Реферат

Дипломная работа содержит 84 страницы (без приложения), 15 рисунков, 1 приложение. Список использованных источников содержит 29 позиций.

Ключевые слова: Распределенные интеллектуальные системы, Искусственный интеллект, CORBA, Интернет, Компонентная архитектура, Встраиваемые экспертные системы, Удаленная консультация.

В работе дан краткий обзор существующих методов и средств построения распределенных интеллектуальных систем, включая системы удаленной консультации; обозначены современные направления развития распределенного искусственного интеллекта (Distributed Artificial Intelligence); предложен подход к построению распределенных интеллектуальных систем с возможностью распределенного коллективного вывода на основе компонентной архитектуры; рассмотрены вопросы оптимального представления знаний для такой архитектуры.

На основе предложенной архитектуры разработан и реализован инструментарий, позволяющий создавать сложные конфигурации распределенного вывода, хранения и использования знаний в интернет-подобных сетях. Инструментарий основан на архитектуре CORBA, является гибко конфигурируемым, легко расширяемым, многоязычным и многоплатформенным. Обсуждаются вопросы выбора оптимальной реализации CORBA, а также вопросы интероперабельности разработанного инструментария в гетерогенных сетях с другими системами.

Обсуждаются вопросы интеграции интеллектуальной компоненты в информационные системы. Предлагается подход к построению встроенных (embedded) экспертных систем на основе генерации высокоуровневого кода, а также набор средств интеграции в интеллектуальные системы возможностей удаленной консультации с использованием раз-

работанного инструментария. На основе этого подхода разработана и внедрена в лечебную практику Городской клинической больницы им. С.П.Боткина интеллектуальная учетно-диагностическая система больных с заболеваниями предстательной железы.

Содержание

Словарь основных понятий

CORBA (Common Object Request Broker Architecture, Общая архитектура брокера объектных заявок) — Стандартная архитектура построения сложных распределенных систем, принятая консорциумом OMG (Object Management Group). Основой интероперабельности распределенных систем в архитектуре CORBA является брокер объектных заявок, взаимодействующий с другими брокерами по протоколу сетевого взаимодействия IIOP.

Java — Многоплатформенный объектно-ориентированный язык программирования, основанный на преобразовании программы в машинно-независимый байт-код с его последующей интерпретацией на виртуальной Java-машине. Получил широкое распространение в среде интернет благодаря высокой защищенности и машинонезависимости.

Агент — Самостоятельная, как правило автономная и мобильная программная единица, предназначенная для выполнения определенной задачи. Агент функционирует в некотором окружении, и реагирует определенным образом на изменения в этом окружении.

Апплет — Самостоятельное графическое приложение, как правило на языке Java, ориентированное на работу в составе Web-страницы. Требуемые для показа страницы апплеты передаются по сети в виде байт-кода на клиентскую ЭВМ, где и выполняются.

База знаний — Определенным образом представленные знания о некоторой предметной области, предназначенные для автоматического решения задач из этой предметной области с использованием ЭВМ.

Динамические знания — Знания о предметной области, задающие правила получения новых статических знаний, т.е. перехода от одного состояния задачи к другому. Как правило составляют основу базы знаний.

Доска объявлений — Составная часть системы коллективного вывода, служащая для связи различных процессоров вывода и обмена знаниями (как правило статическими) между ними.

Интеллектуальная система — Система, обладающая функциями интеллектуальности, т.е. способная в некотором приближении решать плохо алгоритмизуемые задачи, обычно решаемые только человеком. В данной работе будем рассматривать только системы, основанные на управляемом знаниями логическом выводе.

Интернет — Глобальная всемирная компьютерная сеть, основанная на семействе протоколов TCP/IP. Говоря об интернет с маленькой буквы, подразумевают технологии, используемые в соответствующей глобальной сети.

Компонент — Некоторая функционально-законченная часть программной системы, предназначенная для работы в составе большей системы совместно с другими компонентами.

Конфликтное множество — Множество применимых на некотором шаге логического вывода правил, из которых надлежит согласно некоторому критерию выбрать одно правило для применения.

Продукции — В общем случае – способ задания грамматик, предложенный Постом, основанный на т.н. правилах переписывания. Применительно к интеллектуальным системам — способ представления динамических знаний множеством правил перехода из одного состояния системы в другое, обычно формулируемых в виде правил ЕСЛИ–ТО.

Процессор логического вывода — Основная алгоритмическая часть интеллектуальной системы, ответственная за проведение логиче-

ского вывода, т.е. получение новых статических знаний на основании динамических.

Рабочая память — Составная часть интеллектуальной системы, предназначенная для хранения статических знаний о текущем состоянии задачи.

Распределенные системы — Программные системы, состоящие из набора компонентов, выполняющихся в общем случае на различных удаленных ЭВМ в рамках компьютерной сети.

Статические знания — Знания о текущем состоянии решаемой задачи.

Удаленная консультация — Простейший способ обмена знаниями по сети, основанный на проведении консультации с экспертной системой, расположенной на удаленной ЭВМ.

Экспертная система — Интеллектуальная система, предназначенная для проведения консультаций, как правило в некоторой узкой предметной области.

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 практически сейчас большинство компьютерных систем в мире объединены в единую информационную инфраструктуру, содержащую громадное 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 средств доступа к информации в Интернет оказывается невозможным в рамках традиционного подхода к представлению этой 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-апплетов (выполняемых на компьютере-клиенте) или механизмов CGI-скриптов (выполняемых на сервере). Таким образом, в рамках Интернет становится возможным представлять не только текстовую и мультимедийную информацию, но и более сложные процедуральные структуры, иными словами двухуровневые системы, функционирующие по принципу клиент-сервер. В рамках такой технологии возможно построение достаточно сложных программных комплексов, о чем свидетельствует растущая популярность интернет-технологий в масштабах предприятия — так называемых корпоративных интранет-сетей.

Простейшие клиент-серверные архитектуры позволяют осуществлять доступ клиентов к одному централизованному ресурсу сервера. Одна-

ко 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.).

Традиционные службы Интернет, такие, как всемирная паутина, обычно используются для доступа к информациоым ресурсам. Действительно, представление информации в виде гипертекста позволяет организовать определенную семантическую связь между отдельными информационными единицами, обеспечивая возможность направленного смыслового поиска в глобальном информационном пространстве. Таким образом, помимо собственно информации, всемирная паутина содержит некоторые дополнительные данные относительно ее природы.

