данных о пациентах и встроенные в JULIA

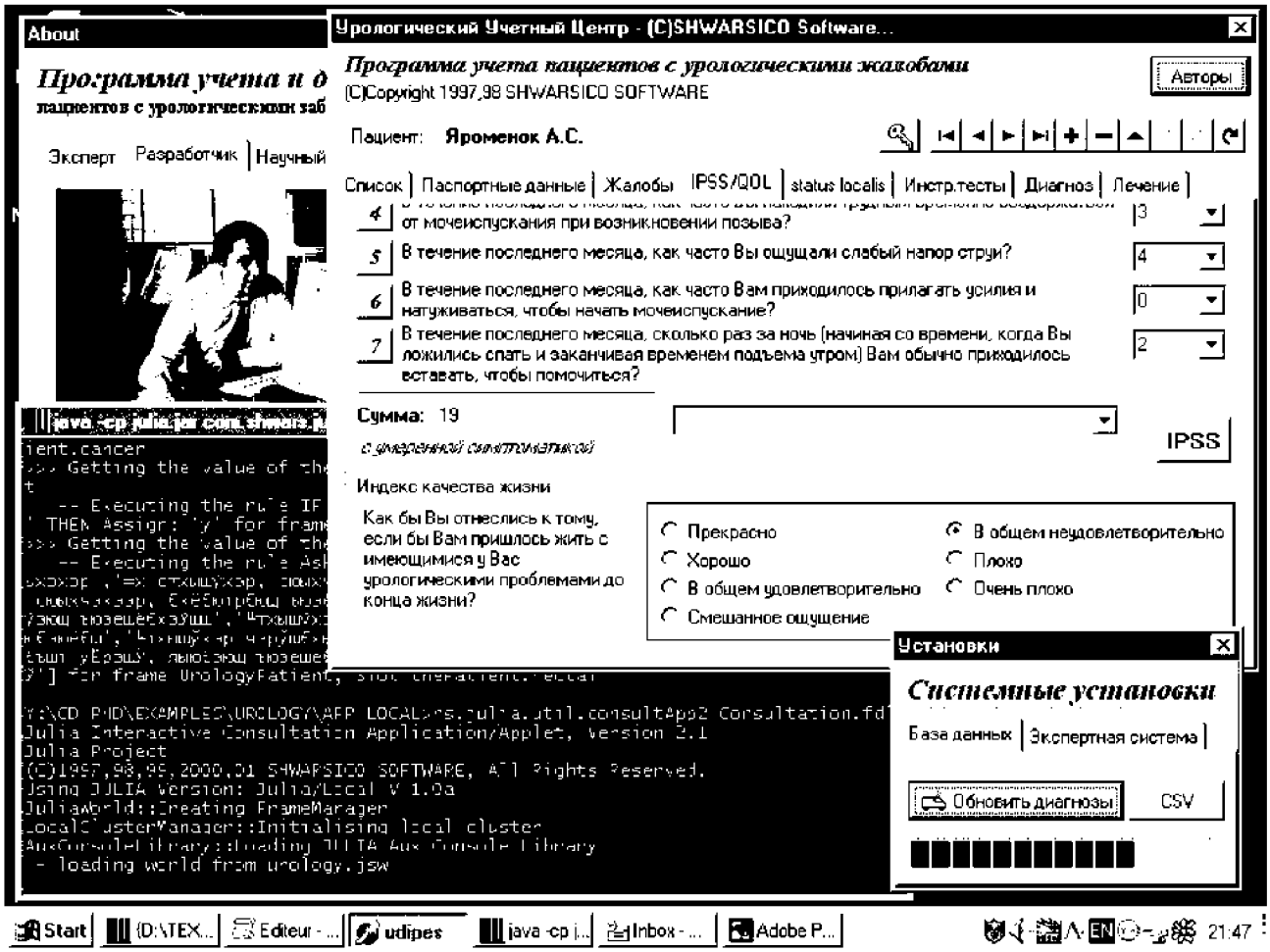


Рис. 3.11. Информационная компонента системы **UroExpert**, работающая совместно с локальным JULIA-сервером (DOS-сессия в левом нижнем углу экрана, кроме того показаны несколько вспомогательных диалоговых окон системы)

возможности по доступу к реляционным БД.

- **Распределенное накопление знаний специалистами в различных областях медицины** (ортогональное), с последующим использованием набора БЗ для полной консультации пациента. Такая модель представляет собой типичный случай баз знаний с непересекающимися доменами ([99], раздел 2.5.3). В этом случае уместно использовать множественное или псевдомножественное наследование для организации модели доски объявлений (раздел 2.5.2). Такая модель подразумевает наличие централизованного сервера, содержащего базу общих знаний (common knowledge) в данной предметной области, набора индивидуальных активных баз знаний или пассивных хранилищ знаний и синхронизатора, управляющего псевдомножественным наследованием (в случае множественного наследования заменяется одним фреймом на центральном сервере).

- **Распределенное накопление знаний специалистами в одной области медицины** (параллельное), с целью сравнения результатов и получения статистики, либо с целью дальнейшего интеллектуального анализа результатов. В этом случае по исходному фрейму-экземпляру пациента создается несколько его копий, которые наследуются от различных распределенных БЗ и подвергаются логическому

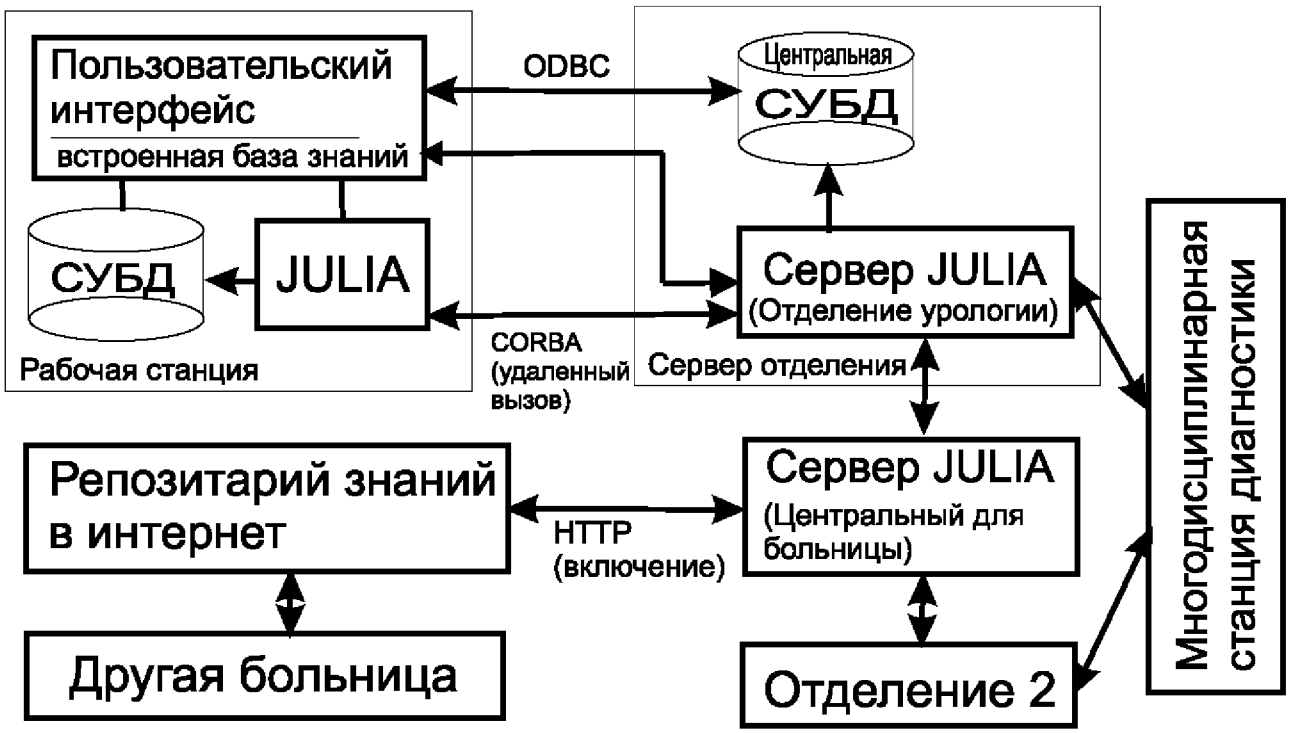


Рис. 3.12. Схема распределенной многодисциплинарной системы медицинской диагностики масштаба больницы

выводу. В дальнейшем результаты могут обрабатываться либо программным кодом с целью статистического сбора информации, либо могут использоваться как исходные данные в некоторой интеллектуальной модели, рассуждающей относительно результатов (выводов) других экспертных модулей.

- Возможность построения **смешанной модели** на основе произвольной комбинации двух предыдущих моделей. Архитектурно удобно представлять такую модель как "последовательно-параллельное" соединение баз знаний.

- Возможность ограниченного доступа к иерархии баз знаний по сети Интернет, как со стороны клиентских приложений и других JULIA-серверов, так и через веб-интерфейс.

Рассмотренная здесь модель позволяет наиболее полно использовать преимущества распределенной фреймовой иерархии. К сожалению, для построения полномасштабной системы необходима не только большая работа по созданию онтологий в различных областях медицины, но и существенная стандартизация на уровне отрасли, что по сути дела является слишком глобальной задачей. Однако за счет способности архитектуры к масштабированию возможно постепенное наращивание системы, начиная с опытных образцов в рамках инфраструктуры одной больницы. Как было отмечено ранее, в настоящее время с участием автора разработано модельное семейство баз знаний в области урологии для диагностики заболеваний предстательной железы, используемое в составе программного комплекса **UroExpert** в лечебном процессе в ГКБ им. С. П. Боткина.

3.4. Распределенное использование знаний с позиций электронного бизнеса и виртуальных корпораций

Обеспечение возможности удаленного *использования* знаний без передачи самих знаний в некотором представлении открывает новые возможности для использования знаний как товара в электронном бизнесе. Во многих случаях может оказаться экономически невыгодным продавать базу знаний как товар: себестоимость создания базы знаний весьма велика, что приводит, как правило, к высокой стоимости конечного продукта, а многим пользователям может быть необходимо всего несколько консультаций, ради которых нецелесообразно приобретать продукт целиком. В таких случаях продавать отдельные консультации может оказаться намного удобнее.

Рассмотренный способ построения распределенных фреймовых иерархий открывает широкие возможности по коммерческому использованию знаний, так как позволяет не только проводить интерактивные консультации с пользователем, но и использовать знания в составе интеллектуальных компонент больших программных комплексов, расширяя коммерческие базы знаний локальными правилами. В частности, рассмотренный подход может успешно применяться для взаимодействия бизнесов владельцев интеллектуальной собственности на базы знаний в рамках электронной коммерции типа business-to-business (B2B). Незначительная доработка инструментария механизмами защищенной аутентификации позволит реализовать гибкие механизмы оплаты за отдельную консультацию, за количество использованных правил и другие.

Другое современное направление, в котором могут эффективно использоваться распределенные базы знаний — это т.н. **виртуальные корпорации** [43], т.е. компании, производящие некоторый, как правило информационный, продукт распределенно. В качестве примера можно привести высокотехнологичные компании по производству программного обеспечения, туристические компании и сети компаний, некоммерческие организации и др.

Для таких компаний оказывается удобным организовать процесс автоматизации принятия решений с использованием распределенной базы знаний, отдельные части которой распределены территориально в соответствии со структурой корпорации (так называемое **территориальное распределение**). Например, если целью консультации является выбор места для туристической поездки в соответствии с различными критериями клиента, то такая консультация может использовать набор территориально-распределенных баз знаний в филиалах для определения степени привлекательности того или иного места для клиента (возможно, с заданием дополнительных уточняющих вопросов). При этом такая консультация опять же может выполняться автоматически в среде Интернет, с целью привлечения клиентов к покупке товаров в рамках электронной коммерции типа business-to-consumer (B2C). С аналогичной ситуацией мы сталкиваемся при выборе товаров в сети электронных магазинов. С точки зрения распределенной фреймовой иерархии такого рода задачи могут эффективно решаться с использованием уже рассмотренных ранее методов распределенного решения задач (раздел 2.5.3) и модели доски объявлений (раздел 2.5.2).

3.5. Интеллектуальная всемирная паутина. Технология активных интеллектуальных страниц IASP.

Следует отметить, что в настоящее время интеллектуальные технологии начинают все шире использоваться в Интернет, и привлечение клиентов к совершению покупок на коммерческих сайтах — только одна из возможных областей такого использования. В качестве примера стоит отметить проект Yandex-Guru (`http://guru.yandex.ru`) — экспертную систему для выбора потребительских товаров в зависимости от потребностей покупателя. Наличие более богатых инструментов для построения интеллектуальных систем в Интернет позволило бы расширить сферу применения технологий ИИ, в том числе для решения таких "проблематичных" задач, как поиск информации, управляемый просмотр и др.

Различные задачи применения интеллектуальных технологий в Интернет можно разделить на две основные группы:

- **Интеллектуальное наполнение**, при котором содержимое веб-страниц представляет собой результат работы интеллектуальной системы (или зависит от этого результата). К этой категории можно отнести рассмотренные ранее удаленные консультации.

- **Интеллектуальная навигация**, т.е. управляемый интеллектуальной системой просмотр (визуальный или автоматический) с определенной целью (поиск информации, демонстрация определенной информации пользователю и др.). Интеллектуальная навигация подразумевает аннотирование страниц некоторыми структурированными знаниями об их содержимом и последующее использование этих знаний для определения наиболее успешного перехода.

В обоих случаях подразумевается, что есть некоторый способ аннотирования страниц знаниями, или комбинирования текста страниц с "интеллектуальным кодом", наподобие того, как технология активных серверных страниц Microsoft (ASP — Active Server Pages [59]) позволяет сочетать HTML-разметку с программным кодом на скриптинговом языке (Visual Basic Scripting Edition, JScript и другие).

Один из подходов состоит в том, чтобы рассматривать веб-страницы как самостоятельные законченные базы знаний, порождающие текст страницы в результате выполнения консультации. Такой подход применительно к логическому программированию развит в [97], где набор связанных веб-страниц рассматривается как взаимодействующие логические программы. Другой подход с аннотированием страниц и рассмотрением набора страниц как реляционной структуры предложен в [69].

В данном разделе (а также в [53, 101]) предлагается такая технология, которую мы назовем IASP (Intelligent Active Server Pages) для продукционно-фреймового представления знаний, реализация которой основана на использовании инструментария JULIA. Описываемая технология базируется на следующих основных принципах:

- ASP-подобный синтаксис, позволяющий внедрять интеллектуальное наполнение в HTML-страницы.

- Взаимодействие IASP-страниц между собой, а также с другими активными (ASP, PHP и др.) и статическими (HTML) страницами на основе стандартного протокола HTTP с помощью HTTP-вызова и включения.

- Возможность вызова IASP-страницы в интерактивном (из веб-браузера) и неинтерактивном (из другой страницы) режимах, а также в диалоговом режиме для поддержки обратного вывода.

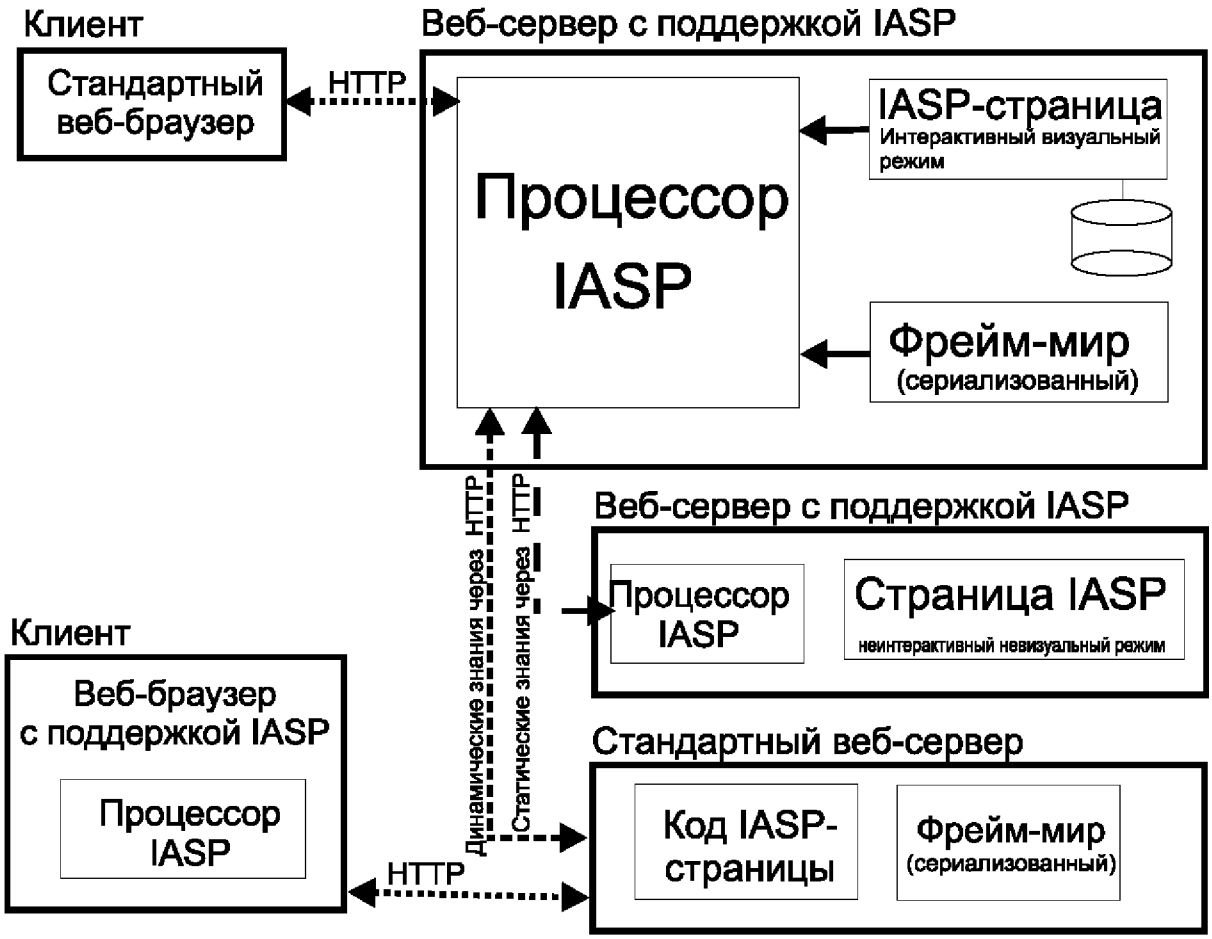


Рис. 3.13. Архитектура приложения, использующего технологию IASP

3.5.1. Формат IASP-страницы и правила трансляции

Каждая IASP-страница может содержать следующие составляющие, которые могут чередоваться в произвольном порядке:

- **Визуальное представление** содержит HTML-код для отображения страницы, который может содержать также произвольные JFMDL-выражения, окруженные скобками <%= и %>.
- **Интеллектуальное наполнение**, т.е. произвольный код на языке представления знаний, окруженный <% и %>.
- **Шаблон диалога**, окруженный <%> и </%>, определяющий HTML-код для страниц подзапросов, используемых в диалоговом режиме в процессе обратного вывода.

Реализация IASP на базе JULIA (см. рис. 3.13) использует IASP-процессор, расположенный на каждом веб-сервере [6]. Когда с сервера запрашивается код, содержащий IASP, процессор производит следующие действия:

- Создает соответствубщий странице фрейм-мир
- Заполняет слоты определенных фреймов этого фрейм-мира в соответствии с параметрами HTTP-запроса (при этом может происходить прямой логический вывод)
- Производит вычисление слота `Response.Body` полученного фрейм-мира и возвращает результат пользователю. В процессе этого как правило происходит обратный или комбинированный логический вывод, а также может осуществляться взаимодействие с другими страницами.

Следующие правила используются для трансляции IASP-страницы во внутреннее представление JULIA:

- Интеллектуальное наполнение интерпретируется как обычный JFMDL-текст и преобразуется по обычным правилам. Конструкция %> ... <% может использоваться для вставки длинных HTML-строк в текст базы знаний, например:

```
IF diagnosis='flu' THEN goal=%>
 <H1>You have got flu!</H1>
 <P><B>Caution:</B> Flu is a kind of
   infection which is easily transmitted....
 </P><% ;
```

- Визуальное представление, содержащее смесь HTML-кода и JFMDL-выражений преобразуется в одно строковое выражение, присваиваемое слоту `Response.Body`. Например, фрагмент

```
<HTML><BODY>
 <H1>The result is <%=MainFrame.Result1 + MainFrame.Result2%>
 </H1>
 </BODY></HTML>
```

эквивалентен следующему тексту на JFMDL:

```
SET Response.Body=
 '<HTML><BODY><H1>The result is '+(MainFrame.Result1+MainFrame.Result2)+
 '</H1></BODY></HTML>'
```

- Шаблон диалога сохраняется в слоте `Runtime.DialogTemplate`.

[6] Возможна также клиентская реализация, использующая браузер с поддержкой IASP, который может быть реализован в виде апплета-надстройки или plug-in к браузеру

3.5.2. Процесс вызова IASP-страницы

После создания фрейм-мира для вызываемой страницы, все переданные ей с HTTP-запросом параметры помещаются в одноименные слоты фрейма `Request`, или (если параметр содержит точку) других фреймов. Параметр может представлять собой обычное строковое значение (например, результат заполнения формы), в этом случае оно будет иметь тип `SCALAR STRING` или `LIST STRING`[7], или же являться внешним XML-представлением в соответствии с синтаксисом JULIA, что позволяет передавать сложные значения. Все невизуальные вызовы, производимые другими страницами в сложном приложении используют такой метод передачи параметров для сохранения типов передаваемых значений.

В дополнение, `Runtime.Method` принимает значение GET или POST в зависимости от использованного метода вызова, `Runtime.Mode` устанавливается в соответствии с режимом вызова (визуальный/невизуальный), а `Runtime.Type` в соответствии с типом вызова (диалоговый/недиалоговый).

Процесс присвоения параметров запроса происходит после того, как фрейм-мир полностью создан, и значения слотов фрейма `Runtime` заполнены. Таким образом становится возможным использовать правила прямого вывода, присоединенные к слотам фрейма `Request`, которые срабатывают в процессе первоначального присваивания параметров запроса.

3.5.3. Тип и режим вызова

Страница IASP может быть запрошена пользователем из веб-браузера (указанием URL или из HTML-формы) или другим IASP-процессором в результате обработки страницы, содержащей функцию запроса `HTTP_Call`. В первом случае (так называемый **визуальный вызов**), соответствующий визуальному образу страницы HTML-код получается в результате вычисления слота `Response.Body` и возвращается пользователю. Во втором случае (**невизуальный вызов**), визуальное представление страницы игнорируется и производится вычисление слота, имя которого передается в качестве зарезервированной переменной запроса `goal`. В этом случае значение обычно возвращается в XML-представлении[8].

В процессе получения значения целевого слота при обратном выводе могут возникнуть дополнительные вопросы к пользователю. Если база знаний содержит предложения `ASK`, то страница может вернуть промежуточный результат вместо окончательного ответа используя шаблон, содержащийся в `Runtime.DialogTemplate`. Когда пользователь отвечает на вопрос, ответ посылается той же самой IASP-странице, и процессор ассоциирует его с уже созданной копией фрейм-мира и продолжает логический вывод. Режим вызова, в процессе которого инициируется дальнейший диалог с пользователем, называется **диалоговым**. В случаях, когда задание дополнительных вопросов нежелательно, а хочется лишь попытаться получить результат на основе имеющихся значений, страница может быть вызвана в **недиалоговом** режиме путем установки значения переменной запро-

[7] В качестве списка представляются одноименные параметры, присутствующие многократно, например в результате выбора нескольких строчек в форме `<SELECT MULTIPLE>`

[8] Для удобства построения веб-приложений, если возвращаемое значение начинается с `<HTML>`, оно возвращается как текст страницы для визуального отображения. Таким образом можно строить страницы, которые могут выполнять различные функции в зависимости от параметра `goal`, возвращая визуальный результат

са **_mode_**, в этом случае все предложения `ASK` заканчиваются неуспехом, инициируя процесс возврата (backtracking).