В этой связи интерес представляет вопрос об обмене не только информацией, но и знаниями по сети. Знания представляют собой более высокий уровень абстракции сообщений, поэтому вопрос обмена знаниями в общем случае не сводится к простой передаче сообщений. В простейшем случае, выбрав определенное представление знаний (например, в виде системы продукций) можно представить некоторое множество знаний в виде сообщений и таким образом построить передачу знаний по сети на 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 сосредотачивать различные знания в различных узлах сети, и затем комбинировать эти знания, используя их совместно для решения определенной задачи. Примером использо-

вания такого подхода может 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, или наконец интеллектуальная составляющая информационной системы поддержки деятельности виртуальной корпорации.

Проблема представления и использования знаний изучается в рамках **искусственного интеллекта**. Наиболее успешным достижением в этой области явилось создание **экспертных систем**, т.е. систем, основанных на определенном представлении знаний в узкой проблемной области и использовании этих знаний для решения конкретных задач. Типичная экспертная система содержит модель предметной области или специальным образом представленный опыт человека-эксперта (**базу знаний**), **рабочую память** для представления данных о поставленной задаче, и систему рассуждений (**механизм логического вывода**), позволяющую применять содержащиеся в системе знания для решения задачи. В настоящее время многие интеллектуальные системы в той или иной степени основываются на архитектуре экспертных систем.

Поставленная проблема построения распределенных интеллектуальных систем, обеспечивающих обмен знаниями между отдельными компонентами системы, по сути дела лежит на стыке искусственного интеллекта и современных технологий построения распределенных систем. В настоящее время можно выделить два основных направления в рамках распределенного искусственного интеллекта (Distributed Artificial Intelligence):

- Распределенная система основана на взаимодействии **автономных интеллектуальных агентов** (Autonomous Intelligent Agents), каждый из которых является мобильной независимой сущностью, способной к выполнению некоторого осмысленного интеллектуального действия. Каждый агент при этом инкапсулирует в себе все основные компоненты интеллектуальной системы, содержит свою

базу знаний и механизм вывода.

- Распределенная система строится из набора **компонентов**, каждый из которых отвечает за тот или иной аспект функционирования системы. В этом случае вся интеллектуальная система имеет традиционную архитектуру с разнесенными по разным узлам компонентами.

Агентная архитектура получила в настоящее время значительное распространение. Современные технологии построения многоплатформенных мобильных приложений с использованием языка Java позволяют достаточно легко создавать агентов с передаваемым по сети программным кодом, которые, перемещаясь в нужные узлы сети могут выполнять требуемые действия. Однако, агентная архитектура имеет определенные недостатки, среди которых:

- Агент представляет собой замкнутую самостоятельную систему с определенным представлением знаний. Обмен знаниями между агентами осуществляется на уровне внешнего представления статических знаний о конкретной решаемой задаче; обмен динамическими знаниями о предметной области обычно не производится.

- Комплексная система на базе агентной архитектуры сравнительно сложна в построении, так как приходится обеспечивать слаженную работу ряда автономных одновременно действующих агентов.

Для ряда задач, в которых успешно применяются традиционные экспертные системы (например, для задач диагностики), оказывается удобным применять компонентную архитектуру. В простейшем случае это может быть обычная экспертная система, снабженная веб-интерфейсом для удаленного доступа (распространенная оболочка экспертных систем NExpert снабжена таким веб-интерфейсом), либо же оболочка, оформленная в виде Java-апплета (система Jess — Java-вариант широко известной оболочки CLIPS). Для внедрения удаленных консультаций в состав информационных систем может использоваться CALL-интерфейс с удаленным вызовом процедур. Такой удаленный интерфейс был, в

частности, разработан на кафедре для оболочки BOW на базе механизма удаленного вызова процедур RPC. Однако во всех описанных примерах механизм вывода, база знаний и знания о решаемой задаче (т.е. по сути дела вся экспертная система) были расположены на одной ЭВМ, при этом по сети пересылаются либо статические данные о решаемой задаче, либо динамические знания о предметной области в определенном представлении.

В данной работе предлагается выделить механизм вывода, источник знаний о предметной области (базу знаний) и информацию о решаемой задаче в виде самостоятельных компонентов, которые могут функционировать на различных ЭВМ. Набор компонентов оформляется в виде инструментария, позволяющего конструировать сложные конфигурации интеллектуальных систем на базе готовых компонентов, а также расширять систему добавлением новых компонентов (например, для доступа к СУБД). При этом между компонентами возможен обмен как динамическими знаниями о предметной области, так и состоянием решаемой задачи. Комбинируя компоненты, возможно создавать интеллектуальные системы с различными конфигурациями и стратегиями вывода, в том числе легко применять различные стратегии вывода к одной базе знаний простой заменой компонента.

Работа посвящена разработке инструментария для создания распределенных продукционных экспертных систем и его платформенно-независимой реализации на языке Java, а также созданию на его основе модельной диагностической базы знаний, функционирующей в среде интернет. **Цель работы** — выработать наиболее подходящие способоы представления знаний для распределенного вывода, разработать структуру инструментария и основных компонентов, реализовать инструментарий с использованием одного из современных средств построения распределенных систем, использовать инструментарий для создания действующего прототипа диагностической экспертной системы, функционирующей в среде интернет, а также спроектировать и реализовать средства интеграции распределенной интеллектуальной компоненты в информационные системы с целью создания и внедрения информационной медицинской системы с возможностью удаленной консультации.

В то время как традиционные подходы к распределенным базам знаний основываются на использовании простой двухуровневой модели клиент-сервер, т.е. проведении логического вывода исключительно на стороне клиента или на стороне сервера, разработанный подход допускает произвольное комбинирование источников знаний и серверов логического вывода и позволяет осуществлять обмен знаниями на различных уровнях представления. В основу рассматриваемого подхода легло представление знаний о предметной области в виде продукций, а статических знаний о решаемой задаче в виде набора пар атрибут-значение, характерное для большинства классических экспертных систем (BOW [5], EMYCIN и др.). Такое сравнительно простое и компактное представление позволяет эффективно реализовать обмен знаниями по низкопроизводительной компьютерной сети, так как для проведения вывода зачастую нет необходимости передавать весь объем знаний. Однако в отличие от традиционной модели **доски объявлений**, в которой все знания о решаемой задаче хранятся централизованно, данный подход использует локальные копии рабочей памяти решаемой задачи на каждом из серверов вывода, содержащие необходимые для вывода пары атрибут-значение, в то время как центральная доска объявлений служит как средство синхронизации.

С целью упростить решение практических задач в модель представления знаний и логического вывода были внесены некоторые императивные расширения: механизм **вставок** (plugins), позволяющий легко интегрировать в базу знаний процедурные компоненты, в том числе для доступа к БД, и т.н. **неустойчивые атрибуты** (volatile attributes).

Легкость расширения инструментария достигается за счет его существенно модульной структуры, основанной на компонентной модели и использовании CORBA как основы распределенной среды. Новые компоненты могут разрабатываться практически на любой платформе и на любом языке программирования, которые поддерживают CORBA.

Таким образом, в работе присутствуют следующие новые результаты:

1) Принцип построения распределенных интеллектуальных систем, допускающий обмен знаниями на различных уровнях представле-

ния (на уровне статических или динамических знаний), дополнительные расширения классического вывода в продукционной модели с использованием неустойчивых атрибутов.

2) Структура и состав инструментария, позволяющего легко строить многоуровневые интеллектуальные системы с распределенным выводом и хранением знаний, обладающие высокой модульностью и расширяемостью, интерфейсами с процедурными языками и БД, многоплатформенностью и многоязычием.

3) Набор средств интеграции интеллектуальной компоненты в информационные системы с базами данных, включающий в себя средство генерации по базе знаний программного кода для проведения консультации, а также компонентов для проведения удаленной консультации.

4) Оригинальное программное обеспечение инструментария, библиотеки и компонентов для использования удаленной консультации.

5) Прототипы базы знаний диагностики заболеваний предстательной железы.

6) Интегрированная учетно-диагностическая система больных с заболеваниями предстательной железы, допускающая удаленную консультацию с набором распределенных баз знаний.

Разработанный инструментарий позволит достаточно просто строить распределенные системы различных конфигураций, основанные на коллективном накоплении, поддержании и использовании знаний. В частности, на базе разработанного инструментария была создана интеллектуальная учетно-диагностическая система больных заболеваниями предстательной железы, внедренная в лечебную практику центральной клинической больницы им. С.П.Боткина. Также в работе рассмотрены принципы использования системы для организации интеллектуального управляемого знаниями поиска в сети Интернет, основанного на поддержании на каждом узле контентной базы знаний и распределенного вывода в таких базах.

Результаты дипломной работы были представлены на конференции "Информатика и информационные технологии CSIT'99" (The Workshop on Computer Science and Information Technologies) [1].

По результатам создания информационной системы учета и диагностики больных с заболеваниями предстательной железы опубликована работа [2], а также создан интерактивный веб-сайт [3], предоставляющий возможность удаленной консультации по Интернет.

Данная работа помимо введения включает в себя три главы и заключение. В первой главе рассматриваются некоторые общие аспекты построения интеллектуальных систем, существующие подходы к реализации распределенных интеллектуальных систем, в том числе с использованием традиционных интернет-технологий, затрагиваются вопросы построения таких систем на базе агентной и компонентной архитектур. Также в этой главе описываются основные принципы построения распределенных интеллектуальных систем на базе набора компонентов, архитектура инструментария, основанного на этих принципах, обсуждаются вопросы оптимального представления знаний и методов вывода в компонентной среде. Вторая глава описывает особенности реализации инструментария для функционирования в среде интернет, подробнее останавливаясь на интерфейсах компонентов и основных библиотек. Аргументируется выбор конкретных программных средств для реализации инструментария, проводится краткое сравнение существующих сред для построения распределенных систем (RPC, RMI, DCOM, DCE, CORBA), описываются некоторые принципы построения таких систем, а также общие соображения по построению архитектурно-нейтральных машинно-независимых программных систем для использования в Интернет. Рассматриваются различные варианты применения описываемой системы, в том числе для медицинской диагностики, интеллектуального поиска, а также в рамках учебной системы, подробнее описывается модельная база знаний по диагностике заболеваний предстательной железы и основанная на ней система учета и диагностики больных урологическими заболеваниями. Третья глава содержит экономические соображения по применению системы на практике.

Глава 1.

Исследовательская часть

1.1. Представление и использование знаний в интеллектуальных системах

1.1.1. Направления развития искусственного интеллекта

Со времени создания вычислительных машин значительное внимание исследователей в области информатики было уделено способам человеко-машинного общения. Работа ЭВМ, способной с высокой скоростью выполнять ограниченный набор простых инструкций в определенной последовательности на первый взгляд существенно отличалась от мыслительного процесса человека, который, будучи не способен к быстрому выполнению арифметических и рутинных операций, мог, тем не менее, с совершенно гигантской скоростью выполнять такие операции, как распознавание образов и доказательство теорем. Все это наводило на мысль о различных принципах мыслительного процесса и процесса выполнения программ в ЭВМ и побуждало исследователей искать способы представления информации и алгоритмы ее обработки, схожие с устройством человеческого мозга, в надежде, что это позволит ЭВМ приблизиться к эффективному решению задач, до этого доступных только человеку.

Научное направление, занимающееся изучением мыслительного процесса человека и разработкой сходных методов в информатике получило

название **искусственный интеллект**. В рамках искусственного интеллекта сразу выделилось два подхода к моделированию работы человеческого мозга.

Один из подходов, появившийся, строго говоря, еще до первых электронно-вычислительных устройств, состоял в том, что так или иначе моделировалась функция отдельных нейронов головного мозга, и большой набор искусственных нейронов (т.н. **нейронная сеть**), будучи определенным образом, наподобие головного мозга, соединненных вместе, будет функционировать как единое целое, обеспечивая те же мыслительные функции. При этом нейроны могут моделироваться на ЭВМ или же в виде электромеханических устройств. Основным ограничением в этом случае остается необходимость использования гигантского количества нейронно-моделирующих узлов для достижения требуемого уровня интеллектуальности, что при нынешнем уровне развития технологии не представляется возможным. Однако, с использованием рассмотренного подхода был разработан ряд математических методов (например, метод группового учета аргументов [27]), позволяющих на основе нейронных сетей строить математические модели явлений по исходным модельным данным о ситуации. Нейронные сети, моделируемые программно, в настоящее время достаточно широко используются в задачах классификации, диагностики и других, для которых отсутствует алгоритм решения или необходимые экспертные знания. Более того, с ростом вычислительных мощностей современных ЭВМ в настоящее время интерес к нейронным сетям растет.

Второй подход состоит в моделировании процессов мышления на более высоком уровне абстракции. Это включает в себя изучение мыслительного процесса человека, способов хранения и доступа к информации, а также общих механизмов мышления и затем построение соответствующей программной модели, т.е. программной системы, устроенной по *схожим* принципам. Основная проблема в данном подходе состоит в том, что процессы мышления до настоящего момента изучены весьма поверхностно, кроме того, сложность этих процессов не всегда позволяет построить соответствующую программную систему. Однако, даже некоторые из принципов, положенные в основу программных систем, позво-

ляют создавать до некоторой степени "интеллектуальные" программы.

1.1.2. Архитектура интеллектуальных систем

Интеллектуальные системы — это программные системы, в той или иной степени использующие методы искусственного интеллекта для решения поставленных задач. Как правило это означает, что решаемая задача не имеет достаточно простого алгоритмического решения, а основывается на применении человеческого опыта, и программа основывается на применении методов, моделирующих мышление человека.