	диалоговый	**недиалоговый**
визуальный	Диалоговая система, использующая обратный вызов, основанный на ответах пользователя	Вызов с помощью веб-формы, в которой пользователь заполняет начальные значения и получает результат за один вызов.
невизуальный	Практически не используется из-за сложности реализации заменяется интероперабельностью на основе CORBA или включением	Автоматический вызов из других страниц, при котором задаются все начальные значения, и результат получается за один вызов.

Таблица 3.1. Различные комбинации режима и типа вызова страниц IASP

3.5.4. Взаимодействие страниц IASP

Сложное интеллектуальное веб-приложение может быть построено из набора IASP страниц, между которыми осуществляются следующие виды взаимодействия:

- **Вызов**, когда одна страница вызывает другую невизуально посредством функции `&IASP_Call(URL, goal)`, тем самым инициируя логический вывод на другом или том же самом сервере. Таким же образом может быть вызвана любая другая веб-страница (ASP, PHP), возвращающая XML-значение в формате JULIA. Страницы, возвращающие произвольный HTML-текст могут быть вызваны посредством функции `&WebCall`, которая возвращает фрейм-наследник от `WebPage`, содержащий некоторые данные по странице (различные служебные поля, список ссылок и т.д.).

  Диалоговый режим вызова обычно не используется, что исключает реализацию таким образом наследования, аналогичного удаленному статическому наследованию. Таким образом, вызываемая страница всегда ведет себя автономно, не позволяя обмениваться динамическими знаниями за пределами однократного процесса вывода.

- **Включение**, посредством которого IASP-процессор может загрузить находящуюся на удаленном или том же самом сервере страницу и сделать ее доступной в виде фрейм-мира для текущей страницы. В этом случае становится возможным использование полноценного наследования и всей описанной для распределенной фреймовой иерархии функциональности — отличие лишь в формате представления загружаемых страниц.

- **Комплексное взаимодействие** и **доступ к корпоративным базам данных и объектам бизнес-логики**, которые производятся стандартными для JULIA методами.

Важно отметить, что одна и та же страница может участвовать в различных видах взаимодействия, а также вызываться в различных режимах для оптимального использования содержащегося в ней интеллектуального наполнения.

3.5.5. Вопросы реализации технологии IASP

При разработке большой экспертной системы с большим количеством правил и сравнительно малой визуальной составляющей может оказаться неэффективно хранить IASP-страницы в исходной форме, так как при каждом вызове будет производиться их трансляция. IASP-процессор поддерживает также вызов сериализованных фрейм-миров во внутреннем представлении JULIA: если URL указывает на файл с расширением `.jsw`, то соответствующий фрейм-мир загружается и далее его слоты заполняются в соответствии с описанным в разделе 3.5.2 алгоритмом. Сериализованный фрейм-мир может быть получен в результате предварительной трансляции IASP-текста с помощью отдельной утилиты.

IASP-процессор может реализовываться одним из следующих способов:

- Как независимое приложение-сервер, который в этом случае помимо IASP должен уметь возвращать обычные HTML-документы. Именно так реализован демонстрационный прототип, входящий в состав инструментария JULIA[9].
- Как Java-сервлет, используемый совместно с одним из стандартных веб серверов (Apache, Microsoft Internet Information Server). В данном случае возможно использовать технологию URL rewriting для обеспечения удобной системы именования.
- Как plug-in, который вызывается веб-сервером (Apache, IIS) для обработки страниц с определенными расширениями.
- Клиентская реализация, использующая включение страниц IASP и их обработку на клиентском браузере.

Как правило, имеет смысл помимо IASP-сервера поддерживать и обслуживание запросов на основе CORBA к тем же самым базам знаний, задаваемым IASP-страницами. Это обеспечит возможность построения более сложных приложений на основе распределенной фреймовой иерархии, в то же время оставляя возможность вызова баз знаний в интерактивном режиме через всемирную паутину.

3.5.6. Заключение к разделу 3.5

Предложенная технология IASP является в некотором роде аналогом технологии распределенной фреймовой иерархии для построения веб-приложений. Будучи основанной на стандартном HTTP-протоколе, она позволяет комбинировать в рамках одного веб-приложения как интеллектуальные страницы, так и традиционные статические (HTML) и динамические (ASP, PHP) веб-страницы. Однако ограничение взаимодействия протоколом HTTP не позволяет эффективно реализовать удаленное статическое наследование, тем самым снижая привлекательность технологии IASP по сравнению с полноценной распределенной фреймовой иерархией. Однако IASP может использоваться совместно с распределенной иерархией на основе инструментария JULIA для построения веб-интерфейсов к корпоративным базам знаний, хранилищам данных и объектам бизнес-логики.

[9]Демонстрационный прототип реализован под руководством автора студентом МАИ Михановым С.В.

3.6. Интеллектуальный поиск в Интернет на базе онтологического описания. Система JEWEL.

Внедрение в веб-страницы интеллектуального наполнения может также быть положено в основу интеллектуального поиска ресурсов в глобальной сети. В простейшем случае можно представить себе множество определенным образом организованных IASP-страниц, каждая из которых содержит базу знаний для определения ее релевантности семейству структурированных запросов в некоторой предметной области. Однако такая "линейная" архитектура организации поиска (все страницы содержат несвязанные базы знаний одного уровня) представляется недостаточно организованной; кроме того, в ней отсутствуют какие-либо средства структуризации описания ресурса.

Для эффективной структуризации аннотирования веб-ресурсов оказывается очень удобным использовать понятие онтологии и онтологической системы. При таком подходе создается некоторая ориентированная на задачи поиска мета-онтология с набором предметных онтологий более низкого уровня (возможно, также организованных в виде многоуровневой иерархии онтологий), а индивидуальные веб-страницы реализуют онтологии нижнего уровня, в простейшем случае являясь экземплярами уже описанных в онтологиях верхнего уровня концептов. Например, домашняя страничка автора может быть унаследована от понятия "старший преподаватель МАИ", которое, в свою очередь, наследуется от более общего понятия "преподаватель", "человек" и т.д. Онтологии верхнего уровня определяют основные атрибуты (имя, размер заработной платы, звание, степень и др.), в то время как индивидуальная страница лишь специфицирует их значения (возможно, с использованием ссылок на другие онтологии, например, определенные на сайте МАИ).

3.6.1. Краткий обзор существующих систем онтологического поиска

В настоящее время в области онтологического поиска ведутся активные исследования, среди которых следует отметить проект SHOE [14, 89], разрабатываемый кафедрой информатики университета Мериленда, а также инициативу $(KA)^2$ [14, 124] (сокращение от Knowledge Annotation Initiative of the Knowledge Acquisition Community), целью которого является организация интеллектуального поиска в Интернет и автоматизация накопления новых знаний. В этой инициативе принято выделять следующие направления:

- Аннотация web страниц интеллектуальной информацией;
- Онтологический инжиниринг
- Организация интерфейса запросов и вывода по распределенной онтологии

Аннотация Web страниц осуществляется за счет расширения HTML специальным тегом <ONTO>, в рамках которого можно задавать онтологии для спецификации WWW страниц. Процесс дополнения Web страниц такой онтологической информацией рассматривается в рамках направления онтологического инжиниринга. В онтологический инжиниринг входит разработка онтологической системы на основе инструментария Ontolingua [76]. На настоящий момент разработано более десятка онтологий для описания организаций, направлений исследований, рабочих процессов, личностей, продуктов производства и пр. Для поиска в рамках $(KA)^2$ предполагается использовать подсистему **Ontocrawler**,

а для организации запросов и вывода — программный продукт **Ontobroker** со своим внутренним языком запросов.

Следует также упомянуть о направлении развития интеллектуального поиска, связанном с автоматическим извлечением онтологического описания страницы из ее исходного текста. В частности, такого рода подход, основанный на анализе синтаксической структуры HTML-документа, предлагается в [33, 84]. Привлекательность анализа исходного текста HTML в неизменном неаннотированном виде очевидна, однако такой анализ относится к классу задач интерпретации естественного языка, которые в наиболее общем виде пока не решены[10]. Однако отмечается [14], что анализ HTML-документов может быть построен на совместном анализе естественно-языкового текста и тегов разметки, поскольку последние обычно несут некоторую эвристическую семантическую нагрузку, которая может использоваться для структурирования текста документа в виде фреймовой сети, затем применяемой для последующего извлечения знаний и вывода.

3.6.2. Основные положения онтологического описания

Общим для всех систем онтологического аннотирования является то, что в качестве аннотации веб-ресурса выступает специальным образом организованная предметная онтология, которая содержит структурированные знания об аннотированном ресурсе относительно некоторой метаонтологии предметной области. Можно предложить различные способы размещения онтологической информации о ресурсе: включить онтологическое описание в HTML код через введение новых HTML тегов, либо хранить онтологическое описание ресурса в отдельном файле в каком либо специальном представлении. Основная задача онтологического подхода состоит в том, чтобы облегчить пользователю поиск информации в большом наборе ресурсов за счет систематизации знаний, создания единой иерархии понятий, унификации терминов и правил интерпретации.

Для описания онтологий можно использовать различные языки представления знаний, в том числе и продукционно-фреймовое представление. При этом, как показано в [46], оказывается удобным делать различие между фреймами-прототипами, описывающими абстрактные **категории** предметной области, и фреймами-экземплярами, которые описывают конкретные **концепты**. Если сосредоточить фактические описания явлений и закономерностей — то есть категорий предметной области — в нескольких онтологиях страниц, то появляется возможность искать нужную информацию во множестве страниц посредством поиска онтологий, концепты которых соответствуют указанным в поисковом запросе условиям. Условия запроса могут касаться как отношений наследования между категориями или отношений представления между категориями и их концептами, так и условий, накладываемых на значения слотов для концептов известных категорий. За счет явного разделения категорий и концептов имеется гарантия, что наследники не претерпели структурных или поведенческих изменений, так как концепт нельзя дополнить новыми слотами или продукциями.

Таким образом, поиск информации разделяется на два этапа: вначале выбираются релевантные запросу описания явлений, а затем ведется поиск частных случаев этих явлений. Это обстоятельство, при условии уникальности используемых имен, дает дополнительное преимущество принудительной унификация понятий в рамках одной предметной

[10]Целевой задачей естественно-языкового анализа является задача построения системы, способной понимать произвольный естественно-языковый текст и отвечать на вопросы относительно содержащихся в нем знаний [88].

области, что исключает возможность двусмысленности поискового запроса.

Рассмотренные принципы легли в основу реализации на базе JULIA системы онтологического поиска **Jewel**, разработанной под руководством автора студентом-дипломником МАИ Сизиковым Е.В.

3.6.3. Язык онтологического описания

Для внедрения онтологического описания в тело HTML-страницы предлагается использовать надмножество стандарта HTML, в котором расширяется стандартный тег <SCRIPT>, а также вводятся новые теги <USE>, <CONCEPT>, <SET> и <ASSIGN>. В качестве языка описания онтологий используется некоторая модификация JFMDL, расширенная введением явного разделения фреймов на категории (вводимых предложением CATEGORY) и концепты (CONCEPT).

Ниже приводится пример фрагмента описания предметной онтологии для аннотирования энциклопедии авиационной техники:

Файл: aircraft.html

```
<HTML>
. . .

<SCRIPT language = ONTODEF>

CATEGORY Firm
{
  SCALAR name;
  SCALAR country;
}

CONCEPT Ilushin IMPLEMENTS Firm;
SET Ilushin.name  = 'Il';
SET Ilushin.country = 'Russia';

CONCEPT Tupolev IMPLEMENTS Firm;
SET Tupolev.name  = 'Tu';
SET Tupolev.country = 'Russia';

CATEGORY Plane
{
  SCALAR name DEF 'Plane';  // Название самолета
  LIST modifications DEF [];  // Список возможных модификаций
  REF firm; // Указатель на концепт,описывающий производителя самолета
  SCALAR type; // Тип самолета
  SCALAR speed; //Скорость самолета
}

IF Plane.speed<=1250 THEN Plane.type ='subsonic';
IF Plane.speed>1250 THEN Plane.type ='supersonic';
SET Plane.type = 'speed is unknown';

CATEGORY PassengerPlane EXTENDS Plane
{
  SCALAR passengers; // Число пассажиров
}

CATEGORY TransportPlane EXTENDS Plane
{
  SCALAR mass; // Масса полезной нагрузки
}

</SCRIPT>

. . .
```

```
</HTML>
```

Конкретные странички различных моделей самолетов определяют в себе экземпляры категорий из этой онтологии следующим образом:
Файл: `tu-154.html`

```
<HTML>
...
<USE 'aircrafts.html' AS aircraft >

<CONCEPT tu154 IMPLEMENTS@aircraft~PassengerPlane>

<ASSIGN tu154.name> Tu-154 </ASSIGN>

<SET tu154. firm = @Tupolev>
<SET tu154.speed = 900>
<SET tu154.modifications = $['Tu-154A', 'Tu-154M']>
<SET tu154.passengers = 100>
...
</BODY></HTML>
```

3.6.4. Язык поисковых запросов

Для составления поисковых запросов в системе JEWEL предлагается использовать специализированный язык, напоминающий по синтаксису предложение SELECT языка SQL. Основным оператором поиска является оператор SEARCH, который имеет следующую форму:

```
SEARCH
USE 'адрес_1' AS имя_1
. . .
USE 'адрес_N' AS имя_N
IMPORT LIBRARY имя_библиотеки_1
. . .
IMPORT  LIBRARY имя_библиотеки_M
WHERE ''условие''
```

Под условием понимается произвольное логическое выражение в синтаксисе JFMDL, определяющее критерии поиска. В процессе поиска производится обход всех подходящих запросу (т.е. содержащих искомые концепты) онтологий, и к элементам каждой из них применяется указанное поисковое условие. В качестве результата возвращаются онтологии, для которых условие оказывается истинным.

Для составе условия могут использоваться следующие предикаты:

- **INHERITED**(*имя категории*) принимает истинное значение в текущей онтологии, если имеется категория, унаследованная непосредственно от указанной в аргументе. В противном случае предикат принимает ложное значение.
- **EXTENDS**(*имя категории*) принимает истинное значение в текущей онтологии, если имеется категория, унаследованная (возможно транзитивно) от указанной в аргументе. В противном случае предикат принимает ложное значение.
- **IMPLEMENTS**(*имя категории*) принимает истинное значение в текущей онтологии, если имеется концепт, представленный категорией, указанной в аргументе. В противном случае предикат принимает ложное значение.

Кроме того, в язык поисковых запросов включен набор операторов для определения имеющихся в индексном файле онтологий, для добавления или удаления онтологий и т.д. Подробнее синтаксис языка запросов изложен в [46].

Приведенный ниже пример поискового запроса

```
SEARCH
USE ' aircrafts.html' AS aircraft
WHERE (@aircraft~Plane.type == 'subsonic')
AND (@aircraft~PassengerPlane.passengers == 100)
```

показывает элемент интеллектуальности проводимого поиска, так как информация о том, что самолет Ту-154 является дозвуковым, явно нигде не указывалась, а была выведена логически по продукционному правилу, общему для всех концептов, прямо или косвенно унаследованных от категории Plane.

3.6.5. Архитектура поискового комплекса и вопросы реализации

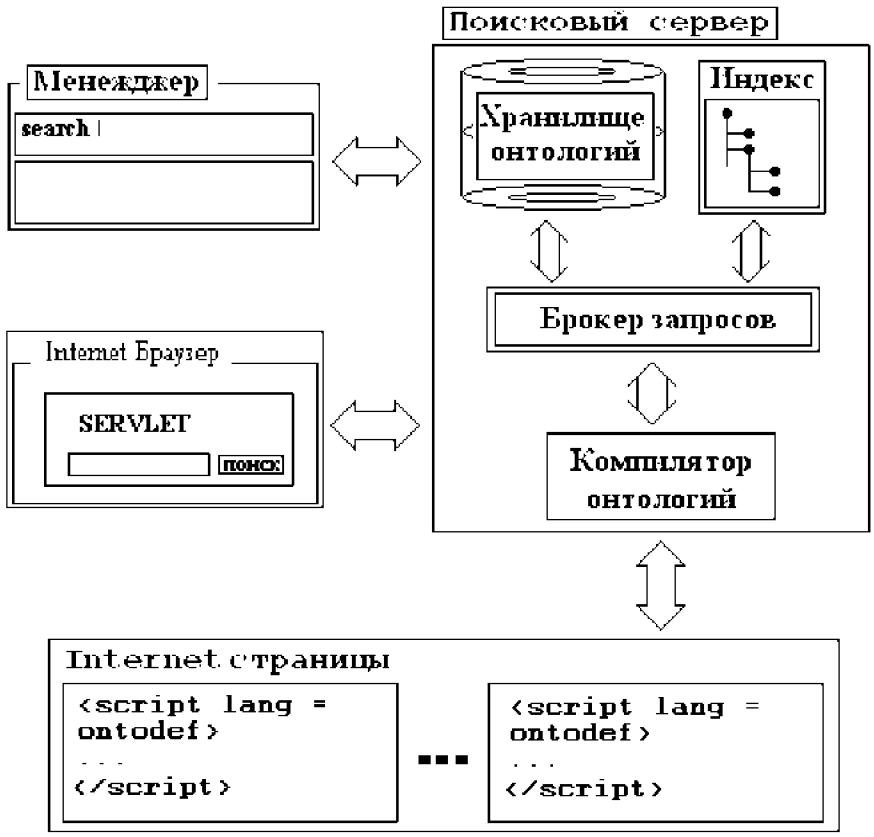


Рис. 3.14. Основные компоненты системы JEWEL и их взаимодействие.

Архитектура поискового комплекса **JEWEL** приведена на рис. 3.14. Основными компонентами комплекса являются:

- **Компилятор онтологических описаний**, вызываемый вручную через утилиту-менеджер, либо автономным агентом-роботом, собирающим онтологические описания в определенном сегменте сети. Онтологические описания транслируются во

внутреннее представление, которое затем сохраняется в так называемой **индексной базе знаний**, которая представляет собой семейство индексных файлов или множество объектов в объектной базе данных. Таким образом, множество известных системе онтологий проиндексировано и сохраняется на поисковом сервере в виде множества фрейм-миров (между фрейм-мирами и онтологиями имеется взаимооднозначное соответствие).

- **Брокер поисковых запросов**, отвечающий за реализацию собственно процесса разбора исходного текста запроса и организацию поиска.

- **Компоненты пользовательского интерфейса**, представленные утилитой-**менеджером**, Java-сервлетом для осуществления поиска и т.д. В дальнейшем планируется расширить систему **браузером онтологий**, **системой интерактивной аннотации ресурсов**, а также более удобным поисковым интерфейсом, предлагающим выбор из набора описанных в онтологиях атрибутов.

Процесс поиска в системе состоит из двух основных этапов:

- **Фильтрации**, в результате которой из всего множества онтологий выбираются те, которые так или иначе используются в поисковом запросе (т.н. **кандидатное множество**). Сюда попадают онтологии, в которых определяется или уточняется один или несколько из фигурирующих в запросе концептов или категорий. Такой выбор осуществляется на основе специальным образом построенного индексного файла, связывающего онтологии с определяемыми в них категориями и концептами с учетом наследования.

- **Логического вывода** в распределенной фреймовой иерархии, образованной всеми входящими в кандидатное множество онтологиями. Соответствующие всем онтологиям кандидатного множества фрейм-миры загружаются в виртуальную машину Java, образуя некоторую локально-распределенную фреймовую иерархию. На этой иерархии для каждого кандидатного фрейма определяется служебный слот, содержащий определенным образом преобразованное выражение `WHERE` поискового запроса. Вычисляя значение этого выражения на всех кандидатных фреймах мы таким образом получаем множество целевых (удовлетворяющих запросу) фреймов, и возвращаем в качестве результата запроса ссылки на веб-страницы, в которых описаны онтологии, содержащие целевые фреймы.

3.6.6. Заключение к разделу 3.6

Разработанная система **Jewel** безусловно пока не готова к полномасштабному применению в Интернет. В первую очередь это связано с необходимостью хранения индексной базы знаний на центральном узле поисковой системы, что обусловлено использованием включения для логического вывода в распределенной фреймовой иерархии, которая на самом деле сосредоточена на одной ЭВМ. Использование активных механизмов логического вывода на промежуточных узлах сети и механизма удаленного вызова позволило бы распределить процесс интеллектуального поиска по всему пространству сети.

Однако основной преградой к полномасштабному внедрению онтологического поиска на основе эксплицитного аннотирования остается отсутствие стандартов, а также

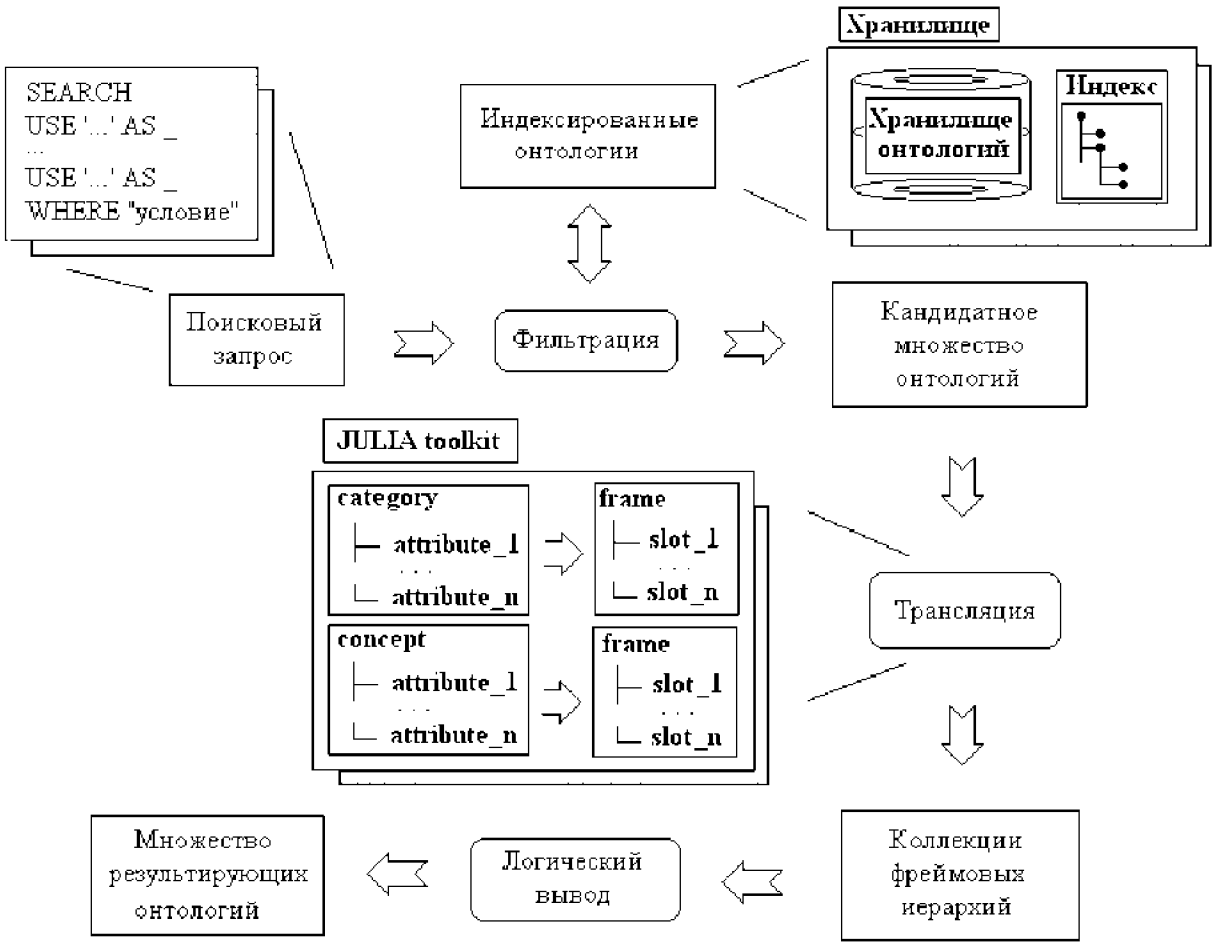


Рис. 3.15. Процесс обработки поискового запроса в системе **JEWEL**.