Первыми системами такого рода стали **экспертные системы**, которые содержали в себе определенным образом представленные знания эксперта (**базу знаний**) и программу (**логический интерпретатор**), способную делать на основании этих знаний определенные выводы о решаемой задаче. Такие системы выполняли роль человека-эксперта, и были способны давать **консультации** о проблемах из узкой предметной области. В настоящее время многие интеллектуальные системы включают в себя экспертные системы, но помимо консультационных могут выполнять и другие функции.

Архитектура простейшей экспертной системы приведена на рис. 1 [11].

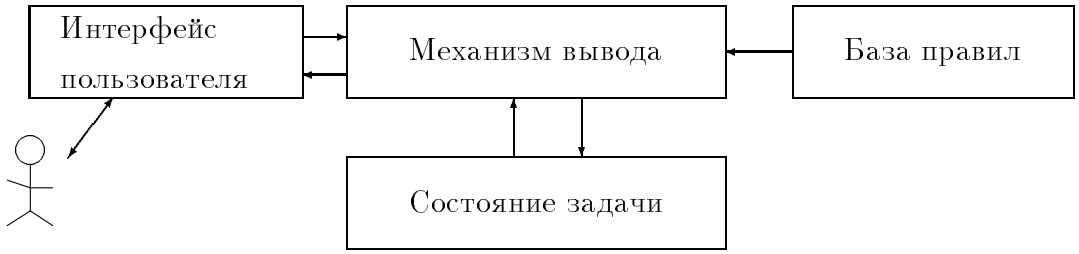

Рис. 1. Архитектура простейшей экспертной системы

Состояние задачи (или рабочее пространство, рабочая память) изначально содержит определенным образом представленные знания о рассматриваемой проблеме (т.н. **статические знания** [10]). В процессе консультации рабочее пространство пополняется новыми фактами, выведенными системой.

База знаний экспертной системы содержит определенным образом представленные знания эксперта о конкретной предметной области, для которой предназначена система (**динамические знания**). Эти знания заносятся в систему при ее создании, остаются неизменными в процессе ее эксплуатации и служат для вывода на базе фактов новых статических знаний.

Механизм вывода (интерпретатор логического вывода) является основной программной компонентой системы. Он реализует ту или иную стратегию (или их сочетание) логического вывода новых фактов в рабочей памяти на базе имеющейся информации о задаче путем применения знаний из базы знаний.

Функционирование экспертной системы, таким образом, сводится к циклическому процессу распознавания и применения определенных знаний предметной области (recognize-act cycle, [12]) до тех пор, пока не будет получен требуемый результат:

1) Выбор на основании текущего состояния задачи тех знаний из базы знаний, которые могут использоваться для вывода новых фактов (получение т.н. **конфликтного множества**).

2) Выбор на основании некоторого критерия из конфликтного множества знаний элементарного подмножества, позволяющего провести один шаг логического вывода.

3) Применение выбранного подмножества знаний и получение нового факта, т.е. изменение состояния задачи.

Заметим, что на каждом шаге производится получение одного нового факта о решаемой задаче, и применяется только часть релевантных знаний из конфликтного множества.

Более сложные системы помимо рассмотренных компонентов могут включать в себя подсистему объяснений, средства общения на естественном языке, средства приобретения знаний и т.д.

Таким образом, при проектировании конкретной экспертной системы необходимо выбрать:

- форму представления для статических и динамических знаний,

- стратегию вывода и стратегию выбора знаний из конфликтного множества.

1.1.3. Представление знаний

При разработке систем искусственного интеллекта важное место занимает проблема **представления знаний** [15]. Компьютерные системы, как известно, производят автоматизированную обработку сообщений, которая соответствует желаемой обработке информации. Однако, термин "знания" не является синонимом информации, и поэтому для оперирования знаниями необходимо построить некий аналог "правила интерпретации", позволяющий представить знания в виде сообщений и построить обработку знаний на обработке сообщений.

Относительно понимания термина "знания" может возникнуть неоднозначность, так как это понятие по-разному интерпретируется даже на интуитивном уровне, часто смешиваясь с понятием информации. Один из распространенных подходов [12] состоит в понимании знаний как абстракции сообщений более высокого уровня, чем информация. Действительно, в иерархии информация–сообщение не всякое сообщение может быть проинтерпретировано как информация; некоторая информация, переданная множеством сигналов, несет в себе определенный смысл, воспринимаемый человеком как единое целое. Точно также не всякая информация есть знание; знанием может считаться некоторая группа сведений, несущая в себе глубинную информацию о сущности явлений. В книге [12] выделяют целую иерархию понятий (см. рис. 2), связанных с термином "знания".

Рис. 2. Иерархия абстракций сообщений

Для практических целей важным вопросом является представление знаний в виде сообщений с целью дельнейшей автоматической обработки. При этом представление знаний должно не только как можно полнее передавать все аспекты знаний, но и допускать их эффективную обработку.

Все множество существующих представлений знаний можно разделить на 4 класса [8]:

Логическое представление. В этом случае для представление знаний используется хорошо исследованный математически аппарат формальной логики, как правило, первого порядка.

Процедуральное представление. Знания представляются в виде набора действий, направленных на решение задачи. Эти действия могут представлять собой алгоритм (упорядоченный набор действий) или же множество правил типа ЕСЛИ–ТО, применяемых к решению задачи в определенной последовательности согласно правилу вывода.

Сетевое представление основано на представлении знаний в виде графа с набором объектов (вершин графа) и отношений между ними (дуг). Применяя различные виды отношений и нагруженных графов возможно отражать различные характеристики предметной области: объектную иерархию, таксономию, родственные, антонимические, синонимические связи и др.

Структурное представление является объединением сетевого представления и объектного подхода, в котором объекты предметной области представляются в виде сложных структур данных (фреймов), объединенных в объектную иерархию.

Применительно к построению интеллектуальных систем необходимо выделить методы, пригодные для представления статических и динамических знаний. Из рассмотренных групп логическое представление позволяет выразить знания обоих типов, сетевое и структурное представления оказываются удобными для представления статических знаний [4], а процедуральное — для динамических знаний.

Сетевое представление, основанное в первую очередь на наглядном графическом изображении, неудобно для представления в ЭВМ. В реальных системах, как правило, применяют представление статических знаний в виде **троек объект-атрибут-значение**, представляющих собой линейную запись множества дуг графа. Такое представление позволяет также выразить отношения наследования между объектами, и, тем самым, может служить как альтернатива структурному представлению (при введении некоторых дополнительных процедуральных расширений). На рис. 3 приведен фрагмент представления знаний в виде семантической сети и соответствующие ему тройки пары объект-атрибут-значение, показывающий, в частности, отношения наследования и агрегирования (включения).

```
ТУ-154 число_пассажиров 200
Фюзеляж содержится_в самолет
Фюзеляж материал алюминий
```

Рис. 3. Пример представления знаний семантической сетью и соответствующие тройки объект-атрибут-значение

В ряде практических задач реально приходится иметь дело только с одним объектом (например, задача диагностики пациента по симптомам). В этом случае нет необходимости представлять объектную иерархию, и вместо троек объект-атрибут-значение используют **пары атрибут-значение** (Attribute-Value pairs). Представление статических знаний в виде пар используют такие оболочки экспертных систем, как

BOW [5].

Для представления динамических знаний часто используются т.н. **продукции**, в наиболее общем случае представляющие собой правила перехода из одного состояния рабочего пространства в другое [10]. Чаще всего продукции представляют собой правила, сопоставляющие определенным комбинациям значений рабочей памяти новые значения, добавляемые в рабочую память. Экспертные системы, основанные на этом принципе, получили название **продукционных**.

В подавляющем большинстве продукционных систем для представления статических знаний в рабочей памяти используются тройки объект-атрибут-значение (либо пары атрибут-значение). В этом случае продукции представляют собой простые ЕСЛИ–ТО правила, ЕСЛИ-части которых содержат некоторые конкретные комбинации атрибутов и значений в рабочей памяти (условие применимости правила), а в посылке содержатся новые атрибуты и значения, добавляемые в рабочую память. Некоторые системы (EMYCIN, см. [8, 10]) позволяют сопоставлять с каждой парой атрибут-значение и с каждым правилом некоторый фактор уверенности, и, тем самым, на основании некоторых эвристических или строго вероятностных выводов получать набор решений с соответствующими вероятностными характеристиками.

1.1.4. Методы вывода и стратегии выбора правил в продукционных системах

Как было описано в разделе 1.1.2, процесс вывода в экспертных системах представляет собой цикл по отбору и применению знаний из базы знаний для получения новых фактов в рабочей памяти. Для продукционных систем знания представляются в виде правил, следовательно в продукционных системах на каждом шаге вывода происходит поиск конфликтного множества применимых правил, выбор одного правила для "исполнения" и применение этого правила с занесением нового значения в рабочую память.

Традиционно в экспертных системах различают два различных метода логического вывода:

- прямой вывод (forward chaining, data-driven inference),
- обратный вывод (backward chaining, goal-driven inference).

При проведении **прямого вывода** исходные данные о задаче заносятся в рабочую память и формируют начальное состояние задачи. Затем на каждом шаге вывода в конфликтное множество попадают те правила, левые части (посылки) которых могут быть вычислены (т.е. данные для вычисления которых содержатся в рабочей памяти), и на основании некоторого критерия выбирается одно из правил для применения, в результате чего в рабочую память заносится новое значение. Процесс завершается, когда получено значение целевого атрибута, указываемого до начала вывода.

Обратный вывод также требует указания целевого атрибута, но исходные данные о задаче в начале вывода не заносятся в рабочую память. Процесс вывода идет от целевого атрибута, формируя конфликтное множество из тех правил, которые могут установить значение этого атрибута. При выборе некоторого правила из конфликтного множества атрибуты в его левой части по очереди делаются целевыми, и процесс вывода продолжается. Когда значение некоторого атрибута не может быть установлено применением правил, оно запрашивается у пользователя. Тем самым система обратного вывода сама запрашивает у пользователя сведения, необходимые ей для проведения рассуждения.

Системы с **комбинированным выводом** используют комбинацию двух видов вывода, при этом, как правило, после каждого шага обратного вывода применяется прямой вывод (исчерпывающий или один шаг). Правила базы знаний могут явным образом предназначаться для определенного типа вывода (демоны и продукции в [4]), либо же вывод может производиться по очереди с использованием одной базы знаний.

В процессе вывода возникает вопрос о выборе применяемого правила из конфликтного множества. При этом могут использоваться различные стратегии выбора правила, например:

- первое идущее по тексту базы знаний правило,
- правило с наибольшим приоритетом,

- правило, содержащее наибольшее количество условий в посылке,
- правило с наименьшей частотой использования.

1.1.5. Коллективный вывод. Архитектура доски объявлений

В ряде случаев желательно разделить процесс решения задачи между несколькими базами знаний и процессорами вывода. Это может делаться по соображениям эффективности (поиск правил в базе знаний меньшего объема происходит существенно быстрее), из-за желания разделить базы знаний логически, с целью распараллеливания вывода, или, наконец, с целью проведения распределенного вывода. Основной архитектурой, позволяющей проводить вывод сразу несколькими процессорами вывода для одной задачи является архитектура **доски объявлений** (blackboard architecture).

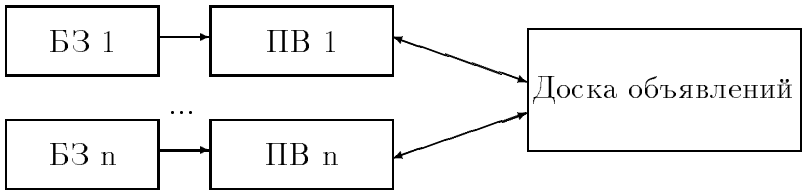

Рис. 4. Архитектура доски объявлений

В этой архитектуре (ст. рис. 4) место рабочей памяти занимает доска объявлений, являющаяся общей для нескольких процессоров вывода. Каждый процессор вывода проводит логический вывод независимо, согласно своей базе знаний, но оперируя общим статическим представлением задачи. Вывод может осуществляться различными процессорами вывода как последовательно, так и одновременно. Традиционным примером системы с архитектурой доски объявлений является система HEARSAY II [8, 14], использующей различные базы знаний и процессоры вывода для распознавания речи на различных уровнях представления фонетических элементов (фонем, лексем, слов, предложений и т.п.).

1.2. Распределенные интеллектуальные системы

1.2.1. Подходы к организации удаленных консультаций через интернет

Простейшая задача, с которой сталкиваются при построении интеллектуальных систем в среде интернет, — это обеспечение работы экспертной системы "на расстоянии", т.е. проведение удаленной консультации по сети. Можно выделить следующие преимущества удаленной консультации через Интернет:

- Удаленная консультация позволяет пользователю воспользоваться базой знаний без установки на свой компьютер дополнительного программного обеспечения.
- Разработчик интеллектуальной системы может легко менять конфигурацию и смысловое наполнение системы на сервере без переустановки программного обеспечения на компьютере пользователя.
- При удаленной консультации обеспечивается нераспространение исходной базы знаний, которая может иметь существенную ценность как интеллектуальный продукт, а только предоставление *однократной консультации*, возможно, в рамках электронной коммерции (e-commerce). В этом случае для периодического использования системы необязательно приобретать дорогостоящую базу знаний, а достаточно оплатить гораздо более дешевые однократные консультации.