некоторая сложность составления описаний ресурсов. Есть надежда, что с повсеместным распространением Semantic Web [64], основанной именно на иерархическом аннотировании ресурсов на XML-подобном языке, простор для применения интеллектуальных технологий в Интернет возрастет, в том числе в направлении интеллектуального поиска.

3.7. Применение JULIA для автоматизации производственных систем. Система управления процессом рафинирования и дезодорирования растительных масел OILMASTER.

Одним из аспектов применения распределенных интеллектуальных систем является интеллектуальное управление территориально-распределенным производственным процессом и диагностика оборудования [24]. В настоящее время для этой цели весьма успешно пытаются применять многоагентные архитектуры (см., например, [107]), которые позволяют эффективно справиться с территориальным распределением узлов системы, обеспечивая такие полезные функции, как динамическая конфигурация агентов и интеллектуальная оптимизация процесса производства на основе имеющихся производственных мощностей.

На основе инструменария JULIA была сделана попытка построения автоматизированной системы управления процессом рафинирования и дезодорирования растительных

масел. В результате был создан демонстрационный прототип системы OILMASTER[11], содержащей в себе базу знаний, которая по текущему состоянию датчиков производственной линии вырабатывает сигнал на управляющие устройства для выполнения следующего цикла производственного процесса, либо сигнализирует о неисправности на линии производства. Структурная схема системы изображена на рис. 3.16.

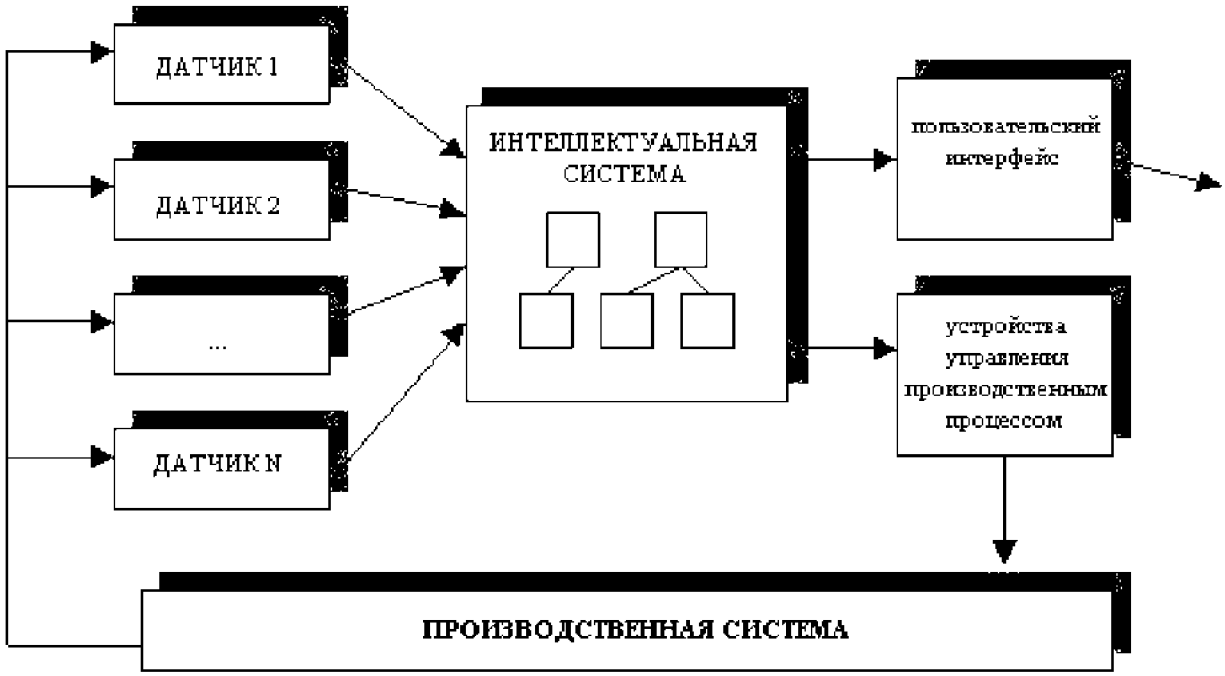


Рис. 3.16. Структура интеллектуальной системы управления производственным процессом OILMASTER.

Внедрение системы такого масштаба на реальном предприятии — задача чрезвычайно трудоемкая, поэтому для демонстрационного прототипа был разработан набор симуляторов реальных устройств, позволяющих отрабатывать поведение базы знаний на программной модели производственной линии. Симуляторы, а впоследствии и реальные датчики, подключаются к фреймовой модели в виде Java-фреймов, которые в процессе запроса значения слотов могут получать данные о состоянии устройства, а при изменении состояния могут инициировать прямой вывод. База знаний содержит знания экспертов по производственному процессу в целом и по возможным неисправностям в производственном комплексе, и совместно с набором симуляторов и графическим интерфейсом (см.рис.3.17) представляет собой чрезвычайно мощную площадку для отработки нововведений в производственный процесс. Кроме того, в процессе эксплуатации модели возможно совершенствование диагностической и управляющей баз знаний с целью дальнейшего внедрения в реальный процесс производства. Если управляющая база знаний представляет собой чрезвычайно ответственную составляющую, трудную для внедрения, то диагностическая составляющая может использоваться уже сейчас, в том числе в диалоговом режиме.

Следует отметить, что традиционный монотонный характер вывода, используемый

[11]Система была создана студентом-дипломником Дубовиком С.Е. под руководством автора при участии сотрудников ООО "Сид-Ойл" в качестве экспертов

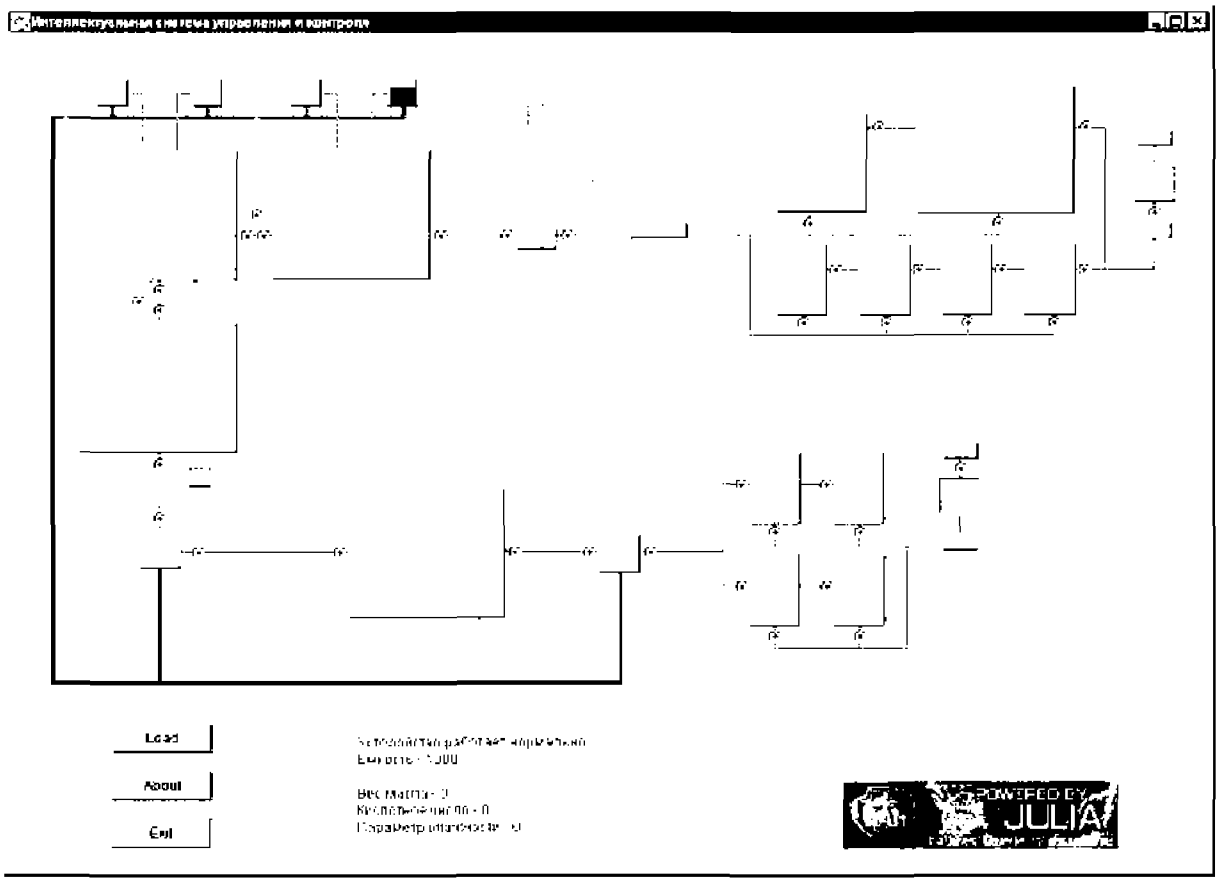


Рис. 3.17. Пользовательский интерфейс системы **OilMaster**.

в JULIA, не очень подходит для реализации интеллектуальных систем реального времени. В **OilMaster** используется циклический алгоритм работы экспертной системы: на каждом шаге цикла производится опрос драйверов датчиков и формирование заключения о работоспособности системы и о следующем необходимом действии. Таким образом, система работает в дискретном времени, привязанном к непрерывному через равномерные интервалы.

Демонстрационный прототип системы **OilMaster** с симуляторами устройств в настоящее время внедрен в опытную эксплуатацию в ООО "Сид-Ойл", о чем в приложении С имеется соответствующий акт о внедрении.

3.8. Использование инструментария JULIA в учебном процессе

Инструментарий JULIA может эффективно использоваться в учебном процессе не только в составе обучающих систем, но и как учебное средство разработки в лабораторных и курсовых работах по курсам, связанным с построением интеллектуальных систем с явным представлением знаний. Основным преимуществом в данном случае является масштабируемость инструментария: в простейшем случае он может использоваться как оболочка для построения диалоговых экспертных систем с классическим продукционным представлением знаний[12], затем простейший синтаксис правил может быть расширен вве-

[12] Особенность языка представления знаний такова, что возможно использование во входном файле только продукционных правил, в этом случае они все будут привязаны к родительскому фрейму `object`, скрывая тем самым от инженера по знаниям фреймовую составляющую представления знаний

дением фреймовой составляющей в базу знаний. На примере JULIA удобно демонстрировать внедрение интеллектуальной составляющей в состав программных систем, интерфейс базы знаний с корпоративной бизнес-логикой на основе JavaBeans, рассуждения по данным из реляционных баз данных, а также аспекты построения распределенных мобильных и статических систем. Так как JULIA является открытым инструментарием, то на его примере возможно также изучение основных принципов построения интеллектуальных систем, с практической реализацией отдельных составляющих (например, стратегии выбора правил, библиотеки функций и т.д.) учащимися.

Таким образом, взяв инструментарий JULIA как практическую основу, можно весьма эффективно и на современном уровне построить курс, знакомящий слушателя с различными аспектами и тенденциями развития интеллектуальных систем. Синтаксис JULIA и семантика логического вывода относительно прозрачны и просты в освоении, позволяя двигаться от простейшей продукционной модели к более сложным стратегиям вывода, таким образом делая его использование доступным для специалистов-экспертов, а не только для инженеров по знаниям (такого достоинства лишены например системы семейства OPS5, для изучения которых требуется известная культура и начальные знания).

По рассмотренной выше схеме автором был поставлен курс "интеллектуальные и экспертные системы", читаемый в настоящее время на вечернем факультете студентам специальности "прикладная математика и информатика". По курсу предлагается комплекс заданий, включающий в себя создание прототипа экспертной системы и его применение как для диалоговых консультаций, так и в составе программных комплексов, в том числе распределенных. Внедрение инструментария в учебный процесс факультета "Прикладная математика и физика" МАИ отмечено актом о внедрении, приведенном в приложении С.

3.9. Выводы к главе 3

В данной главе был рассмотрен целый спектр возможных применений инструментария JULIA как самостоятельной оболочки для построения экспертных систем, работающих в диалоговом режиме и в составе информационных систем, а также как средства построения распределенных и псевдо-распределенных интеллектуальных систем. За короткий срок существования инструментария сложно обеспечить его полноценное внедрение в реальный производственный процесс, особенно учитывая большую трудоемкость создания экспертных систем для реальных задач. Тем не менее, рассмотренный спектр примеров демонстрирует удобство применения инструментария как с точки зрения используемой модели представления знаний и логического вывода, так и для внедрения интеллектуальной составляющей в состав программных комплексов.

Следует отметить, что во многих случаях полноценному использованию распределенной функциональности JULIA мешает слабое развитие сетевой инфраструктуры и необходимая масштабность внедрения. Для эффективного внедрения распределенных интеллектуальных систем в масштабе крупного предприятия или отрасли (это актуально как для рассмотренного примера медицинской диагностической системы, так и для производственных систем) необходима существенная стандартизация, подкрепленная законодательно. Хотя инструментарий и обеспечивает эффективную технологию построения распределенных хранилищ знаний, пройдет еще немало времени пока эти технологии получат широкое распространение на практике.

Заключение

Предложенная архитектура построения распределенных интеллектуальных систем на базе продукционно-фреймового представления знаний, реализованная в инструментарии JULIA, обладает следующими преимуществами:

- Фреймовое представление знаний предоставляет естественный способ кластеризации знаний, в особенности динамических правил прямого и обратного вывода, вокруг соответствующих фреймов в виде процедур-демонов и процедур-запросов, что в свою очередь обеспечивает естественное распределение знаний между различными узлами. Представление знаний в виде иерархических структур находит отражение в виде моделей онтологий [14, 76], и модель распределенной фреймовой иерархии представляет собой один из способов реализации таксономических онтологий, совмещенных с традиционной моделью логического вывода и представления знаний. Методология создания распределенных баз знаний на основе распределенной фреймовой модели заслуживает более подробного изучения, в частности, с позиций адаптации существующих объектных методологий проектирования программных систем.

- Фреймовое представление знаний очень похоже на традиционные в настоящее время объектный и компонентный подходы, что позволяет использовать фреймовую модель как некоторый "общий знаменатель" при создании гибридных систем, сочетающие декларативные знания и императивные компоненты. Таким образом объекты и компоненты открытых систем в форме Java-классов, (Enterprise) JavaBeans, CORBA- и COM-объектов могут быть представлены как фреймы в единой иерархии, и наоборот, любой фрейм иерархии может использоваться как объект, вызываемый из внешней программной системы.

- Фреймы могут эффективно использоваться для доступа к реляционным базам данных, а также к другим типам структурированной информации (например, для анализа сетей веб-страниц [14] и др.).

Таким образом, в инструментарии JULIA реализован целый спектр подходов к созданию гибридных распределенных интеллектуальных систем, что позволяет эффективно использовать его в различных приложениях, ориентированных на распределенное накопление и использование знаний. Безусловно, инструментарий может эффективно использоваться и для создания локальных (функционирующих на одной ЭВМ) экспертных систем, распределенных систем масштаба предприятия, функционирующих на единой шине CORBA, распределенных систем информационно-интеллектуальной поддержки виртуальных корпораций, гетерогенных Интернет-приложений и др.

Разработанный для язык представления знаний JFMDL обладает богатыми декларативными свойствами и является более наглядным (хотя, возможно, за счет некоторых потерь в выразительности), чем языки семейства OPS5, используемые в CLIPS и JESS.

Это позволяет эффективно применять разработанный инструментарий не только для решения реальных задач разработки интеллектуальных систем, но и как основной инструмент в процессе обучения.

В то время как агентный подход в настоящее время находится в стадии бурного развития, нам кажется, что альтернативный подход, основанный на классическом логическом выводе и явном представлении знаний, будет дополнять агентные системы, в ряде случаев оказываясь более удобным для применения. Кроме того, создание расширяемой гибридной многоплатформенной библиотеки для построения интеллектуальных систем, основанной на открытых стандартах (XML, CORBA, HTTP, KQML), само по себе является весьма полезной задачей, находящей множество применений на практике.

На защиту выносятся следующие результаты диссертационной работы:

- Архитектура построения распределенных интеллектуальных систем на основе распределенной фреймовой иерархии с множественным (псевдомножественным) статическим и мобильным наследованием, интеграцией реляционных баз данных и компонентных моделей.

- Математическая модель локального и распределенного вывода на основе продукционно-фреймового представления знаний.

- Архитектура и оригинальное программное обеспечение открытого инструментария JULIA для создания распределенных интеллектуальных систем с продукционно-фреймовым представлением знаний.

- Механизм внедрения знаний в HTML-страницы на основе разработанного языка представления знаний, технология взаимодействия интеллектуальных IASP-страниц в архитектурах тонкого клиента, толстого клиента и смешанных архитектурах.

В работе также присутствуют следующие новые результаты:

- Язык представления знаний JFMDL, отличающийся высокой наглядностью и прозрачностью.

- Архитектура и программное обеспечение интеллектуальной поисковой системы **Jewel**, основанной на явном внедрении онтологического описания в HTML-страницы и поддержании на поисковом узле индексной базы знаний.

На основе инструментария было разработано оригинальное программно-информационное обеспечение ряда интеллектуальных систем:

- комплекса диагностики больных заболеваниями предстательной железы **UroExpert**, в рамках которого поддерживается иерархическое представление медицинских диагностических знаний в распределенной компьютерной сети больницы или семейства медицинских учреждений;

- системы дистанционного обучения логическому программированию в среде Интернет **LPTutor**, основанной на технологии толстого клиента с использованием загрузки правил по мере необходимости и включающей в себя средства тестирования и интеллектуальной навигации по гипертекстовому учебному курсу;

- системы адаптивного интеллектуального тестирования в области Интернет-технологий **IntelliTest**;

- прототипа системы диагностики и управления процессом рафинирования и дезодорирования растительных масел, используемой для анализа работы линии очистки растительного масла в ООО "Сид-Ойл".

Дальнейшее развитие диссертационных исследований возможно по следующим направлениям:

- Методологии проектирования распределенных интеллектуальных и онтологических систем, включая графический язык и систему обозначений для эффективной наглядной работы с распределенными базами знаний. В качестве основы методологии можно рассматривать объекто-ориентированные и компонентные методологии проектирования программных систем, учитывая в них особенности построения систем, основанных на знаниях, а не на алгоритмических способах взаимодействия, а также существующие методологии и графические технологии искусственного интеллекта: деревья вывода, семантические сети и т.д. На основе введенной системы обозначений будет чрезвычайно полезным создание полноценной графической среды разработчика распределенных баз знаний.

- Математические методы оценки эффективности распределенного вывода и оптимизации баз знаний для устранения "узких мест" при таком выводе.

- Интеграция других форм представления знаний в единую распределенную модель, включая представления, основанные на неявном формировании знаний: нейронные сети, нечеткие когнитивные графы (fuzzy cognitive maps) и другие.

Литература

1. Алексеев П.В., Панин А.В. Философия. Учебник. М.: "Проспект", 1999.
2. Аристотель. Метафизика. Российская библиотека образования. Серия "Философия". М., 1998.
3. Бадд Т. Объектно-ориентированное программирование в действии. СПб.: Питер, 1997.
4. Бауэр Ф., Гооз Т. Информатика. М.: Мир, 1976, 1990.
5. Барендрегт Х. Ламбда-исчисление. Его синтаксис и семантика. М.: Мир, 1985.
6. Братко И. Программирование на языке Пролог для искусственного интеллекта. М.: Мир, 1990.
7. Брой М. Информатика. Основополагающее введение. М.: Диалог-МИФИ, 1996.
8. Брюхов Д.О., Задорожный В.И., Калиниченко Л.А., Курошев М.Ю., Шумилов С.С. Интероперабельные информационные системы: архитектуры и технологии. // СУБД, №4, 1995.
9. Булос Дж., Джеффри Д. Вычислимость и логика. М.: Мир, 1990.
10. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С—+., 2-е изд. М.: "Издательство Бином", СПб: "Невский диалект", 1998.
11. Верещагин Н.К., Шень А. Лекции по математической логике и теории алгоритмов. Часть 3. Вычислимые функции. М.: МЦНМО, 1999.
12. Вольфенгаген В.Э. Конструкции языков программирования. Приемы описания. М.: АО "Центр ЮрИнфоР", 2001.
13. Воробьев В.В. Представление, организация и обработка знаний в интеллектуальной системе ИКАР. Дисс...к.ф.-м.н. М.: МАИ, 1992.
14. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. СПб.: Питер, 2000.
15. Гайсарян С.С., Зайцев В.Е. Информатика. Учебный курс. М.: МАИ, 1993.
16. Гретцер Г. Общая теория решеток. М.: Мир, 1982.
17. Громов Г.Р. Очерки информационной технологии. М.: Инфоарт, 1993.
18. Донаху Д. Взаимодополняющие определения семантики языка программирования. // В сб. Семантика языков программирования. М.: Мир, 1980.
19. Джексон П. Введение в экспертные системы. М.: Изд. дом "Вильямс", 2001.

20. Дунаев С. Intranet-технологии. М.: Диалог-МИФИ, 1997.

21. Журавлева Т.Э. Гибридный инструментарий интеллектуальных систем на основе расширенного логического программирования. Дисс ... к.ф.-м.н. М.: МАИ, 1993.

22. Зайцев В.Е., Лукашевич С.Ю. Инструментальные средства для построения встроенных экспертных систем // Информатика, №3-4, 1991. стр. 30-40.

23. Зайцев В.Е., Исаев В.К., Лукашевич С.Ю., Хмелев А.К. Опыт интеграции интеллектуальной компоненты в систему автоматизированного проектирования // Информатика, №3-4, 1991. стр. 74-80.

24. Искусственный интеллект: применение в интегрированных производственных системах / Под ред. Э.Кьюсиака. М.: Машиностроение, 1991.

25. Искусственный интеллект. Справочное издание в 3 кн., Т.2. М.: 1990.

26. Клоксин У., Меллиш К. Программирование на языке Пролог: Пер. с англ. М.: Мир, 1987.

27. Крастелева И.Е., Сошников Д.В. Исследование процесса продвижения интернет-ресурсов с использованием интеллектуальных технологий. Тезисы докладов 9 международной студенческой школы-семинара "Новые информационные технологии" М.: МГИЭМ, 2001. стр.392-394.

28. Крастелева И. Интеллектуальная система продвижения интернет-ресурсов на основе эксплицитного представления знаний. Дипломная работа, МАИ, 2001 г.

29. Лавров С. Программирование. Математические основы, средства, теория. СПб.: БХВ-Петербург, 2001.

30. Лорьер Ж.-Л. Системы искусственного интеллекта: Пер. с фр. М.: Мир, 1991.

31. Лукьянов И.В., Машкович В.Э., Заведеев И.А., Сошников Д.В. Выбор оптимального метода трансуретрального лечения больных с инфравезикальной обструкцией. Тезисы 1-ой Всероссийской научно-практической конференции "Современные эндоскопические технологии в урологии". Уральская медицинская академия, 1999.

32. Лукьянов И.В., Заведеев И.А., Сошников Д.В. Применение экспертного анализа при выработке тактики лечения больных с инфравезикальной обструкцией. Актуальные вопросы лечения онкоурологических заболеваний: Материалы 3-й Всероссийской научной конференции с участием стран СНГ. М.: Российский онкологический научный центр им. Н.Н.Блохина, РАМН, 1999. стр. 131 132.

33. Майкевич Н.В. От информационных пространств к пространству знаний. Онтологии в Интернет, КИИ' 98, Пущино, 1998.

34. Малкина О. Инструментарий для создания систем интеллектуального тестирования в среде Интернет. Дипломная работа, МАИ, 2001 г.

35. Малкина О.И., Сошников Д.В. Создание интерактивных систем адаптивного тестирования в среде Интернет с использованием технологий искусственного интеллекта. Тезисы докладов 9 международной студенческой школы-семинара "Новые информационные технологии" М.: МГИЭМ, 2001. стр.390-392.

36. Марков А.А., Нагорный Н.М. Теория алгорифмов. М.: Наука, 1984.

37. Минский М. Фреймы для представления знаний: Пер. с англ. М.: Энергия, 1979. (Minsky M. A Framework for Representing Knowledge. MIT, Cambridge, 1974.)

38. Морозов М.Н. Логическое программирование. Гипертекстовый курс, 2001 г. `http://www.mari-el.ru/mmlab/home/prolog/study_1.html`

39. Нейлор К. Как построить свою экспертную систему: Пер. с англ. М.: Энергоатомиздат, 1991.