В рамках современных интернет-технологий возможно два подхода к проведению удаленных консультаций:

1) **Логический вывод проводится на стороне сервера** (архитектура "тонкого клиента"). В этом случае для доступа к базе знаний применяется традиционная двухуровневая технология клиент-сервер или архитектура CGI-скриптов, а на удаленном сервере

функционирует по сути дела любая оболочка экспертных систем, адаптированная соответствующим образом (см. рис. 5). По сети передаются исходные данные о задаче (статические знания), и получается результат также в форме статических знаний об результате консультации. Данный подход может реализовываться в рамках WWW (интерактивные веб-консультации, см., например, `http://www.fnmedcenter.com/ccis/cfstest.htm`) или же независимо для использования удаленной консультации из клиентского программного обеспечения. Удаленный программный интерфейс с оболочкой экспертных систем BOW был реализован на кафедре [5].

2) **Логический вывод проводится на стороне клиента** (архитектура "толстого клиента"), при этом на клиентский компьютер передается база динамических знаний, и дальше компьютер-клиент функционирует, по сути дела, как независимая интеллектуальная система (см. рис. 6). Примерами таких систем являются система Jess (удаленный аналог известной системы CLIPS) и разработанная автором система DIET-1.

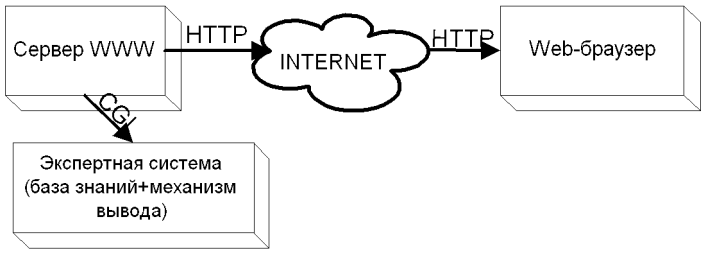

Рис. 5. Удаленная консультация в архитектуре "тонкого клиента"

Первый подход удобен тем, что не расходуются вычислительные ресурсы клиента, и объем передаваемых по сети статических знаний сравнительно мал. Во втором случае объем передаваемых по сети знаний достаточно велик и их структура более сложная, однако, он не требует специальных средств со стороны сервера и может использоваться для внедрения небольших экспертных систем в интернет-страницы в виде Java-апплетов (В качестве примера см. [3]).

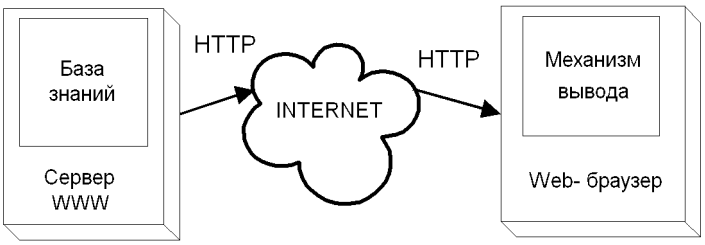

Рис. 6. Удаленная консультация в архитектуре "толстого клиента"

1.2.2. Основные архитектуры распределенных интеллектуальных систем

Рассмотренные методы позволяют лишь организовать удаленный доступ к базе знаний, но не предлагают методов совместного функционирования набора распределенных по сети знаний. Более сложные современные архитектуры в той или иной степени основываются на взаимодействии распределенных компонентов, совместная работа которых обеспечивает функционирование сложной интеллектуальной системы в целом. В зависимости от структуры компонентов можно выделить две различные архитектуры: агентную и компонентную.

1.2.2.1. Агентная архитектура

В рамках **агентной архитектуры** распределенная интеллектуальная система строится из т.н. **интеллектуальных агентов** (intelligent agents) [9]. Агент является функционально законченной системой, т.е. он способен выполнять определенную функцию. Агенты рассматриваются относительно некоторого **окружения** (environment), способны интеллектуально реагировать на изменения в этом окружении и совершать определенные действия. Как правило, агенты отличаются свойством **автономии** (autonomy), т.е. агент способен выполнять свои действия как независимо, так и совместно с другими агентами. Агенты, работающие совместно с другими для достижения некоторой цели называются **кооперирующими** (cooperative).

Система в рамках агентной архитектуры строится на взаимодействии кооперирующих агентов, при этом агенты, взаимодействуя между собой,

функционируют на различных ЭВМ. Выделяют т.н. **мобильные агенты** (mobile agents), которые реально перемещают свой код на удаленную ЭВМ.

Так как агенты представляют собой законченные интеллектуальные системы, они содержат все функциональные узлы интеллектуальной системы и обмениваются между собой знаниями в некотором внешнем представлении. Обычно логический вывод происходит внутри каждого агента по внутренней базе динамических знаний, а обмен производится только статическими знаниями. Для взаимодействия агентов предусматривается некоторое стандартное внешнее представление знаний. В настоящее время разработаны и проходят стандартизацию несколько языков, выполняющих роль универсального средства обмена знаниями (наподобие SQL как стандартного средства запросов к БД), среди которых следует отметить KIF (Knowledge Interchange Format) [18] и KQML (Knowlegde Query and Manipulation Language) [19].

1.2.2.2. Компонентная архитектура

Компонентная архитектура распределенных систем подразумевает построение единой интеллектуальной системы как набора взаимодействующих **компонентов**, каждый из которых способен функционировать на своей ЭВМ. В отличие от агентов, компоненты не обязательно способны выполнять законченную функцию и могут не обладать свойством автономии, а лишь служат "кирпичиками" для построения более крупных систем. Кроме того, компоненты не являются мобильными: их код не перемещается по сети, а функционирует на одной ЭВМ, на которой компонент был запущен.

В данной работе предлагается строить распределенную интеллектуальную систему из набора компонентов с функциями, соответствующими основным элементам традиционной экспертной системы (см. гл. 1.1.2, стр. 20). Между такими компонентами будет происходить обмен как статическими, так и динамическими знаниями, что позволит, комбинируя компоненты соответствующим образом, осуществлять обмен знаниями на различных уровнях представления.

1.3. Архитектура инструментария распределенных интеллектуальных систем

1.3.1. Основные требования к инструментарию

Данная работа посвящена созданию инструментария для построения распределенных интеллектуальных систем. При проектировании инструментария учитывались следующие особенности:

- Возможность обмена по сети как статическими, так и динамическими знаниями; возможность совместного использования различных баз знаний для консультации с объединением знаний на уровне баз знаний или на уровне рабочего пространства.
- Сравнительная легкость построения на базе инструментария различных конфигураций для обмена знаниями и осуществления распределенного вывода.
- Простота расширения инструментария и дополнения его компонентами-расширениями для выполнения специфических функций (доступ к базам данных, сбор данных, веб-интерфейс и др.)
- Функционирование инструментария в рамках Интернет по низкопроизводительным каналам связи, т.е. минимизация необходимого сетевого трафика.
- Возможность реализации компонентов на различных языках программирования и платформах (многоплатформенность и многоязычие).

1.3.2. Общая архитектура инструментария

Данные требования к инструментарию обуславливают его компонентную архитектуру, так как обмен знаниями во внутреннем представлении возможен при разделении на компоненты ядра интеллектуальной системы.

В данной работе предлагается в качестве компонентов инструментария выделить основные элементы традиционной интеллектуальной системы (см. гл. 1.1.2, стр. 20): процессор вывода, рабочую память, базу знаний.

На рис. 7 показана структура инструментария.

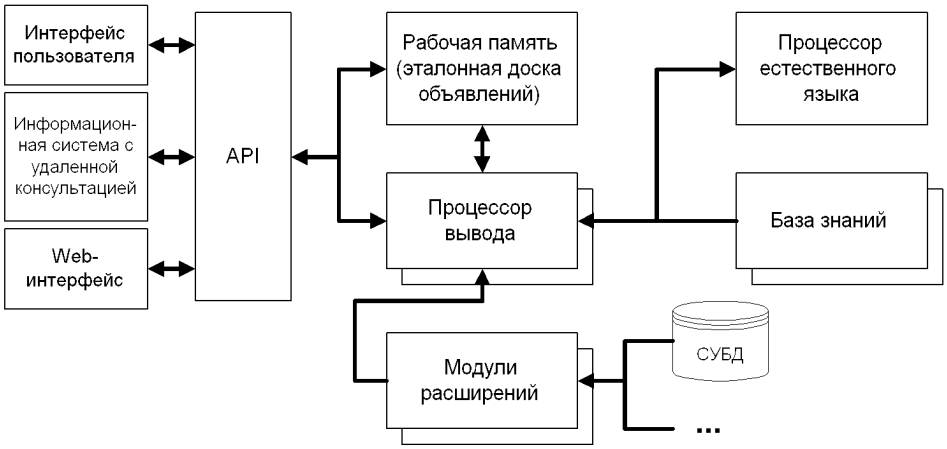

Рис. 7. Структура инструментария

Основными компонентами инструментария являются:

Рабочая память или **эталонная доска объявлений** содержит статические знания о текущем состоянии решаемой задачи. Процессоры вывода, участвующие в решении задачи, в случае необходимости консультируются с доской объявлений для получения данных о текущем состоянии.

Процессор вывода является центральным компонентом системы, производящим логический вывод. Инструментарий содержит несколько процессоров вывода для различных стратегий вывода, рассмотренных в разделе 1.1.4 (стр. 25). Процессор вывода производит вывод на основании одной или нескольких баз знаний, запрашивая у соответствующих компонентов инструментария нужные динамические знания. Для сокращения сетевого трафика процессор вывода содержит внутреннюю рабочую память для хранения состояния задачи и, по мере необходимости, запрашивает недостающие сведения у эталонной доски объявлений. В случае необходимости получения новых данных о решаемой задаче от пользователя процессор

вывода делает запрос соответствующему компоненту инструментария. В решении конкретной задачи может участвовать от одного до нескольких процессоров вывода одного или различных типов.

База знаний или источник знаний (knowlegde source) содержит некоторую базу знаний и по запросам процессора вывода предоставляет релевантные динамические знания.

Источник данных о решаемой задаче строго говоря не является отдельным компонентом инструментария, а может быть представлен либо пользовательским консультационным интерфейсом (вопросы направляются пользователю), либо библиотекой вызовов (запросы посредством CALLBACK-вызовов поступают программе, запросившей консультацию через API), либо другими компонентами (например, доступа к БД).

Описанные компоненты необходимы непосредственно для организации логического вывода. Также существуют компоненты, поддерживающие весь процесс функционирования системы или отвечающие за доступ к распределенному инструментарию со стороны прикладных программ. Среди них:

Менеджер служит для поддержания общего функционирования системы, для учета всех участвующих в выводе или доступных компонентов, процессоров вывода, баз знаний. Он также выполняет функции взаимодействия с внешними программами, обращающимися к инструментарию с целью получить удаленную консультацию, отвечает за поиск подходящего процессора вывода, если конкретный процессор не указан.

Библиотека вызовов представляет собой компонент, через который программы пользователя могут получить простой доступ к удаленной консультации (т.н. CALL-интерфейс). Библиотека выполняет роль посредника с менеджером и оформляется в виде динамической библиотеки (DLL в Windows или .so в UNIX) для универсализации доступа к ней из различных сред программирования.

Дополнительные компонеты ActiveX, Delphi, Java-апплеты и др. служат для дополнительного упрощения доступа к инструментарию из конкретных сред программирования. ActiveX-компонент и Java-апплет позволят, в частности, внедрять элементы удаленных консультаций в web-страницы.

Помимо основных компонентов для обеспечения распределенного вывода инструментарий может дополняться некоторыми дополнительными компонентами, среди которых:

Естественно-языковый интерфейс для поддержания диалога с пользователем. Компонент понимания естественного языка может использоваться для извлечения начальных данных о задаче, или для диалога с пользователем в процессе обратного вывода.

Стандартные модули расширений, предназначенные для выполнения некоторых процедурных функций, вызываемых с помощью императивных расширений языка представления знаний. К таким функциям относится, в частности, доступ к базам данных.

Web-интерфейс, выполненный как серверное приложение CGI. Совместно с ActiveX-компонентами и Java-апплетами это открывает весь спектр возможностей для доступа пользователей к удаленной консультации через интернет.

На рис. 8 показана типовая конфигурация интеллектуальной системы на базе инструментария.

Следует обратить внимание на следующие особенности:

- Для решения одной задачи могут совместно использоваться несколько процессоров вывода, при этом они могут применяться как параллельно, так и поочередно, обеспечивая комбинированный вывод.

- Один процессор вывода может использовать несколько баз знаний, комбинируя знания на динамическом уровне.