40. Ньюэлл А., Саймон Х. Информатика как эмпирическое исследование: символы и поиск. // В сб. "Лекции лауреатов премии Тьюринга". М.: Мир, 1993. С.334.

41. Ованесбеков Л.Г. Гипертекстовые базы правил. // Тезисы докладов II Санкт-Петербургской конференции "Региональная информатика" РИ-93, Санкт-Петербург, 11-14 мая 1993. Часть 1, с. 130-132.

42. Осуга С. Обработка знаний: Пер. с япон. М.: Мир, 1989.

43. Пирогова Н. Как создать виртуальную корпорацию // Открытые системы, №1, 1998. стр. 62-66.

44. Построение экспертных систем: Пер. с англ./Под ред. Ф.Хейеса-Рота, Д.Уотермана, Д.Лената. М.: Мир, 1987.

45. Робачевский А.М. Операционная система UNIX. СПб.: BHV Санкт-Петербург, 1997.

46. Сизиков Е.В., Сошников Д.В. Онтологическая поисковая система Jewel для реализации интеллектуального поиска в Интернет- и интранет-сетях. Электронный журнал "Труды МАИ" М.: МАИ, 2002, №7. `http://www.mai.ru/projects/mai_works/index.htm`

47. Соколов Н.Е. Использование экспертно-диагностирующих систем повышения эффективности оценки знаний. Тезисы докладов 6 международной студенческой школы-семинара "Новые информационные технологии" М.: МГИЭМ, 1998. стр.97-98.

48. Сошников Д.В. Инструментальные средства для построения встраиваемых продукционных экспертных систем. Тезисы докладов 10-ой юбилейной международной конференции по вычислительной механике и современным прикладным программным системам. М.: МГИУ, 1999. стр.325-326.

49. Сошников Д.В. Инструментарий для построения распределенных интеллектуальных систем. М.: МАИ, Дипломный проект, 1999.

50. Сошников Д.В., Лукьянов И.В., Заведеев И.А. Интеллектуальная учетно-диагностическая программа для учета и диагностики больных заболеваниями предстательной железы. Тезисы докладов 10-ой юбилейной международной конференции по вычислительной механике и современным прикладным программным системам. М.: МГИУ, 1999. стр.326-328.

51. Сошников Д.В. Инструментарий для построения распределенных интеллектуальных систем на базе архитектуры CORBA. Тезисы докладов 7 международной школы-семинара "Новые информационные технологии" М.: МГИЭМ, 1999. стр. 200-201.

52. Сошников Д.В. Построение распределенных интеллектуальных систем на основе распределенной фреймовой иерархии. Тезисы докладов международной научно-практической конференции "Информационные технологии в образовании". Шахты, 2001.

53. Сошников Д.В., Миханов С.В. Технология IASP и её использование для построения интеллектуальных web-приложений. Тезисы докладов международной научно-практической конференции "Информационные технологии в образовании". Шахты, 2001.

54. Сошников Д.В. Инструментарий JULIA для построения распределенных интеллектуальных систем на основе продукционно-фреймового представления знаний. Электронный журнал "Труды МАИ" М.: МАИ, 2002, №7. `http://www.mai.ru/projects/mai_works/index.htm`

55. Таунсенд К., Фохт Д. Проектирование и программная реализация экспертных систем на персональных ЭВМ: Пер. с англ. М.: Финансы и статистика, 1990.

56. Тьюринг А.М. Может ли машина мыслить? Саратов, Издательство ГосУНЦ "Колледж", 1999.

57. Уолрэнд Дж. Телекоммуникационные и компьютерные сети. Вводный курс. М.: Постмаркет, 2001.

58. Уэно Х., Кояма Т., Окамото Т., Мацуби Б., Исидзука М. Представление и использование знаний: Пер. с япон. М.: Мир, 1989.

59. Фландерс Й. ASP взгляд изнутри. М.: ДМК Пресс, 2001.

60. Хоор Ч.Э.Р., Лауэр П.Е. Непротиворечивые дополняющие теории семантики языков программирования. // В сб. Семантика языков программирования. М.: Мир, 1980.

61. Хорошевский В.Ф. Управление проектами, основанное на знаниях, в среде PiES Workbench.// Изв. РАН, серия "Техническая кибернетика", 1993, №5.

62. Шампанер Г., Шайдук А. Обучающие компьютерные системы // Высшее образование в России. 1998, №3. с. 95 96.

63. Элти Дж., Кумбс М. Экспертные системы: концепции и примеры: Пер. с англ. М.: Финансы и статистика, 1987.

64. Berners-Lee T., Hendler J., Lassila O. The Semantic Web. Scientific American, May 2001.

65. Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide, Addison-Wesley, MA, USA, 1999.

66. Bratko I. Programming in Prolog for Artificial Intelligence (3rd edition), Addison-Wesley Publishers, 2001.

67. CORBA: Architechture and Specification. Object Management Group, 1995. (Доступно на сервере `http://www.omg.org`)

68. DESS: Document and Reporting Decision Support System. European Software Laboratories, 1993.

69. Dobson S. A., Burrill V. A. Lightweight databases, *Computer Networks and ISDN Systems*, Vol. 27, No. 6, 1995. pp. 1009 1015.

70. Durfee E.H., Lesser V.R., Corkill D.D. Trends in Distributed Cooperative Problem Solving, IEEE Transactions on Knowledge and Data Engineering, 1(1):63 83, March 1989.

71. Fikes R., Farquhar A. Distributed Repositiories of Highly Expressive Reusable Ontologies.*IEEE Intelligent Systems*, 1999; March/April. pp. 73 79.

72. Fischer K. The Rule-Based Multi-Agent System MAGSY. In *Proceedings of the CKBS'92 Workshop*. DAKE Centre, Keele University, 1993.

73. Fitting M. Fixpoint Semantics for Logic Programming: A Survey. Elsevier Preprint, 1996.

74. Forgy C.L. RETE: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, Vol. 19, No.1, 1982. pp. 17 37.

75. Girratano J. , Riley G. Expert Systems: Principles and Programming. PWS Publishing Company, Boston, 1993. (2nd Ed.)

76. Gruber T. R., A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), 1993. pp. 199-220.

77. Gruber T.R. Towards Principles for the Design of Ontologies used for Knowledge Sharing. *International Journal of Human and Computer Studies*, №43 (5/6), 1995. pp.907 922.

78. Hanson E.N., Hasan M.S. Gator: An optimized discrimination network for active database rule condition testing. Technical Report TR-93-036, CIS Department, University of Florida, 1993.

79. Hanson N.H.,Widom J. An Overview of Production Rules in Database Systems. In *the Knowledge Engineering Review*, Vol.8, No.2, 1993. pp.121 143.

80. Jennings N.R., Sycara K., Wooldridge M., A Roadmap of Agent Research and Development, *Automomous and Multi-Agent Systems*, 1, 1998. pp.275 306.

81. Karp P.D. The Design Space of Frame Knowledge Representation Systems. SRI AI Center Technical Note #520, 1993.

82. Khoroshevsky V.F., Knowledge Based Design of Knowledge Based Systems in PiES WorkBench, In: Proc. of JCKBSE'94, Japan-CIS Symposium on Knowledge Based Software Engineering '94, 1994, p.p. 256-261.

83. Khoroshevsky V.F. Situation Control Software: From Symbol Manipulation Languages Through Knowledge Representation Systems to Semiotic Technologies. In: Proc. of the 1995 ISIC Workshop, 10 IEEE Int.Symp. on Intelligent Control, 27-29 August 1995, Monterey, California, USA, 1995.

84. Khoroshevsky V. F., Maikevich N. V. Knowledge Driven Processing of HTML-Based Information for Intellectual Spaces on Web, In Proceedings of JCKBSE'98, Smolenice, 1998.

85. Kifer M., Lausen G., Wu J. Logical Foundations of Object-Oriented and Frame-Based Languages. Technical Report 90/14, Department of Computer Science, State University of New York at Stony Brook (SUNY), June 1990.

86. Kleinmuntz B., McLean R.S. Computers in behavior science: diagnostic interviewing by digital computer. *Behavior science*, 1968, 13, 75-80.

87. Lesser, V.R. An Overview of DAI: Viewing Distributed AI as Distributed Search, *Journal of Japanese Society for Artificial Intelligence*, Vol. 5, No. 4, 1990.

88. Luger G.F., Stubblefield W.A. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Benjamin Cummings Publishing Company, 1993. (2nd Ed.)

89. Luke S., Heflin J. SHOE 1.0, Proposed Specification, 1997. `http://www.cs.umd.edu/projects/plus/SHOE/`

90. Miranker D.P. TREAT: A New and Efficient Match Algorithm for AI Production Systems. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

91. Nwana H.S. Software Agents: An Overview, *Knowledge Engineering Review*, Vol. 11, No.3, 1996. pp. 1 40.

92. Orfali R., Harkey D., Edwards J. Instant CORBA. Wiley Computer Publishing, 1997.

93. Pfleger K., Hayes-Roth B., An Introduction to Blackboard-Style Systems Organization, KSL Technical Report KSL-98-03, Computer Science Department, Stanford University, 1997.

94. Ramakrishnan K., Ullman, J. A Survey of Research on Deductive Database Systems, *Journal of Logic Programming*, 23(2), 1995. pp. 125 149.

95. Rogerson D. Inside COM. Microsoft Press, 1997.

96. Russel S., Norvig P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 1994.

97. Seng Wai Loke, Adding Logic Programming Behaviour to the World Wide Web, PhD Thesis, Department of Computer Science, The University of Melbourne, Australia, 1998.

98. Sherstnew W.Yu., Worfolomeew A.N., Aleshin A. Yu. FRAME/2 - Application Program Interface for Frame Knowledge Bases, In: Proc. of JCKBSE'94, Japan-CIS Symposium on Knowledge Based Software Engineering'94, 1994.

99. Soshnikov D. An Approach for Creating Distributed Intelligent Systems. In J.-C. Freytag and V. Wolfengagen, editors, *Proceedings of the 1st International Workshop on Computer Science and Information Technologies*, Moscow, Mephi Publishing, 1998. pp. 129 134.

100. Soshnikov D. Software Toolkit for Buiding Embedded and Distributed Knowledge-Based Systems. In *Proceedings of the 2nd International Workshop on Computer Science and Information Technologies, Vol.1*, USATU Publishing, Ufa, 2000. pp. 103 111. (Депонировано в `arXiv:cs.AI/0106054`).

101. Soshnikov D. Technologies for Building Intelligent Web Applications based on JULIA Toolkit. In *Proceedings of the 3rd International Workshop on Computer Science and Information Technologies*, USATU Publishing, Ufa, 2001.

102. Tarski, A. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, 1955, Vol.5. pp 285 309.

103. UML: OMG Unified Modeling Language Formal Specification, Version 1.4. September 2001. (Доступно по адресу `http://www.omg.org/cgi-bin/doc?formal/01-09-67`).

104. Weber J. Special Edition: Using Java. QUE Corporation, 1996.

105. Whorf B.L. Language Thought & Reality. MIT Press, Cambridge, MA, 1956.

106. Wooldridge M. A Knowledge-Theoretic Approach to Distributed Problem Solving. In Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98), 1998.

107. Wörn H., Längle T., Albert M. Distributed Diagnosis for Automated Production Cells. In *Proceedings of the 3rd International Workshop on Computer Science and Information Technologies*, USATU Publishing, Ufa, 2001.

108. Zhang M., Zhang C. Synthesis of Solutions in Distributed Expert Systems. In *Proc. 7th Australian Joint Conference on AI*, 1994. pp.362 369.

109. Microsoft Encarta'98 Encyclopedia, CD-ROM edition. Microsoft Press, 1998.

110. `http://herzberg.ca.sandia.gov/jess/main.html` JESS: Java Expert System Shell.

111. `http://inf.susu.ac.ru/ pollak/expers/commercial/faq-doc-7.htm` Обзор оболочек для создания экспертных систем.

112. `http://kleio.dcn-asu.ru/ai/krug/5/7.shtml` Комличенко В.Н. Новые технологии и автоматизированное обучение.

113. `http://www.amzi.com` Web Site of AMZI Prolog.

114. `http://www.cs.umbc.edu/agents/kse/kif/` Knowledge Interchange Format.

115. `http://www.cs.umbc.edu/kqml` Knowledge Query and Manipulation Language.

116. `http://www.fipa.org/specs/fipa00061` FIPA ACL Message Structure Specification.

117. `http://www.ghg.net/clips/CLIPS.html` CLIPS Expert System Shell.

118. http://www.glasnet.ru/ lukianovi Урологическая online-консультация.

119. http://www.gnu.org.

120. http://www.javasoft.com Java Technology Web Site.

121. http://www.javasoft.com/rmi Remote Method Invokation.

122. http://www.xml.com XML Web Site.

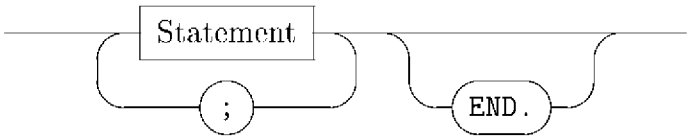
123. http://www.xml-rpc.com XML-RPC Technology Web Site.

124. http://www.ksl.svc.stanford.edu Knowledge System Laboratory: Interactive Ontology Server.

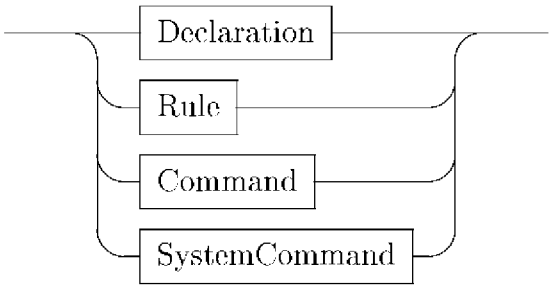
125. http://www.pcp.lanl.gov Turchin V.F. et al. Principia Cybernetica Project.

A. Синтаксис языка представления знаний JFMDL

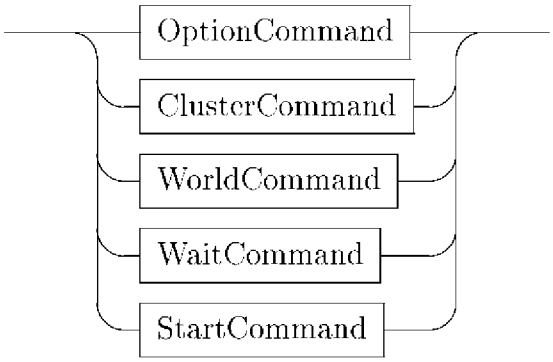
Program



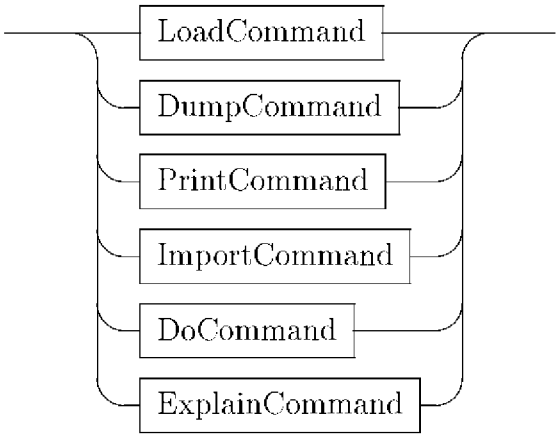
Statement



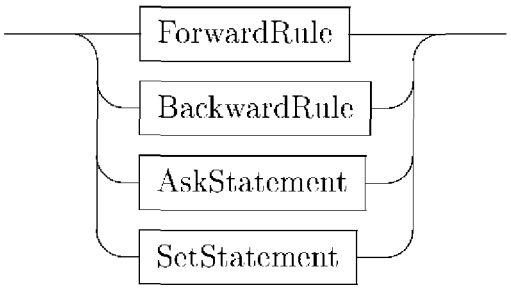
SystemCommand



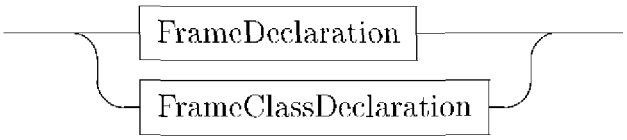
Command



Rule

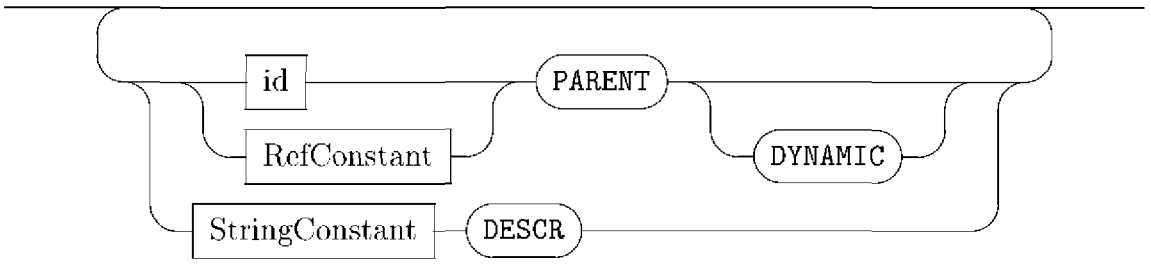


Declaration

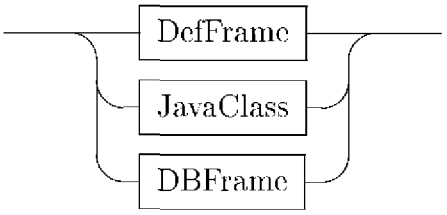


FrameDeclaration

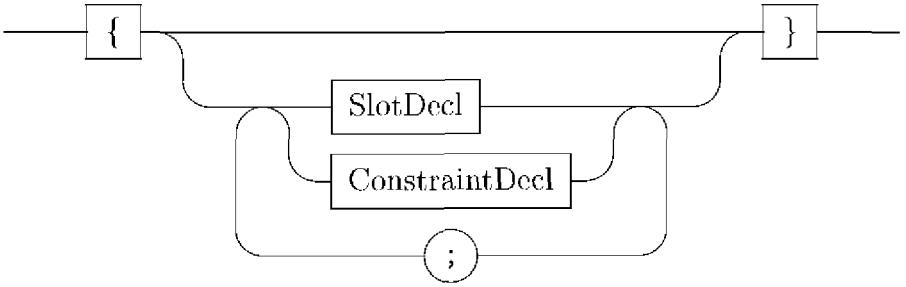
ParentDecl



FrameDecl

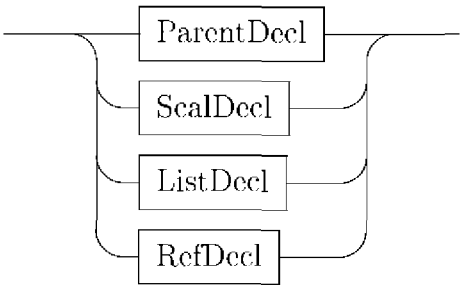


DefFrame



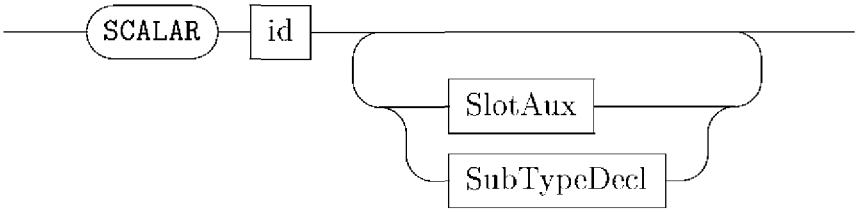
ConstraitDecl

SlotDecl

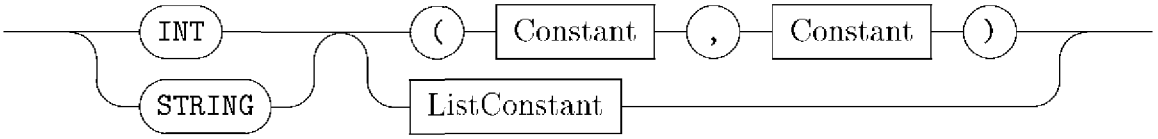


ParentDecl

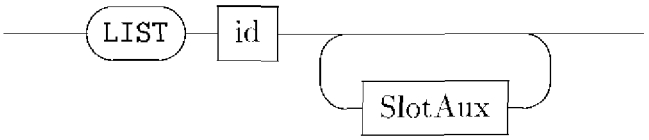
ScalDecl



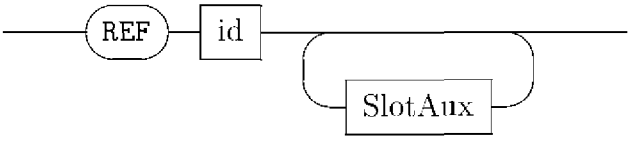
SubTypeDecl



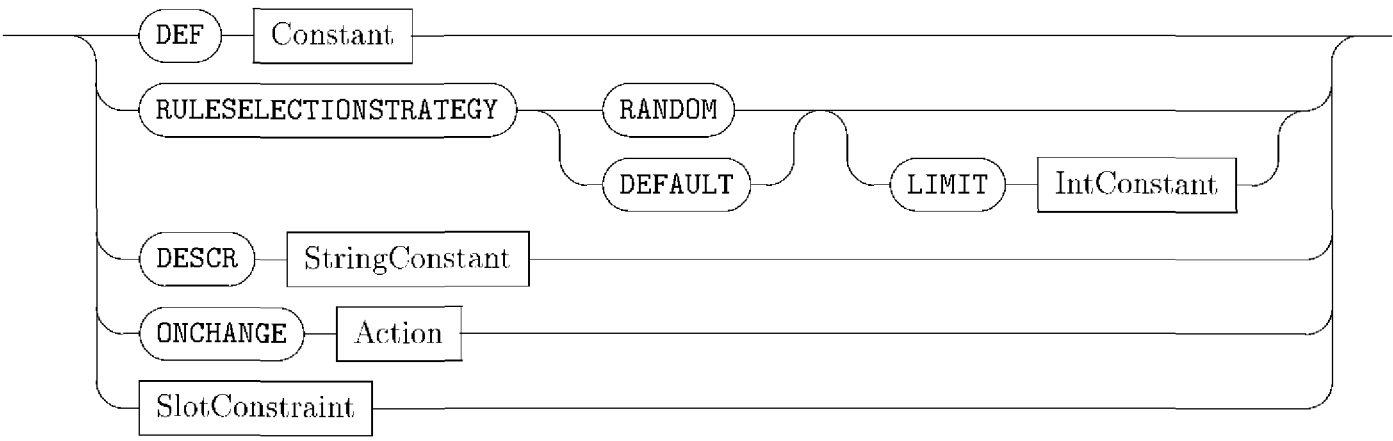
ListDecl



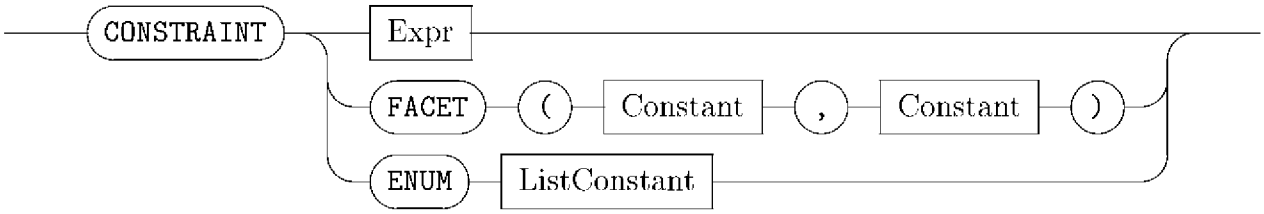
RefDecl



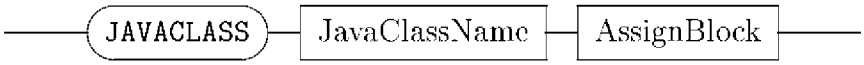
SlotAux



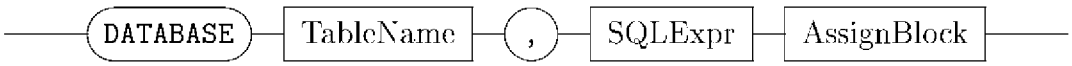
SlotConstraint



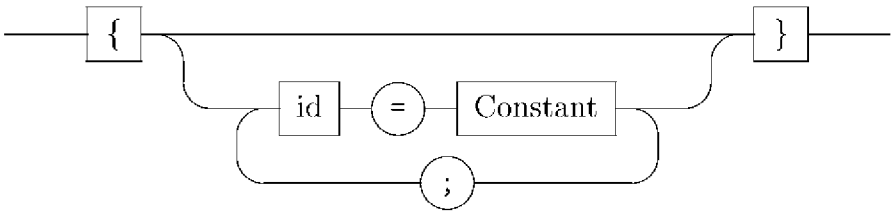
JavaClass



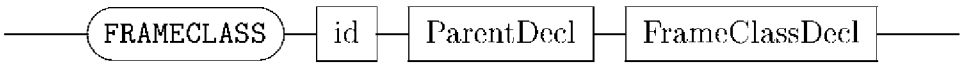
DBFrame



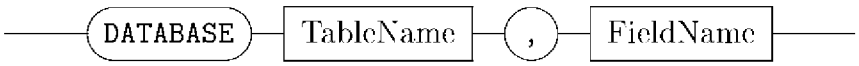
AssignBlock



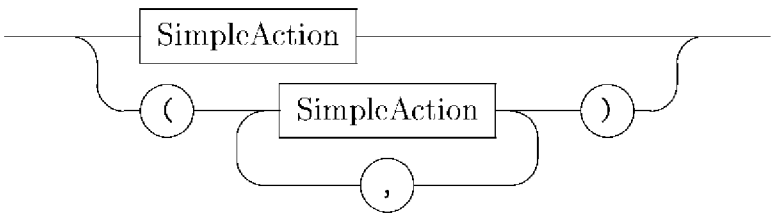
FrameClassDeclaration



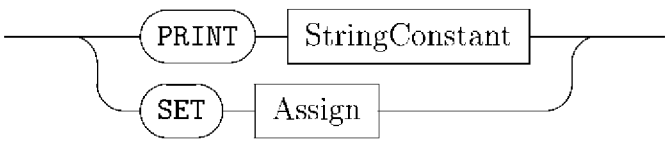
FrameClassDecl



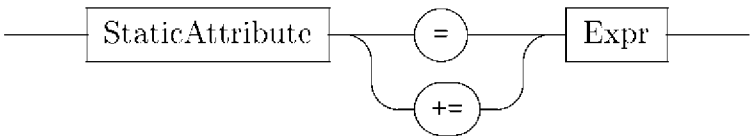
Action



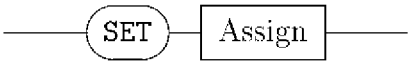
SimpleAction



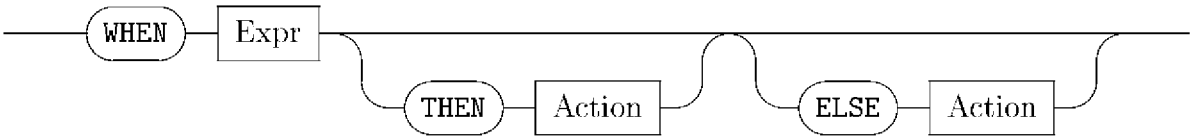
Assign



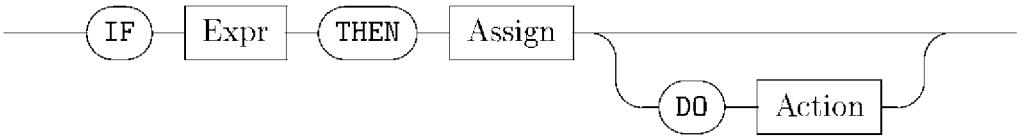
SetStatement



ForwardRule

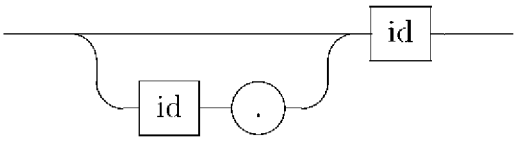


BackwardRule

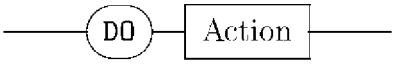


AskRule

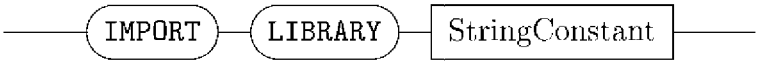
StaticAttribute



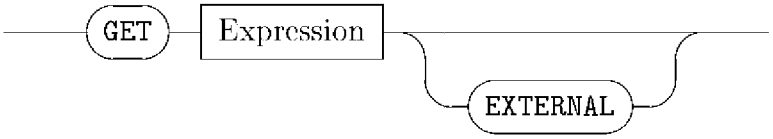
DoCommand



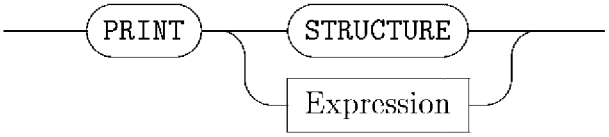
ImportCommand



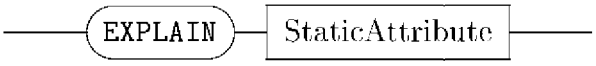
GetCommand



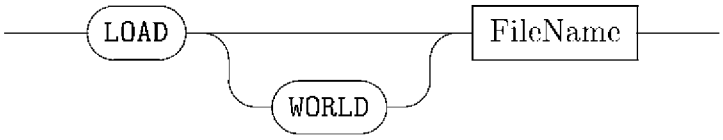
PrintCommand



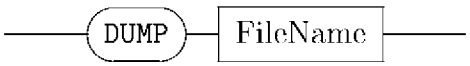
ExplainCommand



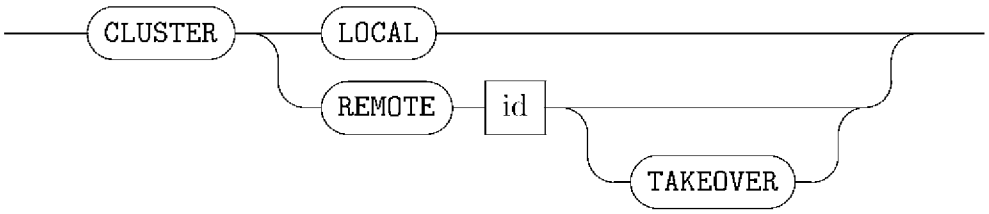
LoadCommand



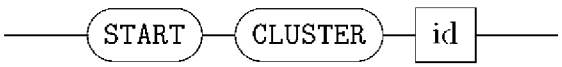
DumpCommand



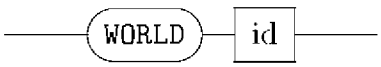
ClusterCommand



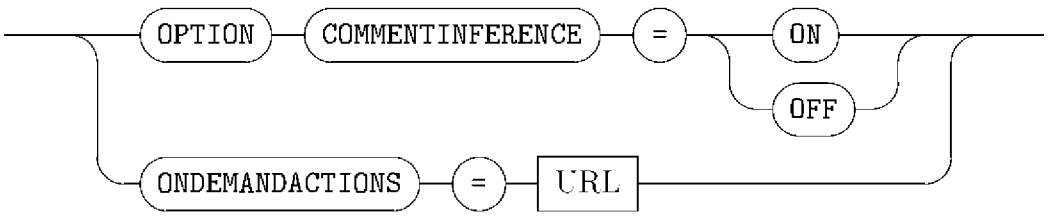
StartCommand



WorldCommand

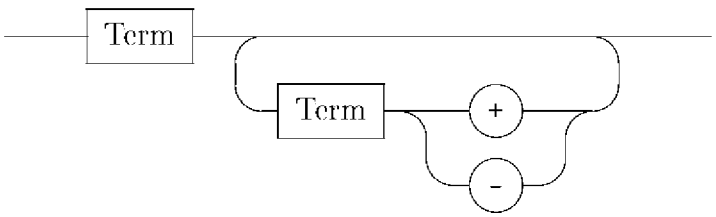


OptionCommand

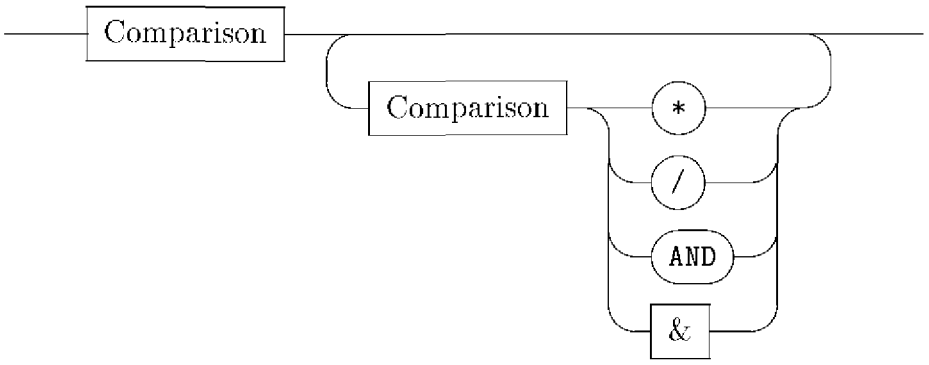


WaitCommand

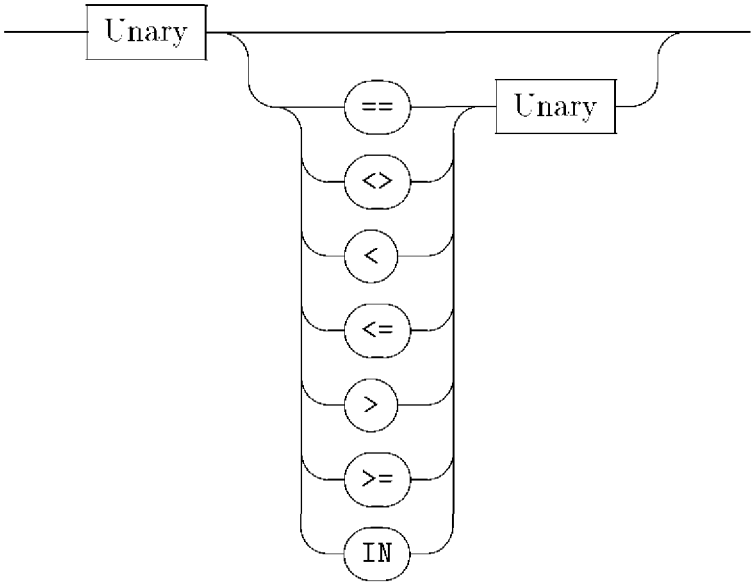
Expr



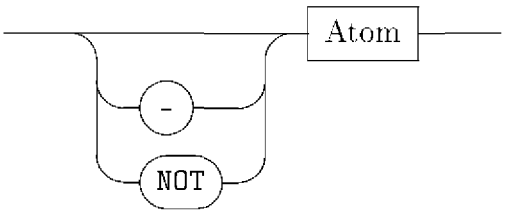
Term



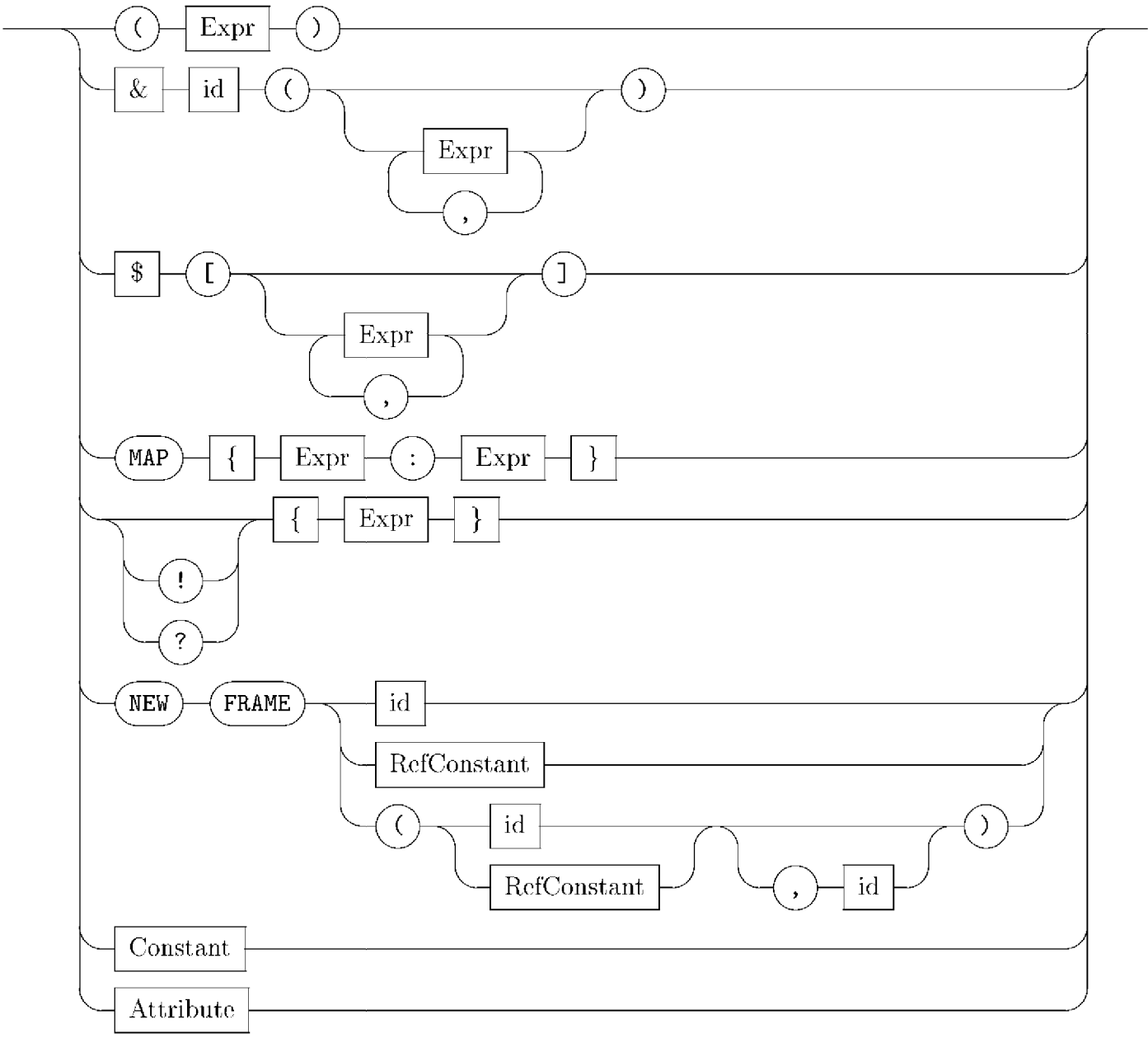
Comparison



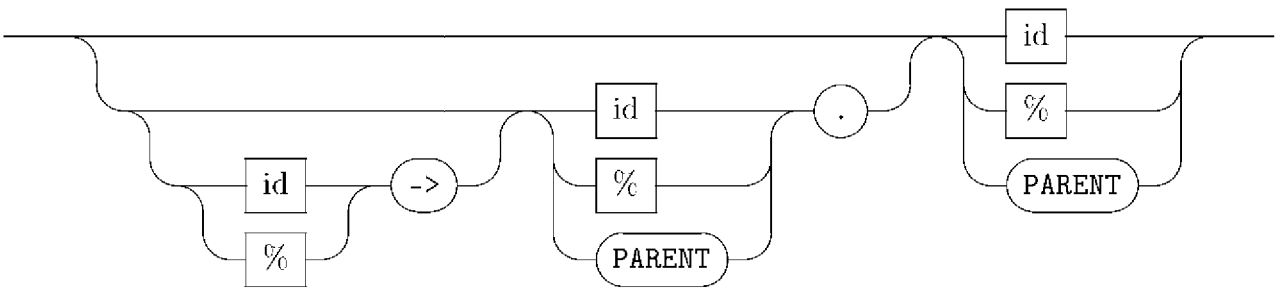
Unary



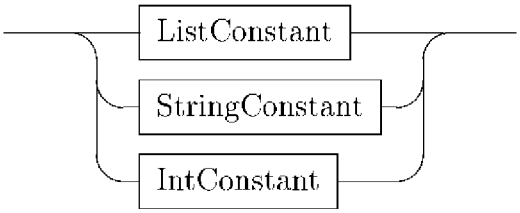
Atom



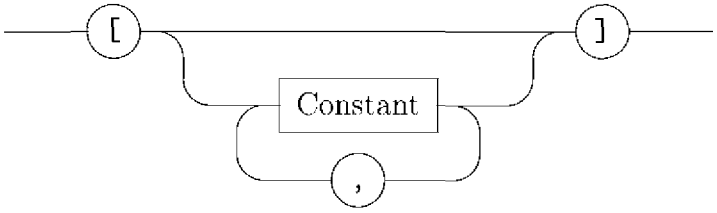
Attribute



Constant

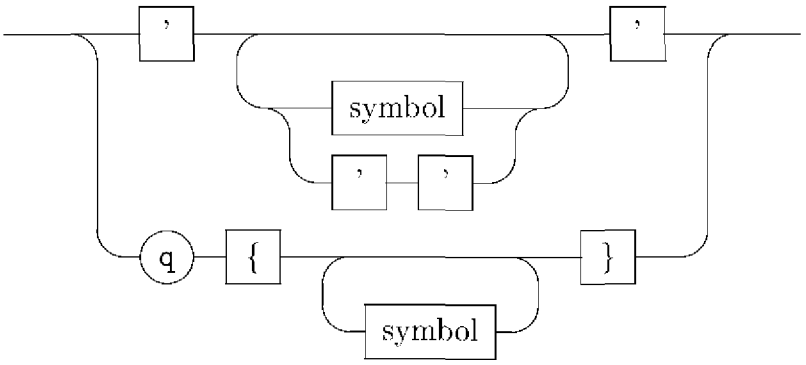


ListConstant

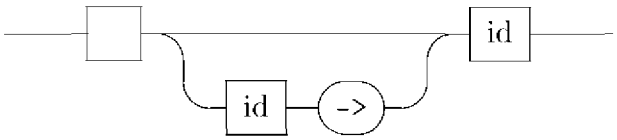


IntConstant

StringConstant



RefConstant



B. Тексты программ

B.1. Инструментарий JULIA

B.1.1. Интерфейс ядра инструментария для поддержки удаленного взаимодействия на CORBA IDL

Файл: `julia.idl`

```
/*
 * JULIA    Java Universal Library for Intelligent Applications
 *
 * (C)1997,98,99,2000 SHWARSICO SOFTWARE, All Rights Reserved
 * (C)1997,98,99,2000 Dmitri Soshnikov, dsh@shwarsico.com
 *
 * This software has been developed by Dmitri Soshnikov as
 * part of his PhD thesis on Distributed Intelligent Systems.
 * For more details, visit http://www.shwarsico.com
 *
 * Version 1.0    Unified Source
 */

module idl
{

#pragma prefix "com.shwars.julia.CORBA"

typedef string TAny;

// Represents Asker
interface Asker
{
 TAny ask (in string text, in TAny choices);
};

// Represents a remote frame interface
interface Frame
{
 TAny get(in string key, in Frame F, in short CT);
 TAny get1(in string key);
 TAny getValue(in string key);
 void executeSetActions(in string key);
 void put(in string key, in TAny value);
 void setValue(in string key, in TAny value);
 void addValue(in string key, in TAny value);
 string getName();
 string getFullName();
 void registerChild(in Frame F);
 void unRegisterChild(in Frame F);
 void setParent(in Frame F);
 Frame getParent();
 void unSetParent(in Frame F);
// Slot inherit(in string name, in Frame F); // TODO: This one difficult to
// handle

 TAny getDescendantsCanonical();
```

```
  void reset();
 };

 // Represents a JULIA world

 interface World
 {
  Frame getFrame(in string frame);
  TAny get(in string frame, in string slot);
  string getName();
  void setAsker(in Asker A);
 };

 interface ClusterManager
 {
  void setWorld(in World W);
  World getWorld(in string world);
  Frame getFrame(in string world, in string frame);
  string getName();
  void setAsker(in Asker A);
 };

};
```

B.1.2. Исходный текст апплета диалоговой консультации

Файл: `consultApp.java`

```
package com.shwars.julia.util;

import com.shwars.julia.*;
import java.awt.*;
import java.lang.*;
import java.applet.*;
import java.util.*;
import java.net.URL;

public class consultApp extends Applet implements Runnable, Asker,
 com.shwars.io.Console
 {
  private static java.awt.Frame af;
  private Thread sysThread = null;
  private Label labelStatus;
  private boolean choiceMade;
  private Choice choicer;
  private TextField textBox;
  private TextArea log;
  private Label quest;
  private Button btnNext;
  private Button btnClose;
  private int stage;
  private String res;
  private String fname;
  private int sw; // size().width
  private boolean isApplet = true;
  private String FrameName = "object", SlotName = "goal";

  private World expSys = null;

  public void reinit()
  {
```

```
      expSys.reset();
    }

    public void init()
      {
       setLayout(null);
       if (isApplet) fname=getParameter("URL");
       sw = size().width;
       labelStatus=new Label("Loading...");
       if (isApplet)
       {
        try
        {
         URL U=getClass().getClassLoader().getResource(fname);
         expSys=World.load(U.openStream());
        }
        catch (Exception e) { expSys=null; };
       }
       else
       {
        expSys = ParserHelper.load(fname);
       }
       if (expSys == null)
       {
        System.out.println("Error parsing input file");
        System.exit(0); // TODO: Exit 1
       }
       expSys.setAsker(new com.shwars.julia.Askers.AdvAsker(expSys,this));
       expSys.setLog(new com.shwars.util.Log(this,Defs.verbosity));
       expSys.setExtension("AUXCONSOLE",this);
       expSys.setExtension("APPLET",this);
       expSys.forceClusterProperties(ClusterManager.IPT_MAIN);
       labelStatus.reshape(20,45,this.size().width 40,20);
       labelStatus.setBackground(new Color(180,180,180));
       add(labelStatus);
       log = new TextArea("JULIA/Consult Engine\n",80,2,
        log.SCROLLBARS_VERTICAL_ONLY);
// JDK 1.0.2          log = new TextArea();
       log.reshape(20,190,this.size().width 40,115);
       log.setEditable(false);
       add(log);
       log.appendText("JULIA/ConsultEngine, "+Defs.VersionShort+", \
(C)1997,98,99,00,01 SHWARSICO SOFTWARE\n");
       quest = new Label();
       quest.reshape(40,100,this.size().width 80,20);
       quest.setBackground(new Color(200,200,200));
       add(quest);
       btnNext = new Button("Next Consultation");
       btnNext.reshape(50,140,size().width/2 100,20);
       btnNext.hide();
       add(btnNext);
       btnClose = new Button("Finish");
       btnClose.reshape(50+size().width/2,140,size().width/2 100,20);
       btnClose.hide();
       add(btnClose);
       stage = 0;
      }
```

```
    public consultApp()
90  {
    }

    public consultApp(boolean b)
    {
95   isApplet=b;
    }

    public void start()
      {
100    println("Consuling for goal "+FrameName+"."+SlotName+"...");
       if(sysThread==null)
         {
          sysThread=new Thread(this);
          sysThread.start();
105      }
      }

    public boolean handleEvent(Event evt)
     {
110   if ((evt.target==this)&&(evt.id == Event.WINDOW_DESTROY))
     {
              stop();
              destroy();
              af.dispose();
115           System.exit(0);
      return true;
     }
       if (evt.id == Event.ACTION_EVENT && evt.target == choicer)
       {
120     choiceMade = true;
        return true;
       }
       if (evt.id == Event.ACTION_EVENT && evt.target == textBox)
       {
125     choiceMade = true;
        return true;
       }
       if (evt.id == Event.ACTION_EVENT && evt.target == btnNext)
       {
130     btnNext.hide();
        btnClose.hide();
        quest.show();
        stage=0;
        reinit();
135     choiceMade=false;
        sysThread.start();
        return true;
       }
       if (evt.id == Event.ACTION_EVENT && evt.target == btnClose)
140    {
        stop();
        destroy();
        af.dispose();
        System.exit(0);
145     return true;
       }
      return super.handleEvent(evt);
```

```
  }
150 public void stop()
    {
     if(sysThread!=null)
      {
       sysThread.stop();
155    sysThread=null;
      }
    }
  public void paint(Graphics g)
160 {
     int i;
     int x,y;
     x=size().width;
     y=size().height;
165
     g.setColor(new Color(192,192,192));
     g.fillRect(0,0,x,y);
     // Caption
     for(i=1;i<=20;i++)
170   {
       g.setColor(new Color(0,0,255*(20 i)/19));
       g.fillRect((i 1)*(x/20),0,(x+1)/20,30);
      }
     g.setFont(new Font("TimesRoman",3,16));
175  g.setColor(Color.black);
     g.drawString("JULIA Expert System Shell",7,22);
     g.setColor(Color.white);
     g.drawString("JULIA Expert System Shell",5,20);
     if (stage==2)
180  {
      g.setColor(Color.black);
      g.setFont(new Font("TimesRoman",Font.BOLD,14));
      g.drawString("Question:",40,90);
      g.setColor(new Color(160,160,255));
185   g.drawRect(20,75,x 40,100);
     }
     if (stage==3)
     {
      g.setColor(Color.black);
190   g.setFont(new Font("Times New Roman Cyr",Font.BOLD+Font.ITALIC,20));
      g.drawString(res,(x g.getFontMetrics().stringWidth(res))/2,120);
      g.setColor(new Color(160,160,255));
      g.drawRect(20,75,x 40,100);
     }
195  g.setColor(Color.black);
     g.drawRect(0,0,x 1,y 1);
    }
  public void run()
200 {
     consultIt();
    }
  private void consultIt()
205 {
       stage = 2;
```

```
            repaint();
            TAny R = expSys.getFrameManager().get(FrameName).get(SlotName);
            if (R==null) res = "No solution found...";
            else res = R.asString();
            stage = 3;
            quest.hide();
            btnNext.show();
            btnClose.show();
            repaint();
        }
    public String readln()
        {
         //deprecated
         return null;
        };
    public void print(String s)
        {
         if (s.equals("#CLEAR")) { log.setText(""); return; }
         labelStatus.setText(s);
         log.appendText(s);
         return;
        };
    public void println(String s)
        {
         if (s.equals("#CLEAR")) { log.setText(""); return; }
         labelStatus.setText(s);
         log.appendText(s+"\n");
         return;
        };
    public void trace(String s)
        {
         return;
        };
    public void comment(String s)
        {
         // label
         log.appendText(s+"\n");
         return;
        };
    public void error(String s)
        {
         // console
         System.out.println("FATAL ERROR: "+s);
         return;
        };
    public TAny ask(String text,Vector posAns)
        {
         //show text
         //answers in box
         String s;
         quest.setText(text);
         if (posAns==null || posAns.size()<1)
         { // text selection
          textBox = new TextField(10);
```

```
        textBox.reshape(40,130,sw - 80,20);
        add(textBox);
        choiceMade = false;
        while (!choiceMade);
        s=textBox.getText();
        remove(textBox);
       }
       else
       {
        choicer = makeChoice(posAns);
        choicer.reshape(40,130,sw - 80,20);
        add(choicer);
        choiceMade = false;
        while(!choiceMade);
        s=choicer.getSelectedItem();
        remove(choicer);
       }
       return new TScal(expSys,s);
      };

    private Choice makeChoice(Vector v)
    {
     Choice c = new Choice();
     for(Enumeration e=v.elements();e.hasMoreElements();)
     {
      TAny T = (TAny)e.nextElement();
      c.addItem(T.asString());
     }
     c.reshape(40,130,size().width - 80,20);
     return c;
    }

    public void setApplet(boolean b)
    {
     isApplet = b;
    }

    public static void main(String av[])
    {
     System.out.println("Julia Interactive Consultation Application/Applet, \
Version 2.1");
     System.out.println(Defs.Name+"\n"+Defs.Copyright);
     System.out.println("Using JULIA Version: "+Defs.VersionShort);
     if (av.length<1)
     {
      System.out.println("ERROR: Argument missing");
      System.out.println(" - Please specify either FDL or JSW file as \
argument");
      System.exit(0);
     }
     af=new java.awt.Frame("Julia Consult Application/Applet, Ver. 2.1/Net");
     af.setResizable(false);
     Applet ca=new consultApp(false);
     ((consultApp)ca).fname=av[0];
     ca.resize(500,340);
     ca.init();
     ca.start();
     af.setLayout(new FlowLayout());
```

```
325     af.add(ca);
        af.resize(500,350);
        af.show();
        ca.repaint();
      }
330
      }
```

B.1.3. Фрагмент исходного текста библиотеки для работы со списками

B.2. Экспертная система продвижения интернет-ресурсов PromoWeb

B.2.1. Исходный тескт базы знаний по методам продвижения Интернет-ресурсов

Файл: `promo.fdl`

```
IMPORT LIBRARY 'com.shwars.julia.Libraries.System';
IMPORT LIBRARY 'inet';

SET goal=PROMOTION.Recommendations;
5
/*  ********************************************** */
/* *                                              * */
/* * PROMOTION KNOWLEDGEBASE                      * */
/* * (C)2000,01 Dmitri Soshnikov, Irina Krasteleva * */
10 /* *                                              * */
/*  ********************************************** */

OPTION COMMENTINFERENCE=OFF;

15 FRAME PROMOTYPE
{
 SCALAR MaxMoney INT;
 SCALAR MoneyLeft INT;
 SCALAR Money INT;
20 SCALAR MoneyAviable;
 SCALAR ResourceType;
 LIST Recommendations DEF [];
};

25 SET PROMOTYPE.MaxMoney = PROMOTION.MaxMoney;
SET PROMOTYPE.MoneyAvailable = NOT(PROMOTYPE.MaxMoney==0);

FRAME Optimisation PARENT PROMOTYPE
{
30
SCALAR NonCheck;
SCALAR HowMuch;
SCALAR DomenName;
SCALAR Dynamique;
35 SCALAR Cash;
SCALAR Navigation;

SCALAR Keywords;

40
};

/*                                    Дополнительные вопросы
```

```
    *                                                    */

ASK Optimisation.NonCheck 'Есть ли у Вашего сайта страницы которые Вы не
    ходите индексировать в поисковых системах (наример, страницы
    администратора)?' ['y-Да','n-Нет'];

ASK Optimisation.HowMuch 'Таких страниц много?' ['y-Да','n-Нет'];

ASK Optimisation.DomenName 'Вы уже зарегистрировали доменное имя?' ['y
    -Да','n-Нет'];

ASK Optimisation.Dynamique 'Содержимое Вашего сайта генерится
    динамически?' ['y-Да','n-Нет'];

ASK Optimisation.Cash 'Вы хотите, чтобы Ваши страницы кэшировал броузер?'
    ['y-Да','n-Нет'];

ASK Optimisation.Navigation 'Навигация на Вашем сайте тоже генерится
    автоматически?' ['y-Да','n-Нет'];

ASK Optimisation.Keywords 'Хотите ли Вы найти наиболее популярные
    ключевые слова для Вашего сайта? Это может занять около 10 минут? ' ['y
    -Да','n-Нет'];


/*                                                   Рекомендации
 *                                                   */

IF SiteType.OftenChanges == 'y' THEN
    Optimisation.Recommendations+='Добавьте <META NAME="\
DOCUMENT-STATE" CONTENT="Dynamique"> в HTML    код страниц, которые
    часто изменяются.';

IF Optimisation.Keywords == 'y' THEN
    Optimisation.Recommendations+=&KEYWORD(SiteTheme.Url);

IF Optimisation.NonCheck == 'y' AND Optimisation.HowMuch == 'y' THEN
    Optimisation.Recommendations+='Создайте файл robots.txt с перечислением
    неиндексируемых страниц. Его формат описан на странице "\
Оптимизация страниц". ';

IF Optimisation.NonCheck == 'y' AND Optimisation.HowMuch == 'n' THEN
    Optimisation.Recommendations+='Добавьте <META NAME="ROBOTS"
    CONTENT="NOINDEX"> в неиндексируемые страницы.';

IF Optimisation.DomenName == 'n'        THEN
    Optimisation.Recommendations+='Поисковые системы индексируют доменное
    имя. Постарайтесь использовать один два ключевых термина в доменном
    имени.';

IF Optimisation.Dynamique AND Optimisation.Cash == 'n'              THEN
    Optimisation.Recommendations+='Для избежания кэширования используйте
    теги: <meta http equiv="pragma" content="no-cache"> и <meta http
    equiv="cache-control" content="no-cache">';

IF Optimisation.Dynamique AND Optimisation.Navigation == 'y'
 THEN
    Optimisation.Recommendations+='Для индексации внутренних страниц сайта
    создайте страницу со всеми ссылками (можно оформить как карту
    сайта) и зарегистрируйте ее в поисковых системах.';

FRAME SearchEngines PARENT PROMOTYPE
{
SCALAR SitePart;
SCALAR ByHand;
```

```
    SCALAR Eng;
    SCALAR Eng2;
    SCALAR RusMoney;
    SCALAR Button;
    };

    /*                                        Дополнительные вопросы
     *                                  */

    ASK SearchEngines.SitePart 'Можно ли выделить в Вашем сайте отдельные
     тематические разделы?' ['y-Да','n-Нет'];

    ASK SearchEngines.ByHand 'Готовы          ли Вы регистрировать сайт в 10
     крупнейших русскоязычных поисковых системах          самостоятельно ?' ['y
     -Да','n-Нет'];

    ASK SearchEngines.Eng 'Вашему сайту нужна регистрация в англоязычных
     системах?' ['y-Да','n-Нет'];

    ASK SearchEngines.Eng2 'Все же я рекомендую зарегистрировать сайт в 10
     крупнейший зарубежных системах, которые поддерживают поиск на русском
     языке. Вы согласны?' ['y-Да','n-Нет'];

    ASK SearchEngines.RusMoney 'Хотите ли Вы использовать платную систему
     регистрации www.registratura.ru ( $40 )?' ['y-Да','n-Нет'];

    ASK SearchEngines.Button 'Хотите ли Вы регистироваться в каталогах, которые
     требуют расположить на сайте рекламную кнопку?' ['y-Да','n-Нет'];

    /*                                        Рекомендации
     *                                  */

    IF   SearchEngines.MaxMoney >= 40
    AND SearchEngines.RusMoney == 'y' THEN
            SearchEngines.Recommendations+='Зарегистрируйте Ваш сайт в
             поисковых системах используя платную службу регистрации
             www.registratura.ru ';

    IF   (SearchEngines.MaxMoney < 40 OR SearchEngines.RusMoney == 'n')
    AND SearchEngines.SitePart == 'y' THEN
            SearchEngines.Recommendations+='Зарегистрируйте в каталогах ресуров
             и поисковых системах каждую тематическую часть Вашего сайта со
             своим описанием.';

    IF   (SearchEngines.MaxMoney < 40 OR SearchEngines.RusMoney == 'n')
    AND SearchEngines.ByHand == 'y'        AND ( SearchEngines.Eng == 'y' OR
     SearchEngines.Eng2 == 'y') THEN
            SearchEngines.Recommendations+='Зарегистрируйте Ваш сайт
             самостоятельно в 20 основных русскоязычных и англоязычных
             поисковых системах. Список Вы найдете в разделе "Поисковые \
    системы"';

    IF   (SearchEngines.MaxMoney < 40 OR SearchEngines.RusMoney == 'n')
    AND SearchEngines.ByHand == 'y'        AND ( SearchEngines.Eng == 'n' AND
     SearchEngines.Eng2 == 'n')          THEN
            SearchEngines.Recommendations+='Зарегистрируйте Ваш сайт
             самостоятельно в 10 основных русскоязычных          поисковых
             системах. Список Вы найдете в разделе "Поисковые системы"';

    IF   (SearchEngines.MaxMoney < 40 OR SearchEngines.RusMoney == 'n')
     THEN
            SearchEngines.Recommendations+='Зарегистрируйте Ваш сайт в 75
             русскоязычных     поисковых системах используя бесплатную систему
```

```
            регистрации http://1ps.ru';

    IF    (SearchEngines.MaxMoney < 40 OR SearchEngines.RusMoney == 'n')
130 AND  SearchEngines.Eng == 'y'            THEN
            SearchEngines.Recommendations+='Зарегистрируйте Ваш сайт в 35
              англоязычных поисковых системах используя бесплатную систему
              регистрации www.addme.com';


    /*                                    Тематические каталоги ресурсов
     *                                 */
135
    IF    SiteTheme.Main == 'Дети' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог детских
              ресурсов      www.kinder.ru';

    IF    SiteTheme.Main == 'Медицина'
140 AND SiteTheme.Psiho == 'y' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              по психологии http://catalog.psyinfo.net ';

    IF    (SiteTheme.Main == 'Города и Регионы' OR SiteTheme.Main == 'Туризм')
    AND  SiteTheme.Sibir == 'y' THEN
145         SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              посвященный Сибири http://intersib.ab.ru              ';

    IF    SiteTheme.Main == 'Информационные Технологии' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог IT
              ресурсов www.citforum.ru/lists ';

150 IF     SiteTheme.Main == 'Программы'       THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              для программистов http://unreal.aic.ru/vbs/wr                ';

    IF    SiteTheme.Main == 'Развлечения'
    AND SiteTheme.Talk == 'y' THEN
155         SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              виртуального общения      http://virtualist.al.ru              ';

    IF    SiteTheme.Main == 'Спорт' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              о спорте http://sport.kc.ru              ';

160 IF    SiteTheme.Main == 'Железо' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              о компьютерном железе http://kpdlabs.uka.ru/rus/link.htm ';

    IF    ( SiteTheme.Main == 'Развлечения'          OR SiteTheme.Main ==
     'Юмор' OR  SiteTheme.Main == 'MP3 и музыка' OR      SiteTheme.Main ==
     'Кино' OR SiteTheme.Main == 'Театр')
    AND SearchEngines.Button == 'y' THEN
165         SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог
              развлекательных ресурсов (для регистрации Вам придется разместить
              рекламную кнопку на своем сайте) http://www.loner.sp.ru/
              //';

    IF    SiteTheme.Main ==      'Безопасность' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог сайтов
              посвященных безопасности www.sec.ru';

170 IF    ( SiteTheme.Main == 'Компании' OR SiteTheme.Main == 'Товары и
     Услуги' )THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог фирм и
              предприятий СНГ http://www.pms.donetsk.ua/pmsdb/';
```

```
    IF    SiteTheme.Main == 'Города и Регионы'
    AND SiteTheme.Moscow == 'y' THEN
175         SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог ресурсов
              города Москвы http://city.mos.ru/';

    IF    SiteTheme.Main == 'СМИ'      THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог
              русскоязычных газет и журналов http://catalog.press.net.ru/';

180 IF    SiteTheme.Main == 'Дети'       THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог
              www.7ya.ru';

    IF    SiteTheme.Main == 'Программы'
    AND SiteTheme.Linux == 'y' THEN
185         SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог http://
              //ru.linuxstart.com';

    IF    SiteTheme.Main == 'Образование'
    AND SiteTheme.Student == 'y' THEN
            SearchEngines.Recommendations+='Добавьте Ваш сайт в каталог
              www.students.ru';
190
    /*                                        Делим деньги
     *                                    */
    IF    PROMOTION.FirstVisitor == 'y' AND PROMOTION.SiteChanges == 'n'
     AND  SearchEngines.RusMoney == 'y'          THEN
            SearchEngines.Money=40;

195 IF PROMOTION.FirstVisitor == 'n' OR PROMOTION.SiteChanges == 'y' OR
     SearchEngines.RusMoney == 'n'          THEN
            SearchEngines.Money=0;


    FRAME Statistic PARENT PROMOTYPE
200 {
    SCALAR Place;
    SCALAR PlaceCounter;
    SCALAR CounterLanguage;
    };
205
    /*                                        Дополнительные вопросы
     *                                    */

    ASK Statistic.Place 'Предлагает ли Ваш провайдер статистику посещаемости
     сайта?' ['y=Да','n=Нет'];

210 ASK Statistic.PlaceCounter 'Хотите ли Вы расположить на своем сайте систему
     анализа статистики и посещаемости?' ['y=Да','n=Нет'];

    ASK Statistic.CounterLanguage 'Вы хотите использовать русскоязычную или
     англоязычную систему статистики?' ['rus=Русскоязычную','eng=Англоязычную'];

    /*                                        Рекомендации
     *                                    */
215
    IF Statistic.Place == 'n'          AND SiteType.PlaceScript == 'n'          AND
     Statistic.PlaceCounter == 'y'          AND Statistic.CounterLanguage == 'rus'
     THEN
            Statistic.Recommendations+='Расположите на всех страницах Вашего
              сайта бесплатную систему анализа посещаемости
              www.spylog.ru';

    IF Statistic.Place == 'n'          AND SiteType.PlaceScript == 'n' AND
```

```
    Statistic.PlaceCounter == 'y'              AND Statistic.CounterLanguage == 'eng'
    THEN
220         Statistic.Recommendations=='Расположите на всех страницах Вашего
              сайта бесплатную систему анализа посещаемости
              www.hitbox.com';

    IF Statistic.Place == 'n'              AND SiteType.PlaceScript == 'y'          THEN
            Statistic.Recommendations=='Расположите на Вашем сайте скрипт
              анализа посещаемости. Код можно взять на http://
              //ru.linuxstart.com/applications/networking/web/statistics.html';

225
    FRAME BannerExchange PARENT PROMOTYPE
    {
     SCALAR PlaceBanner;
     SCALAR PlaceBanner2;
230  SCALAR PlaceCounter;
     SCALAR NumberVisitors;
     SCALAR FormRegistration;
    };

235 /*                                          Дополнительные вопросы
     *                                  */

    SET BannerExchange.MoneyLeft = PROMOTION.MaxMoney
     SearchEngines.Money;

    ASK BannerExchange.PlaceBanner 'Готовы ли Вы расположить на своем сайте "\
    чужие" баннеры (по обмену)?' ['y=Да','n=Нет'];
240
    ASK BannerExchange.Network 'Хотите ли Вы участвовать в бесплатной сети
     обмена баннерами?' ['y=Да','n=Нет'];

    ASK BannerExchange.PlaceBanner2 'Возможно, сделать раздел "Друзья сайта"
     или "Полезные ссылки" и публиковать "чужие" баннеры там?' ['y=Да','n=Нет'];

245 ASK BannerExchange.PlaceCounter 'Готовы ли Вы расположить на своем сайте
     счетчики участия в рейтингах?' ['y=Да','n=Нет'];

    ASK BannerExchnage.Thematical 'Вас интересует регистрация в
     узкоспециализированных рейтингах?' ['y=Да','n=Нет'];

    ASK BannerExchange.NumberVisitors 'Количество посетителей Вашего сайта
     в день больше 30?' ['y=Да','n=Нет'];
250
    ASK BannerExchange.FormRegistration 'У Вас на сайте есть формы для
     регистрации?' ['y=Да','n=Нет'];


    /*                                          Рекомендации
     *                                  */
255
    IF    BannerExchange.PlaceBanner == 'y'
    AND BannerExchange.NumberVisitors == 'y'
    AND BannerExchange.Network == 'y'
    AND SiteTheme.Main == 'Автомобили'
260 AND SiteType.LowPeople == 'n'          THEN
            BannerExchange.Recommendations=='Зарегистрируйтесь в бесплатной сети
              обмена баннерами для сайтов автомобильной тематики http://
              //rotabanner.auto.ru';

    IF    BannerExchange.PlaceBanner == 'y'
    AND BannerExchange.NumberVisitors == 'y'
265 AND BannerExchange.Network == 'y'
    AND SiteTheme.Main == 'Медицина'
```

```
    AND SiteType.LowPeople -- 'n'          THEN
          BannerExchange.Recommendations--'Зарегистрируйтесь в бесплатной сети
            обмена баннерами для сайтов медицинской тематики
            www.rusmedserv.com';

270
    IF    BannerExchange.PlaceBanner -- 'y'
    AND BannerExchange.NumberVisitors -- 'y'
    AND BannerExchange.Network -- 'y'
    AND SiteTheme.Main <> 'Автомобили'
275 AND SiteTheme.Main <> 'Медицина'
    AND SiteType.LowPeople -- 'y'          THEN
          BannerExchange.Recommendations--'Зарегистрируйтесь в бесплатной сети
            обмена баннерами www.linkexchange.ru. В настройках Вы сможете
            выбрать только сайты, близкие к Вам по тематике для показа на них
            Ваших баннеров.';


280 IF    BannerExchange.PlaceBanner -- 'y'
    AND BannerExchange.NumberVisitors -- 'y'
    AND BannerExchange.Network -- 'y'
    AND SiteTheme.Main <> 'Автомобили'
    AND SiteTheme.Main <> 'Медицина'
285 AND SiteType.LowPeople -- 'y'          THEN
          BannerExchange.Recommendations--'Зарегистрируйтесь в бесплатной сети
            обмена баннерами www.linkexchange.ru. В настройках Вы сможете
            выбрать только сайты, близкие к Вам по тематике для показа на них
            Ваших баннеров.';

    IF    BannerExchange.MoneyLeft > 500       AND BannerExchange.PlaceBanner
     -- 'n'
     AND BannerExchange.PlaceBanner2 -- 'n' THEN
          BannerExchange.Recommendations--'Используя рейтинги http://
            //top100.rambler.ru, www.list.ru выберите наиболее популярные
            //сайты Вашей тематике. Предложите опубликовать (платно) свой
            //баннер.';
290
    IF    BannerExchange.MoneyLeft > 500       AND ( BannerExchange.PlaceBanner
     -- 'y' OR
     BannerExchange.PlaceBanner2 -- 'y') THEN
          BannerExchange.Recommendations--'Используя рейтинги http://
            //top100.rambler.ru, www.list.ru выберите наиболее популярные
            //сайты Вашей тематике. Предложите опубликовать (платно) свой
            //баннер или обменяться баннерами.';

    IF    BannerExchange.PlaceBanner -- 'y'
295 AND BannerExchange.NumberVisitors -- 'y'
    AND BannerExchange.Network -- 'y'
    AND BannerExchange.FormRegistration -- 'y'          THEN
          BannerExchange.Recommendations--'Разбейте процесс регистрации на
            несколько страниц и расположите на каждой баннер(ы) сети обмена';

300
    IF    BannerExchange.PlaceCounter -- 'y' THEN
          BannerExchange.Recommendations--'Расположите у себя на сайте
            счетчик рейтингов www.toplist.ru';

    IF SiteTheme.Main -- 'Спорт'
305 AND SiteTheme.Official -- 'y'          THEN
          BannerExchange.Recommendations--'Добавьте Ваш сайт в список
            официальных сайтов спортивных организаций, федераций и изданий
            на www.sport express.ru/links.shtml';
```

```
    /*                                        Конкретные примеры
     *                                  */

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Бизнес и Финансы'          THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          www.rbc.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Информационные Технологии'          THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          www.citforum.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Автомобили'        THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          www.auto.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Медицина'        THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          www.rusmedserv.com';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Железо'        THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          http://ixbt.stack.net';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Спорт'        THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          www.sport express.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Связь'
AND SiteTheme.Mobile -- 'y' THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайтах
          http://sotovik.ru          и www.sota1.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Кино'
OR SiteTheme.Main -- 'Театр' THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          www.afisha.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Автомобили'        THEN
        BannerExchange.Recommendations--'Купите баннерные показы на сайте
          http://autoport.ru';

IF BannerExchange.MoneyLeft > 500
AND (SiteTheme.Main -- 'Провайдеры'
OR  SiteTheme.Main -- 'Информационные Технологии')
THEN  BannerExchange.Recommendations--'Купите баннерные показы на сайте
  http://internet.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'Работа'
THEN  BannerExchange.Recommendations--'Купите баннерные показы на сайте
  http://rabota.ru';

IF BannerExchange.MoneyLeft > 500
AND SiteTheme.Main -- 'MP3 и музыка'
THEN  BannerExchange.Recommendations--'Купите баннерные показы на сайте
  http://music.ru';
```

```
360
    IF BannerExchange.MoneyLeft > 500
    AND SiteTheme.Main == 'Спорт'
    THEN BannerExchange.Recommendations+='Купите баннерные показы на сайте
     http://sportnews.ru';

365 IF BannerExchange.MoneyLeft > 500
    AND SiteTheme.Main == 'Игры'
    THEN BannerExchange.Recommendations+='Купите баннерные показы на сайте
     http://gameport.ru';


370 /*                                        Делим деньги
     *                                  */

    IF BannerExchange.MoneyLeft <          500 THEN BannerExchange.Money=0;
    IF BannerExchange.MoneyLeft >=         500 AND BannerExchange.MoneyLeft <
     5000 THEN BannerExchange.Money=BannerExchange.MoneyLeft*3/4;
    IF BannerExchange.MoneyLeft >=         5000 THEN BannerExchange.Money
     =BannerExchange.MoneyLeft*4/5;
375
    FRAME MailingLists PARENT PROMOTYPE
    {
    };

380 SET MailingLists.MoneyLeft = PROMOTION.MaxMoney    BannerExchange.Money
      SearchEngines.Money ;

    /*                                        Рекомендации
     *                                  */

    IF MailingLists.MoneyLeft > 0 THEN MailingLists.Recommendations+='Купите
     рекламу в близких по теме списках рассылки на сайтах www.subscribe.ru и
     www.maillist.ru';
385
    IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Информационные
     Технологии' THEN MailingLists.Recommendations+='Купите рекламу в списках
     рассылки на сайте www.citforum.ru ($100    20 000 подписчиков).';

    IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Бизнес и Финансы'
     OR SiteTheme.Main == 'Банки' THEN MailingLists.Recommendations+='Купите
     рекламу в списках рассылки на сайте www.k2kapital.com.';

390 IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Медицина' THEN
     MailingLists.Recommendations+='Купите рекламу в списках рассылки на сайте
     www.rusmedserv.com.';

    IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Железо' THEN
     MailingLists.Recommendations+='Купите рекламу в списках рассылки на сайте
     ixbt.stack.net.';

    IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Связь' AND
     SiteTheme.Mobile == 'y' THEN MailingLists.Recommendations+='Купите
     рекламу в списках рассылки на сайте www.sota1.ru';
395
    IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Дети' THEN
     MailingLists.Recommendations+='Купите рекламу в списках рассылки на сайте
     www.7ya.ru';

    IF MailingLists.MoneyLeft > 100 AND SiteTheme.Main == 'Образование' AND
     SiteTheme.Student == 'y' THEN MailingLists.Recommendations+='Купите
     рекламу в списках рассылки на сайте www.referat.ru';

400 /*                                        Делим деньги
     *                                  */
```

```
IF MailingLists.MoneyLeft <= 0 THEN MailingLists.Money=0;
IF MailingLists.MoneyLeft <= 200 AND MailingLists.MoneyLeft > 0 THEN
 MailingLists.Money=MailingLists.MoneyLeft;
IF MailingLists.MoneyLeft <= 500 AND MailingLists.MoneyLeft > 200 THEN
 MailingLists.Money=MailingLists.MoneyLeft/2;
IF MailingLists.MoneyLeft <= 5000 AND MailingLists.MoneyLeft > 500 THEN
 MailingLists.Money=MailingLists.MoneyLeft/4;
IF MailingLists.MoneyLeft > 5000 THEN MailingLists.Money =
 MailingLists.MoneyLeft/8;

FRAME BBS PARENT PROMOTYPE { };

/* Рекомендации
 */

IF SiteTheme.Main == 'Работа' THEN
      BBS.Recommendations+='Опубликуйте информацию о себе в
       конференциях relcome.commerce.job и fido.mo.job';

IF SiteTheme.Main == 'Железо' THEN
      BBS.Recommendations+='Опубликуйте информацию о себе в
       конференции http://ixbt.stack.net';

IF SiteTheme.Main == 'Спорт' THEN
      BBS.Recommendations+='Опубликуйте информацию о себе в
       конференции www.sport express.ru';

IF SiteTheme.Main == 'Дети' THEN
      BBS.Recommendations+='Опубликуйте информацию о себе в
       конференции и на доске объявлений сайта www.7ya.ru';


FRAME CommercialAdvertising PARENT PROMOTYPE
{
};
SET CommercialAdvertising.MoneyLeft = PROMOTION.MaxMoney
 BannerExchange.Money SearchEngines.Money MailingLists.Money;

/* Рекомендации
 */

IF CommercialAdvertising.MoneyLeft<50
AND SiteType.OfflineFirm == 'y' THEN
     CommercialAdvertising.Recommendations=='Бесплатно разместите о себе
      информацию в справочнике "Желтые страницы" www.yellowpages.ru';


IF CommercialAdvertising.MoneyLeft>=50
AND SiteType.OfflineFirm == 'y' THEN
     CommercialAdvertising.Recommendations=='Бесплатно разместите о себе
      информацию в справочнике "Желтые страницы" www.yellowpages.ru.
      Разместите подробную информацию на этом же сайте платно $50';


IF CommercialAdvertising.MoneyLeft>0
AND SiteTheme.Main == 'Работа'
AND SiteTheme.Agency =='y' THEN
     CommercialAdvertising.Recommendations=='Разместите о себе платную
      информацию на сайте www.job.ru в разделе "Агентства"';

IF CommercialAdvertising.MoneyLeft>50
AND SiteTheme.Main == 'Автомобили'
```

```
    AND SiteTheme.Autosalon ==''y'' THEN
        CommercialAdvertising.Recommendations==''Разместите платную информацию
         на сайте www.adb.ru, информацию о машинах добавте в базу данных.
         $50'';

    IF    CommercialAdvertising.MoneyLeft>0
455 AND SiteTheme.Main == ''Железо''
    AND SiteTheme.Autosalon ==''y'' THEN
        CommercialAdvertising.Recommendations==''Разместите платную информацию
         на сайте ixbt.stack.net со ссылкой на Ваш сайт'';

    IF    CommercialAdvertising.MoneyLeft>0
460 AND  SiteTheme.Main == ''Связь''
    AND  SiteTheme.Mobile == ''y'' THEN
        CommercialAdvertising.Recommendations==''Разместите платную информацию
         о себе в базе данных на сайтах http://sotovik.ru и
         //www.sota1.ru'';

    IF    CommercialAdvertising.MoneyLeft>0
465 AND  SiteType.OfflineFirm == ''y'' THEN
        CommercialAdvertising.Recommendations==''Разместите платную информацию
         о своей продукции в базе данных на сайте www.torg.ru'';


    IF    CommercialAdvertising.MoneyLeft>0 THEN
470     CommercialAdvertising.Recommendations==''Используя каталог ресурсов
         www.list.ru выберите наиболее популярные сайты Вашей
         тематике. Предложите опубликовать (возможно, платно) информацию о
         себе.'';


    IF    ( SiteTheme.Main == ''Дети'' OR SiteTheme.Main == ''Образование''
     )
    AND  SiteTheme.Charity == ''y'' THEN
475     CommercialAdvertising.Recommendations==''Разместите бесплатную
         информацию о себе в базе данных на сайте www.ngo.org.ru'';


    /*                                    Делим деньги
     *                                 */

480 SET CommercialAdvertising.Money = CommercialAdvertising.MoneyLeft;

    FRAME SiteFeatures PARENT PROMOTYPE
    {
    SCALAR EffortsInvolved;
485 };

    /*                                    Дополнительные вопросы
     *                                 */

    ASK SiteFeatures.EffortsInvolved ''Укажите, какие усилия Вы тратите на
     обновление сайта?'' ['hi=Существенные','med=Средние','lo=Незначительные'];
490 ASK SiteFeatures.EBusiness ''Вы хотите продавать свою продукцию через
     Internet?'' ['y=Да','n=Нет'];

    /*                                    Рекомендации
     *                                 */
    //SET SiteFeatures.Recommendations+=''Расположите на сайте ссылку
    //"Добавить в избранное", форму "Отправить сообщение о сайте друзьям" и
    //баннер 60x20 с предложением разместить на сайте посетителя.'';

495 IF    SiteType.OftenChanges == ''y'' AND SiteFeatures.EffortsInvolved <> ''hi''
     THEN
```

```
        SiteFeatures.Recommendations+='Публикуйте на первой странице новости
          сайта';

    IF  SiteType.OftenChanges == 'y' AND SiteFeatures.EffortsInvolved ==
     'hi'  THEN
        SiteFeatures.Recommendations+='Сделайте на сайте систему "ушек"
          графических ссылок на различные разделы сайта. Такая система
          значительно увеличивает посещаемость внутренних страниу сайта.
          Пример на сайте www.estart.ru';
500
    IF  SiteType.OftenChanges == 'y' AND SiteType.PlaceScript == 'y'
     THEN
        SiteFeatures.Recommendations+='Сделайте список рассылки новостей
          Вашего сайта. Пример можно взять на http://
          //www.cgi download.com/html/download/list.htm';

    IF  SiteType.OfflineFirm == 'y'
505 AND SiteType.PlaceScript == 'y' THEN
        SiteFeatures.Recommendations+='Сделайте форму заказа (отправление по
          e mail) Вашей продукции через Internet. Пример можно взять на http:
          //www.cgi download.com';

    IF  SiteType.OfflineFirm == 'y'
    AND SiteType.PlaceScript == 'y'
510 AND SiteFeatures.EBusiness == 'y' THEN
        SiteFeatures.Recommendations+='Сделайте покупку Вашей продукции
          через Internet с использованием системы Assist www.assist.ru';

    FRAME OfflineMarketing PARENT PROMOTYPE
    {
515 SCALAR AddMoney;
    SCALAR BusinessCard;

    };

520 /*                                        Дополнительные вопросы
     *                                  */

    ASK OfflineMarketing.BusinessCard 'У Вашей фирмы есть предметы offline
     рекламы (визитки, плакаты, рекламные объявления)?' ['y=Да','n=Нет'];

    ASK OfflineMarketing.AddMoney 'Укажите, какие Вы планируете траты на
     рекламу offline?' ['none=Никаких','low=Низкие','med=Средние','hi=Высокие'];
525
    /*                                        Рекомендации
     *                                  */

    IF OfflineMarketing.AddMoney<>'0'
    AND OfflineMarketing.AddMoney<>'lo'
530 AND SiteTheme.Main == 'Города и Регионы'
    AND SiteTheme.Moscow == 'y'
    AND SiteTheme.MoscowForVisitors=='y' THEN
        OfflineMarketing.Recommendations+='Разместите рекламу в газете Moscow
          Times (минимальный заказ $1000)';

535 IF OfflineMarketing.AddMoney <> '0'
    AND SiteTheme.Main =='Программы' THEN
        OfflineMarketing.Recommendations+='Разместите рекламу в журнале
          Компьютерра (от $200)';

    IF OfflineMarketing.AddMoney == 'hi' THEN
540     OfflineMarketing.Recommendations+='Устройте презентацию Вашего сайта
          (открытие, годовщина...)';

    IF OfflineMarketing.BusinessCard == 'y' THEN
```

```
        OfflineMarketing.Recommendations+-'Везде на рекламной информации Вашей
          фирмы (визитки, плакаты, рекламные объявления) публикуйте адрес
          Вашего сайта.';

IF    OfflineMarketing.AddMoney == 'hi'
AND SiteTheme.Main == 'Связь'
AND SiteTheme.Mobile == 'y' THEN
        CommercialAdvertising.Recommendations--'Разместите информацию о себе в
          журналах Mobile News, Russian Mobile, MTC 2000';

IF    OfflineMarketing.AddMoney == 'hi'
AND SiteTheme.Main == 'Кино'
OR  SiteTheme.Main == 'Театр' THEN
        CommercialAdvertising.Recommendations--'Разместите информацию о себе в
          журнале "Афиша" (от $700)';

IF OfflineMarketing.AddMoney <> '0'
AND SiteType.OfflineFirm ==-'y' THEN
    OfflineMarketing.Recommendations+-'Разместите рекламу в справочнике "\
Желтые страницы". Детали можно узнать на сайте www.yellowpages.ru';

FRAME SiteType
{
 SCALAR OftenChanges;
 SCALAR PlaceScript;
 SCALAR LowPeople;
 SCALAR OfflineFirm;
};

// Получаем основные свойства сайта, которые пригодятся для некоторых
// типов рекламы

ASK SiteType.OftenChanges 'Содержимое Вашего сайта часто меняется?' ['y
  -Да','n-Нет'];

ASK SiteType.PlaceScript 'Есть ли у Вас возможность использовать CGI
  скрипты?' ['y-Да','n-Нет'];

ASK SiteType.LowPeople 'Сайт рассчитан на узкий круг пользователей?' ['y
  -Да','n-Нет'];

ASK SiteType.OfflineFirm 'Ваш сайт представляет фирму (товар, услугу),
  которая существует offline?' ['y-Да','n-Нет'];

FRAME SiteTheme
{
 SCALAR Main;
 SCALAR Url;

 SCALAR Psiho;
 SCALAR Sibir;
 SCALAR Talk;
 SCALAR Moscow;
 SCALAR MoscowForVisitors;
 SCALAR Agency;
 SCALAR Autosalon;
 SCALAR Official;
 SCALAR Mobile;
 SCALAR Linux;
 SCALAR Charity;
 SCALAR Student;
};

// Получаем основную тему сайта
```

```
    ASK SiteTheme.Main 'Выберите раздел, к которому Вы бы отнесли Ваш сайт:'
        ['Автомобили','Банки','Безопасность','Бизнес и Финансы','Города и
          Регионы','Дети','Дизайн','Железо','Законы','Игры','Информационные
          Технологии','Искусство','Кино','Каталоги
          Ресурсов','Компании','Компьютеры','Литература','MP3 и
          музыка','Медицина','Наука','Недвижимость','Образование','Политика','Правительство','
          питания','Работа','Радио','Развлечения','Реклама','Религия','СМИ','Связь','Спорт','Стро
          и Услуги','Транспорт','Туризм','Фото','Электроника','Электронная
          коммерция','Юмор'];
600
    ASK SiteTheme.Psiho 'Сайт по психологии?' ['y-Да','n-Нет'];
    ASK SiteTheme.Sibir 'Сайт имеет отношение к Сибири?' ['y-Да','n-Нет'];
    ASK SiteTheme.Moscow 'Сайт имеет отношение к Москве?' ['y-Да','n-Нет'];
    ASK SiteTheme.Talk 'Сайт виртуального общения?' ['y-Да','n-Нет'];
605 ASK SiteTheme.MoscowForVisitors 'Сайт о Москве для иностранцев?' ['y-Да','n
      -Нет'];
    ASK SiteTheme.Agency 'Сайт агенства по трудоустройству?' ['y-Да','n-Нет'];
    ASK SiteTheme.Autosalon 'Сайт автомобильного салона?' ['y-Да','n-Нет'];
    ASK SiteTheme.Official 'Официальный сайт спортивной организации, федерации
      и издания?' ['y-Да','n-Нет'];
    ASK SiteTheme.Mobile 'Сайт агенства сотовой связи?' ['y-Да','n-Нет'];
610 ASK SiteTheme.Linux 'Сайт имеет отношение к ОС Linux?' ['y-Да','n-Нет'];
    ASK SiteTheme.Charity 'Сайт имеет отношение к благотворительности?' ['y
      -Да','n-Нет'];
    ASK SiteTheme.Student 'Сайт имеет отношение к студентам?' ['y-Да','n-Нет'];

    /*                                        URL с ключевыми словами
     *                                    */
615   IF SiteTheme.Main == 'y' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Technologies/';

      IF SiteTheme.Main == 'Автомобили' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Automotive/';
      IF SiteTheme.Main == 'Банки' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Banks/';
      IF SiteTheme.Main == 'Безопасность' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Security/';
620   IF SiteTheme.Main == 'Бизнес и Финансы' THEN       SiteTheme.Url =
        'http://top100.rambler.ru/top100/Business/';
      IF SiteTheme.Main == 'Города и Регионы' THEN       SiteTheme.Url =
        'http://top100.rambler.ru/top100/Towns/';
      IF SiteTheme.Main == 'Дети' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Kids/';
      IF SiteTheme.Main == 'Дизайн' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Design/';
      IF SiteTheme.Main == 'Железо' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Hardware/';
625   IF SiteTheme.Main == 'Законы' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Law/';
      IF SiteTheme.Main == 'Игры' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Games/';
      IF SiteTheme.Main == 'Информационные Технологии' THEN
        SiteTheme.Url = 'http://top100.rambler.ru/top100/Technologies/';
      IF SiteTheme.Main == 'Искусство' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Art/';
      IF SiteTheme.Main == 'Кино' THEN       SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Cinema/';
630   IF SiteTheme.Main == 'Каталоги Ресурсов' THEN       SiteTheme.Url =
        'http://top100.rambler.ru/top100/Classificators/';
```

```
IF SiteTheme.Main == 'Компании' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Companies/';
IF SiteTheme.Main == 'Компьютеры' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Computers/';
IF SiteTheme.Main == 'Литература' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Literature/';
IF SiteTheme.Main == 'MP3 и музыка' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/MP3/';
IF SiteTheme.Main == 'Медицина' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Health/';
IF SiteTheme.Main == 'Наука' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Science/';
IF SiteTheme.Main == 'Недвижимость' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Real Estate/';
IF SiteTheme.Main == 'Образование' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Education/';
IF SiteTheme.Main == 'Политика' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Politics/';
IF SiteTheme.Main == 'Правительство' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Government/';
IF SiteTheme.Main == 'Природа' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Nature/';
IF SiteTheme.Main == 'Провайдеры' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/ISPs/';
IF SiteTheme.Main == 'Программы' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Software/';
IF SiteTheme.Main == 'Продукты питания' THEN SiteTheme.Url =
  'http://top100.rambler.ru/top100/Food/';
IF SiteTheme.Main == 'Туризм' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Travel/';
IF SiteTheme.Main == 'Работа' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Job/';
IF SiteTheme.Main == 'Радио' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Radio/';
IF SiteTheme.Main == 'Развлечения' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Entertainment/';
IF SiteTheme.Main == 'Реклама' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Advertising/';
IF SiteTheme.Main == 'Религия' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Religion/';
IF SiteTheme.Main == 'СМИ' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Media/';
IF SiteTheme.Main == 'Связь' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Communications/';
IF SiteTheme.Main == 'Спорт' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Sport/';
IF SiteTheme.Main == 'Строительство' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Construction/';
IF SiteTheme.Main == 'Театр' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Theatre/';
IF SiteTheme.Main == 'Телевидение' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/TV/';
IF SiteTheme.Main == 'Товары и Услуги' THEN SiteTheme.Url = 'http:
  //top100.rambler.ru/top100/Consum/';
IF SiteTheme.Main == 'Транспорт' THEN SiteTheme.Url = 'http://
  //top100.rambler.ru/top100/Transport/';
IF SiteTheme.Main == 'Фото' THEN SiteTheme.Url = 'http://
```

```
        //top100.rambler.ru/top100/Photo/';
    IF SiteTheme.Main == 'Электроника' THEN        SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Electronics/';
    IF SiteTheme.Main == 'Электронная коммерция' THEN        SiteTheme.Url
        = 'http://top100.rambler.ru/top100/E  commerce/';
    IF SiteTheme.Main == 'Юмор' THEN      SiteTheme.Url = 'http://
        //top100.rambler.ru/top100/Humor/';

FRAME Money
{
LIST  Recommendations;
};

SET Money.Recommendations += ' Регистрация в поисковых системах: $'+
    SearchEngines.Money;
SET Money.Recommendations += ' Обмен баннерами: $' +
    BannerExchange.Money;
SET Money.Recommendations += ' Списки рассылки: $' + MailingLists.Money;
SET Money.Recommendations += ' Платная публикация информации, новости:
    $' + CommercialAdvertising.Money;

FRAME PROMOTION
{
 SCALAR MaxMoney INT;
 SCALAR RegularVisitor;
 SCALAR FirstVisitor;
 SCALAR SiteChanges;

 LIST  Recommendations;
 LIST  PromoTypes;
};


IF PROMOTION.RegularVisitor == 'y' AND PROMOTION.FirstVisitor == 'y'
    AND PROMOTION.SiteChanges == 'n' THEN
        PROMOTION.PromoTypes-$|@Optimisation, @SearchEngines, @Statistic,
            @BannerExchange, @MailingLists, @BBS, @CommercialAdvertising,
            @SiteFeatures, @OfflineMarketing, @Money|;

IF PROMOTION.RegularVisitor == 'n' AND PROMOTION.FirstVisitor == 'y'
    AND PROMOTION.SiteChanges == 'n' THEN
        PROMOTION.PromoTypes-$|@Optimisation, @SearchEngines, @Statistic,
            @BannerExchange, @MailingLists, @BBS, @CommercialAdvertising,
            @OfflineMarketing, @Money|;

IF PROMOTION.RegularVisitor == 'y' AND ( PROMOTION.FirstVisitor == 'n'
    OR PROMOTION.SiteChanges == 'y')       THEN
        PROMOTION.PromoTypes-$|@Statistic, @BannerExchange, @MailingLists,
            @BBS, @CommercialAdvertising, @SiteFeatures, @OfflineMarketing, @Money|;

IF PROMOTION.RegularVisitor == 'n' AND ( PROMOTION.FirstVisitor == 'n'
    OR PROMOTION.SiteChanges == 'y')       THEN
        PROMOTION.PromoTypes-$|@Statistic, @BannerExchange, @MailingLists,
            @BBS, @CommercialAdvertising, @OfflineMarketing, @Money|;

SET PROMOTION.Recommendations = &FLATTEN(MAP { %.Recommendations :
    PROMOTION.PromoTypes });


// Получаем основные свойства рекламной кампании
ASK PROMOTION.MaxMoney 'Какую сумму Вы планируете потратить на online
    рекламу сайта?       $' ['0','100','200','300','500','1000', '2000','5000','10000'];
ASK PROMOTION.RegularVisitor 'Вам важно, чтобы посетители регулярно
```

```
возвращались на Ваш сайт?       ' ['y=Да','n=Нет'];

// Вопросы для определения аудитории сайта
ASK PROMOTION.FirstVisitor 'Вам нужен постоянный приток новых
 пользователей?       ' ['y=Да','n=Нет'];

// Основные свойства сайта
ASK PROMOTION.SiteChanges 'Будет ли еще изменяться структура сайта (
 картина навигации на узле, имена файлов и каталогов) ?' ['y=Да','n=Нет'];

DUMP 'promo.jsw';
```

B.2.2. Исходный текст JULIA-библиотеки для извлечения ключевых слов из Интернет-ресурсов

Файл: `keyword.java`

```
import java.net.*;
import java.io.*;
import com.shwars.julia.*;

public class keyword

{
static PrintStream out = null;
static int[] TABLE_cp1251={0340,0341,0342,0343,0344,0345,0270,0346,0347,0350,
 0351,0352,0353,0354,0355,0356,0357,0360,0361,0362,0363,0364,0365,0366,0367,0370,
 0371,0372,0373,0374,0375,0376,0377,0300,0301,0302,0303,0304,0305,0250,0306,0307,
 0310,0311,0312,0313,0314,0315,0316,0317,0320,0321,0322,0323,0324,0325,0326,0327,
 0330,0331,0332,0333,0334,0335,0336,0337};
static int[] TABLE_koi8={0301,0302,0327,0307,0304,0305,0243,0326,0332,0311,0312,
 0313,0314,0315,0316,0317,0320,0322,0323,0324,0325,0306,0310,0303,0336,0333,0335,
 0337,0331,0330,0334,0300,0321,0341,0342,0367,0347,0344,0345,0263,0366,0372,0351,
 0352,0353,0354,0355,0356,0357,0360,0362,0363,0364,0365,0346,0350,0343,0376,0373,
 0375,0377,0371,0370,0374,0340,0361};
static int n_site=25; // Max number of analyzed sites
static int n_keyword=100; // Max number of analyzed words
static int n_unique_keyword=50; // Max number of resulting keywords
static int num_site=0;

 static String keyword[][];
 static int num_keyword=0;
 static String unique_keyword[];
 static int num_unique_keyword=0;

 static String words[];
 static String urls[];

/*************************************************************/

 public keyword()

 {
  String s;
  keyword = new String [n_site][n_keyword];
  unique_keyword = new String [n_unique_keyword];
  urls = new String [n_site];

  try
  {
```

```
        FileOutputStream fos = new FileOutputStream("mine.log");
        out = new PrintStream(fos);
        s = open_url();              // Take the initial page from the catalogue
        urls = get_url_line(s);              // Get URLs of top n_site popular sites

        for (int i=0; i<n_site; i++)
        {
              s = get_word_line(urls[i]); // Get the line with META KEYWORDS
              s = get_word(s);                    // Only extract the line containing
                                                  // keywords
              keyword[i] = get_array(s);          // Form array of words
        }

        for (int i=0; i<n_site; i++)
        {
            for (int j=0; j<n_keyword; j++)
            if (keyword[i][j]!=null)
                  out.println(j+": "+keyword[i][j]);

        }

        unique_keyword = check_unique(keyword);
        for (int i=0; i<num_unique_keyword; i++)
        out.println((i+1)+": "+unique_keyword[i]);

       }
       catch(Exception e)
       {
        e.printStackTrace();
       }

      }

     /*************************************************************/
     /* Get the line with sites URLs                              */
     /*************************************************************/

      public static String[] get_url_line(String url)
      {
       urls = new String [n_site];
       try
       {
        int i=0;
        int idx= 1;
        String s="";
        String s1="";

        URL U=new URL(url);
        Reader is = new InputStreamReader(U.openConnection().getInputStream());
        BufferedReader dis=new BufferedReader(is);
        boolean todo=true;
        while ((s=dis.readLine())!=null && i<n_site)
        {
         idx = s.indexOf("0&_URL=");
         if (idx>=0)
         {
          urls[i]=get_url(s);
          i++;
         }
```

```
        }
100     return urls;
       }

      catch (IOException e)
      {
105         System.out.println("Caught i/o exception");
            return urls;
      };

    }
110

    /*************************************************************/
    /* Extract URLs of popular sites                             */
    /*************************************************************/
115
     public static String get_url(String line)
     {
      String s="";
      int idx;
120   int idx2;
      idx = line.indexOf("&_URL=");
       if (idx>=0)
       {
        s=line;
125     idx=="&_URL=".length();
        s=s.substring(idx);
        s=s.substring(0,s.indexOf("\" target=_top>"));
        if (s.indexOf("%")>=0)
            s=s.substring(0,s.indexOf("%"));
130         line=line.substring((idx+s.length()));
       }
       return s;
     }

135  /*************************************************************/
     /* Extract the line containing META KEYWORDS                 */
     /*************************************************************/

     public static String get_word_line(String url)
140  {
      try
      {
       int idx  = 1;
       int idx1 = 1;
145    int idx2 = 1;
       int idx3 = 1;

       String s="";
       String s1="";
150
       URL U=new URL(url);
       URLConnection C = U.openConnection();
       C.connect();
       Reader is = new InputStreamReader(C.getInputStream());
155    BufferedReader dis=new BufferedReader(is);
       boolean todo=true;

       while ((s=dis.readLine())!=null)
       {
160     s = s.toLowerCase();
```

```
        idx = s.indexOf("keywords");
        idx2 = s.indexOf("meta");
        idx3 = s.indexOf("koi8");
165     if ((idx2>=0) && (idx3>=0))
        { s = convert(s);
          s = s.toLowerCase();
        }
170     if ((idx2>=0) && (idx>=0))
        {
        s1 += s;
        idx1 = s.indexOf("<body");
        if (idx1>=0)
175     {
          System.out.println(s);
          return s1;
        }
        }
        }
180     return s1;
      }
      catch (Exception e)
      {
185    e.printStackTrace();
       return "";
      };

    }
190

    /*****************************************************************/
    /* Get the keyword list itself                                   */
    /*****************************************************************/
195
    public static String get_word(String line)

    {
     int idx= 1;
200  int idx2= 1;
     idx = line.indexOf("content=\"");

     if (idx>=0)
      {
205    idx+="content=\"".length();
       line=line.substring(idx);
       if (line.indexOf("\"")>=0 )
                line=line.substring(0,line.indexOf("\""));
       }
210  idx= 1;
     idx = line.indexOf("content=");
     if (idx>=0)
     {
       idx+="content=\"".length();
215    line=line.substring(idx);
       if (line.indexOf(">")>=0 )
           line=line.substring(0,line.indexOf(">"));
      }

220   line=line.replace('<',' ');
      line=line.replace('>','\n');
```

```
        line=line.replace(';','\n');
        return line;
    }

    /*****************************************************************/
    /* Form an array of words separated by "," ";" " "               */
    /*****************************************************************/
    public static String[] get_array(String line)
    {
        words = new String [n_keyword];
        int i =0;
        int idx = 0;
        String s;

        idx = line.indexOf("\n");
        while ((idx>=0) && (i<n_keyword))
        {
         s=line;
         s=s.substring(0,s.indexOf("\n"));
         line=line.substring((idx+1));
         if (s.length()>1)
         {
             words[i]=s;
             i++;
         }
         idx = line.indexOf("\n");
        }
        return(words);
    }

    /*****************************************************************/
    /* Remove duplicates                                             */
    /*****************************************************************/
    public static String[] check_unique(String[][] kw)
    {
      unique_keyword = new String [n_unique_keyword];

        for (int i=0; i<n_site; i++)
         for (int j=0; j<n_keyword; j++)
          for (int k=0; k<n_site; k++)
           for (int l=0; l<n_keyword; l++)

        if ((kw[i][j]!=null) && (i!=k) && kw[i][j].equals(kw[k][l]) &&
         num_unique_keyword<n_unique_keyword)
        {

           // add to the words already found
            unique_keyword[num_unique_keyword] = kw[i][j];
            num_unique_keyword++;

           // delete the word
            for (int m=0; m<n_site; m++)
               for (int n=0; n<n_keyword; n++)
                   if ( (kw[i][j]!=null) && (i!=m || j!=n) &&
                    (kw[i][j].equals(kw[m][n])) )
                        kw[m][n]=null;
                   kw[i][j]=null;
```

```
            };
        return unique_keyword;
    }

    /*****************************************************************/
    /* translate from koi8_r to win1251                              */
    /*****************************************************************/

    public static String convert(String s)throws Exception{
      int i,j,k,c;
      String out="";
      i=0;
      while(i<s.length()){
        c=s.charAt(i);
        for(k=TABLE_koi8.length-1;k>=0;k--){
          if(TABLE_koi8[k]==c) break;
        }
        out+=((char)((c<128||k<0)?c:TABLE_cp1251[k]));
        i++;
      }
      return out;
    }


    }
```

Файл: `KeywordXtractor.java`

```
    import com.shwars.julia.*;
    import java.io.*;


    /**
     * Class Keywords
     * JULIA Function Library for Keywords Extraction
     */
    public class KeywordXtractor extends com.shwars.julia.Library
    {

     public inet(JObject J)
     {
      super(J);
      getWorld().log.warn("KWX","Loading KeywordXtractor Library");
     }

     public String getImplementedFunctionsDescriptor()
     {
      return "KEYWORD";
     }

     static int n_site=25;
     static int n_keyword=100;
     static int n_unique_keyword=50;

     static String keywords[][];
     static String unique_keyword[];

     static String words[];
     static String urls[];


     public TAny KEYWORD(TList L)
```

```
{
 String res;
 String url;
 String s;

 url=L.firstElement().asString();

 getWorld().getLog().println("Keyword extraction started");

 keywords = new String [n_site][n_keyword];
 unique_keyword = new String [n_unique_keyword];
 urls = new String [n_site];
 urls = keyword.get_url_line(url);                    // Извлекаем из строки
                                                      // URLi популярных сайтов

  for (int i=0; i<n_site; i++)
  {
        s = keyword.get_word_line(urls[i]); //Извлекаем из сайта
                                            //строку с метой keywords
        s = keyword.get_word(s);                      //Извлекаем строку
                                                      //только слов
        keywords[i] = keyword.get_array(s);           //Разбиваем строку
                                                      //слов на массив слов
  }


  for (int i=0; i<n_site; i++)
  {
      getWorld().getLog().println("Processing URL: "+urls[i]);
      for (int j=0; j<n_keyword; j++)
      if (keywords[i][j]!=null)
            getWorld().getLog().println(j+": "+keywords[i][j]);

  }


  res="";
  getWorld().getLog().println("Processing Result");
  unique_keyword = keyword.check_unique(keywords);

  for (int i=0; i<n_unique_keyword; i++)
  {
     if (unique_keyword[i]!=null)
     {
          res+=unique_keyword[i]+"\n";
          getWorld().getLog().println(i+": "+unique_keyword[i]);
     }
  }

 if (res.length()<2) return null;

 TAny T = new TScal(this, res);
 return T;
}

};
```

В.2.3. Пример диалоговой консультации с экспертной системой

В данном приложении приведена диалоговая консультация с экспертной системой, направленная на продвижение в Интернет сайта кафедры 506 МАИ.

Фрагмент консультации

Вам важно, чтобы посетители регулярно возвращались на Ваш сайт?

Да

Вам нужен постоянный приток новых пользователей?

Да

Будет ли еще изменяться структура сайта (картина навигации на узле, имена файлов и каталогов)?

Нет

Содержимое Вашего сайта часто меняется?

Нет

Хотите ли Вы найти наиболее популярные ключевые слова для Вашего сайта? Это может занять около 10 минут?

Да

Выберите раздел, к которому Вы бы отнесли Ваш сайт:

Образование

Есть ли у Вашего сайта страницы которые Вы не ходите индексировать в поисковых системах (например, страницы администратора)?

Нет

Вы уже зарегистрировали доменное имя?

Да

Содержимое Вашего сайта генерится динамически?

Нет

Какую сумму Вы планируете потратить на online рекламу сайта?

$500

Можно ли выделить в Вашем сайте отдельные тематические разделы?

Да

Готовы ли Вы регистрировать сайт в 10 крупнейших русскоязычных поисковых системах самостоятельно?

Да

Вашему сайту нужна регистрация в англоязычных системах?

Нет

Все же я рекомендую зарегистрировать сайт в 10 крупнейший зарубежных системах, которые поддерживают поиск на русском языке. Вы согласны?

Да

Сайт имеет отношение к студентам?

Да

Предлагает ли Ваш провайдер статистику посещаемости сайта?

Нет

Есть ли у Вас возможность использовать CGI скрипты?

Нет

Хотите ли Вы расположить на своем сайте систему анализа статистики и посещаемости?

Да

Вы хотите использовать русскоязычную или англоязычную систему статистики?

русскоязычную

Готовы ли Вы расположить на своем сайте "чужие"баннеры (по обмену)?

Да

Хотите ли вы участвовать в бесплатной сети обмена баннерами?

Да

Сайт рассчитан на узкий круг пользователей?

Да

У Вас на сайте есть формы для регистрации?

Да

Готовы ли Вы расположить на своем сайте счетчики участия в рейтингах?

Да

Ваш сайт представляет фирму (товар, услугу), которая существует offline?

Да

Сайт имеет отношение к благотворительности?

Нет

Укажите, какие Вы планируете траты на рекламу offline?

небольшие

У Вашей фирмы есть предметы offline рекламы (визитки, плакаты, рекламные объявления)?

Да

Полученные рекомендации:

- Зарегистрируйте Ваш сайт самостоятельно в 20 основных русскоязычных и англоязычных поисковых системах. Список Вы найдете в разделе "Поисковые системы"
- Зарегистрируйте в каталогах ресуров и поисковых системах каждую тематическую часть Вашего сайта со своим описанием.
- Зарегистрируйте Ваш сайт в 75 русскоязычных поисковых системах используя бесплатную систему регистрации http://1ps.ru
- Добавьте Ваш сайт в каталог www.students.ru
- Расположите на всех страницах Вашего сайта бесплатную систему анализа посещаемости www.spylog.ru
- Зарегистрируйтесь в бесплатной сети обмена баннерами www.linkexchange.ru. В настройках Вы сможете выбрать только сайты, близкие к Вам по тематике для показа на них Ваших баннеров.
- Разбейте процесс регистрации на несколько страниц и расположите на каждой баннер(ы) сети обмена
- Расположите у себя на сайте счетчик рейтингов www.toplist.ru
- Бесплатно разместите о себе информацию в справочнике "Желтые страницы"www.yellowpages.ru

- Везде на рекламной информации Вашей фирмы (визитки, плакаты, рекламные объявления) публикуйте адрес Вашего сайта.
- Разместите рекламу в справочнике "Желтые страницы". Детали можно узнать на сайте www.yellowpages.ru
- Регистрация в поисковых системах: $250
- Обмен баннерами: $50
- Списки рассылки: $100
- Платная публикация информации, новости: $100

Ключевые слова: реферат, курсовая, диплом, пособие, аудит, литература, маркетинг, математика, экономические, технология, экономика, рефератов, диссертация, доклад ,сочинение, статья, материал, образование, россия, Moscow, referat, москва, учебные, пособия, учебник, обучение, студенты, сервер, курсы

B.3. Система дистанционного обучения логическому программированию LPTutor

B.3.1. Фрагмент базы знаний, управляющей тестированием и навигацией

В приведенном фрагменте базы знаний описана основная иерархия фреймов (см. рис. 3.7) управляющей базы знаний и приведен фрагмент фреймов-экземпляров, описывающих одну из глав курса.

Файл: `lptutor.fdl`

```
IMPORT LIBRARY 'com.shwars.julia.Libraries.Applet';
IMPORT LIBRARY 'com.shwars.julia.Libraries.System';

OPTION COMMENTINFERENCE-OFF;

ASK mode 'Выберите способ изучения курса'
['4-Ответить на вопросы теста после прочтения всего курса',
 '2-Ответить на вопросы теста по каждой главе после прочтения этой главы',
 '3-Ответить на вопросы теста, после чего прочесть неосвоенные фрагменты',
 '1-Ответить на вопросы теста по каждой главе, после чего прочесть
  неосвоенные фрагменты главы',
 '5-Пройти тест в режиме, при котором после неверного ответа на вопрос Вы
  увидите фрагмент, содержащий ответ',
 '6-Пройти тест',
 '7-Ответить на вопросы теста после прочтения курса, а затем просмотреть
  неосвоенные фрагменты',
 '0-Ответить на вопросы теста по каждой главе после ее прочтения, после
  чего прочесть неосвоенные фрагменты главы'];

FRAME Course
{
  SCALAR Mark INT;
  SCALAR Num INT;
  LIST Chapters DEF [];
  SCALAR Show INT DEF 0;
  SCALAR Url;
  SCALAR Learn DEF '';
```

```
    SCALAR notfine INT;
};

FRAME Chapter PARENT Course
{
    SCALAR Theme;
    LIST Pages DEF [];
    SCALAR Wait;
    SCALAR Res INT;
    SCALAR Bad INT DEF 0;
};

SET Chapter.Mark=&SUM(MAP{ %.Mark : Chapter.Pages});
SET Chapter.Num=&SUM(MAP{ %.Num : Chapter.Pages});
SET Chapter.Show=&SUM(MAP{ %.Show : Chapter.Pages});
ASK Chapter.Wait 'Для показа следующей главы нажмите кнопку OK' |'
 '|;

FRAME Page PARENT Chapter
{
    LIST Questions DEF||;
};
SET Page.Mark=&SUM(MAP{%.Correct:Page.Questions});
SET Page.Show=&SUM(MAP{%.Show:Page.Questions});

FRAME Question PARENT Page
{
    SCALAR Correct DEF 0;
    SCALAR X INT;
};
IF Question.X==1 THEN Question.Correct=1;
IF (m==3 OR m==1 OR m==7 OR m==0) AND Question.Correct==0 AND
 Question.Wait=='          '
          AND &Navigate(Question.Url) THEN Question.Show=1;
IF m==5 AND &Navigate(Question.Url) THEN Question.Correct=0;

ASK Question.Wait 'Для показа следующей страницы нажмите кнопку OK' |'
 '|;


FRAME LP PARENT Course {};

//      Chapter 2
//_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ |P2_1
//_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

FRAME Q_2_1 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
    Q='что не является простым объектом данных в Прологе?';
    A='Структура';
    A='Переменная';
    A='число';
    A='Атом';
    Url='Chapter2/Ch2_p1.htm';
};

FRAME Q_2_1_1 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
    Q='что из нижеперечисленного не является атомом?';
    A='_15';
```

```
    A='\'Mary\'';
    A='...';
    A='x';
    A='book';
    Url='Chapter2/Ch2_p1_1.htm';
};

FRAME Q_2_1_3 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
    Q='что является функтором в выражении date(Day,march,2000)?';
    A='date';
    A='Day';
    A='march';
    A='2000';
    A=',';
    Url='Chapter2/Ch2_p1_3.htm';
};

FRAME Q_2_1_2 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
    Q='Каков лексический диапазон (область известности) переменных в
     Прологе?';
    A='Одно предложение';
    A='Вся пролог программа';
    A='Предложение + вызываемые из него предложения';
    Url='Chapter2/Ch2_p1_2.htm';
};

FRAME Q_2_1_4 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
    Q='Операторная запись выражения: (a b)*(4-c). Как оно будет
     представлено в виде структуры?';
    A='*( (a,b),+(4,c))';
    A='*((a-b),(4 c))';
    A='(a, ,b,*,4,+,c)';
    A='Эта операторная запись неверна';
    Url='Chapter2/Ch2_p1_4.htm';
};

FRAME P_2_1 PARENT Page {};
SET P_2_1.Theme='Синтаксис Пролога.';
SET P_2_1.Num=3;
SET P_2_1.Questions=&CUT(P_2_1.Num,&RANDOMIZE($[@Q_2_1,
 @Q_2_1_1,@Q_2_1_3,@Q_2_1_2,@Q_2_1_4]));

//_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ P2_2
//_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

FRAME Q_2_2 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
    Q='Известно, что Y=X+1, Y    конкрктизированная переменная. Выберите,
     какой запрос приведет к вычислению значения X.';
    A='X is Y 1';
    A='Y=X-1';
    A='X=Y 1';
    A='Ни один из перечисленных запросов не приведет к вычислению значения
     X';
```

```
   Url='Chapter2/Ch2_p2.htm';
};

FRAME P_2_2 PARENT Page {};
SET P_2_2.Theme='Арифметика.';
SET P_2_2.Num=1;
SET P_2_2.Questions=&CUT(P_2_2.Num,&RANDOMIZE($[@Q_2_2]));

//_____________________________ P2_5
//_____________________________

FRAME Q_2_5 PARENT Question JAVACLASS
'com.shwars.julia.JavaClassExt.JCQuestApp'
{
   Q='Какова будет результирующая конкретизация переменных при такой
    операции сопоставления:
date(D,M,1995)=date(D1,may,Y1), date(D,M,1995)=date(15,M,Y).';
   A='D=15  D1=15  M=may  Y1=1995  Y=1995';
   A='D=D1  M=may  Y1=1995  Y=M';
   A='D=15  Y=Y1';
   A='Y1=15  D1=D  Y=1995  Y=may';
   Url='Chapter2/Ch2_p5.htm';
};

FRAME P_2_5 PARENT Page {};
SET P_2_5.Theme='Второе значение операции = в Прологе.';
SET P_2_5.Num=1;
SET P_2_5.Questions=&CUT(P_2_5.Num,&RANDOMIZE($[@Q_2_5]));

FRAME Ch_2 PARENT Chapter {};
SET Ch_2.Url='Chapter2/Ch2_p1.htm';
SET Ch_2.Pages=$[@P_2_1,@P_2_2,@P_2_5];
SET Ch_2.Theme='Синтаксис Пролога. Арифметика. Сопоставление.\n';
IF m==0 AND Ch_2.Wait=='  ' AND &Navigate(Ch_2.Url) AND
 Ch_2.Mark<100 AND Ch_2.Show<100 THEN Ch_2.Res=Ch_2.Mark;
IF m==1 AND Ch_2.Mark<100 AND Ch_2.Show<100 THEN Ch_2.Res
 =Ch_2.Mark;
IF m==2 AND &Navigate(Ch_2.Url) THEN Ch_2.Res=Ch_2.Mark;
IF Ch_2.Mark<(Ch_2.Num 2) THEN Ch_2.Bad=1;
IF Ch_2.Bad>0 THEN LP.Learn+=Ch_2.Theme;
IF Ch_2.Bad==0 THEN goodchapters+=Ch_2.Theme;


IF m<3 THEN number=Ch_1.Res=Ch_2.Res+Ch_3.Res=Ch_4.Res+Ch_6.Res
 =Ch_7.Res=Ch_8.Res;
SET LP.Url='Course.htm';
SET LP.Mark=Ch_1.Mark+Ch_2.Mark+Ch_3.Mark=Ch_4.Mark+Ch_6.Mark
 =Ch_7.Mark+Ch_8.Mark;
SET LP.Show=Ch_1.Show=Ch_2.Show+Ch_3.Show+Ch_4.Show=Ch_6.Show
 =Ch_7.Show+Ch_8.Show;
SET LP.Num=Ch_1.Num=Ch_2.Num+Ch_3.Num+Ch_4.Num=Ch_6.Num
 =Ch_7.Num+Ch_8.Num;
SET LP.Learn=Ch_1.Bad=Ch_2.Bad=Ch_3.Bad+Ch_4.Bad+Ch_6.Bad
 =Ch_7.Bad=Ch_8.Bad;

IF m==3 AND LP.Mark<100 AND LP.Show<100 THEN number=LP.Mark;
IF m==4 AND &Navigate(LP.Url) THEN number=LP.Mark;
IF m==5 THEN number=LP.Mark;
IF m==6 THEN number=LP.Mark;
IF m==7 AND &Navigate(LP.Url) AND LP.Mark<100 AND LP.Show<100 THEN
 number=LP.Mark;
```

```
SET LP.notfine=Ch_1.Bad=Ch_2.Bad=Ch_3.Bad=Ch_4.Bad=Ch_6.Bad
  =Ch_7.Bad=Ch_8.Bad;
SET results='Вы ответили на '=number+' вопросов из '=LP.Num+' заданных\n';

IF number==LP.Num THEN
    goal=results+'/nи показали отличный уровень знаний';
IF LP.notfine==0 AND LP.Mark<LP.Num THEN goal=results='\nи показали
  хороший уровень знаний';
IF number<100 AND LP.notfine>0 AND goodchapters==" THEN goal=results
  ='\nВам стоит повторить весь курс: \n'+LP.Learn;
IF number<100 AND LP.notfine>0 THEN goal=results='\nВы показали хороший
  уровень знаний по разделам\n'=goodchapters='\nно Вам стоит повторить
  разделы: \n'+LP.Learn;

SET goodchapters=";

DUMP 'logic.jsw';

END.
```

B.3.2. Исходный текст Java-класса, реализующего фрейм задания вопроса

Данный класс служит основой для реализации фреймов-вопросов, и подключается к базе знаний при помощи механизма, описанного в разделе 1.5.2.1. Его основная функция взаимодействие с управляющим Java-аплетом для обеспечения интерфейса с пользователем.

Файл: `JCQuestApp.java`

```
import com.shwars.julia.*;
import java.util.Enumeration;

public class JCQuestApp extends JavaClass
{

  public TAny theQ = null;
  public TAny theX = null;

  private TList Answers = new TList(this);

  public void setA(TAny A)
  {
    Answers.add(A);
  }

  public TAny getA() { return Answers; }

  public TAny getX()
  {
    String s[]=new String[10],l;
    int i=1,n=0;

    TestApp A=(TestApp)getWorld().getExtension("APPLET");
    // Получить ссылку на аплет

    if (theX == null)
    {
      A.setQuest(theQ.asString());

      for (Enumeration e=Answers.elements(); e.hasMoreElements();)
      {
```

```
            s[n++]=((TAny)e.nextElement()).asString();
        }
        A.setAnswers(s,n);
        while (!A.getchoice());
        i=A.getAnswer(n);
        A.setchoice(false);
        theX = new TScal(this,i);
      }
      return theX;
    }
  }
```

С. Акты о внедрении

В данном приложении приведены акты о внедрении полученных в диссертации результатов:

- В лечебный процесс Государственной клинической больницы им. С.П.Боткина
- В учебный процесс кафедры Вычислительной математики и программирования МАИ
- В производственный процесс ООО "Сид-Ойл" в качестве прототипа для исследования возможности интеллектуальной диагностики и управления производственной линией рафинирования и дезодорирования растительных масел

Кроме того, в данном разделе приводятся отзыв по результатам выступления автора на спецсеминаре лаборатории искусственного интеллекта факультета компьютерных наук и информатики университета Любляны и отдела искусственного интеллекта института Йожефа Стефана под руководством профессора И. Братко, а также копия свидетельства о регистрации инструментария JULIA в Роспатенте.