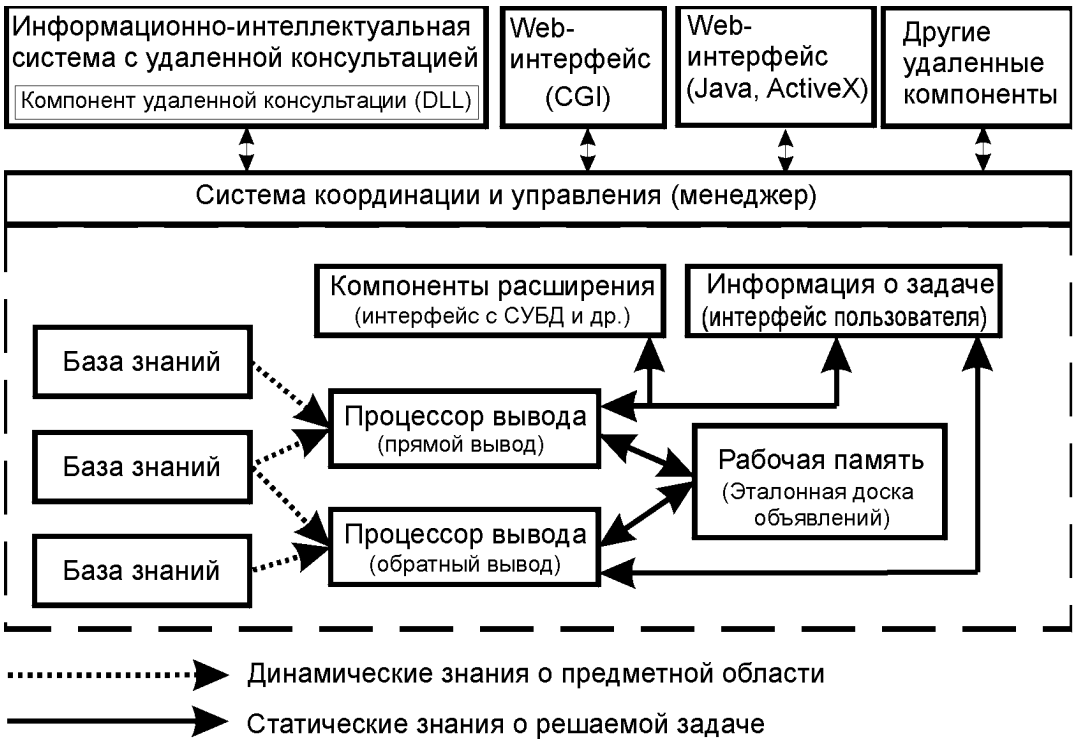

Рис. 8. Типовая конфигурация распределенной интеллектуальной системы на базе инструментария.

- Одна база знаний может использоваться несколькими процессорами вывода. В частности, для обеспечения комбинированного вывода одна база знаний используется совместно с двумя процессорами прямого и обратного вывода.
- Каждый их указанных на рисунке компонентов может функционировать на отдельной ЭВМ.

1.3.3. Выбор представления статических и динамических знаний и особенности распределенного вывода

При выборе представления знаний и методов вывода необходимо руководствоваться следующими соображениями:

- **Компактность представления знаний.** Так как статические и динамические знания могут передаваться по сети между распре-

деленными компонентами, то их представление не должно быть слишком громоздким.

- **Возможность передачи части базы знаний для проведения вывода.** Так как на каждом шаге вывода в конечном итоге применяется только одно правило, то в принципе нет необходимости передавать по сети все правила базы знаний. Однако в общем случае для выбора конфликтного множества и далее применяемого правила необходимо знать состояние задачи, т.е. располагать большим объемом статических знаний. Таким образом, желательно выбрать такое представление знаний и стратегию отбора правил, которые позволили бы отобрать минимальное множество правил для передачи процессору вывода, не располагая статическими знаниями о текущем состоянии задачи.

- **Практическая применимость и распространенность.** Тенденция упрощения представления для ускорения передачи правил может привести к практической неприменимости такой системы. Хотелось бы, чтобы система была не только успешно применима в различных предметных областях, но и основывалась на достаточно распространенной модели, чтобы облегчить перенос в распределенную среду существующих баз знаний.

С учетом этих правил для построения инструментария удобно выбрать широко распространенную продукционную модель экспертных систем, в которой динамические знания представляются продукциями, а статические — тройками объект-атрибут-значение или парами атрибут-значение. В инструментарии разумнее использовать более простое представление парами атрибут-значение, так как только такое представление позволит избежать знания всего множества правил базы знаний при выборе конфликтного множества (так как нет необходимости в сложной унификации с учетом наследования). Такое упрощение безусловно сужает круг эффективно решаемых задач, однако, во многих случаях (например, в задачах диагностики, когда мы имеем дело с одной сущностью) оно оказывается достаточным.

Для упрощения выбора конфликтного множества без информации о состоянии задачи введем следующие упрощения: потребуем, чтобы в правилах, предназначенных для обратного вывода, заключения содержали не более одного присваивания. Тогда конфликтное множество правил можно будет сформировать зная лишь целевой атрибут для текущего шага вывода.

Выбор одного правила из конфликтного множества возложим на компонент базы знаний, при этом сохраняется требование о независимости алгоритма выбора правила от состояния задачи. В текущей реализации применен простейший алгоритм выбора первого идущего по тексту правила. Возможно реализовать дополнительные компоненты баз знаний с другими стратегиями выбора правил, например, по сложности посылки.

При использовании прямого вывода для выбора конфликтного множества необходимо знание состояния задачи. Возможно два варианта распределения функций между процессором вывода и базой знаний: возложение выбора конфликтного множества на компонент базы знаний (в этом случае компонент базы знаний должен включать в себя копию доски объявлений, и возрастает обмен по сети статическими знаниями) или же на процессор вывода (в этом случае база знаний передает большее количество динамических знаний). В рассматриваемой реализации был использован второй подход, так как он позволяет унифицировать компоненты баз знаний для всех процессоров вывода не слишком усложняя компоненты баз знаний. При использовании прямого вывода рекомендуется располагать процессор прямого вывода на той же ЭВМ (или на связанной с ней скоростными каналами связи), что и компоненты баз знаний — это позволит минимизировать задержки на передачу большого количества динамических знаний.

1.3.4. Типовые конфигурации распределенного вывода на базе инструментария

Богатство возможных комбинаций распределенного вывода на базе инструментария достигается за счет разделения элементарных компонентов интеллектуальной системы и возможности их разнесения на раз-

личные ЭВМ. Рассмотрим несколько типовых конфигураций распределенного вывода с использованием инструментария:

1.3.4.1. Модель серверных вычислений (тонкого клиента)

В этом случае все основные компоненты инструментария (процессор вывода, база знаний, менеджер, рабочая память) функционируют на одной ЭВМ (сервере), клиентская машина содержит только пользовательский интерфейс, представляющий собой источник данных о решаемой задаче (рис. 9). Такая конфигурация соответствует описанной в разделе 1.

Рис. 9. Модель серверных вычислений на базе инструментария

Возможно также полностью перенести поддержку инструментария на сторону сервера с использованием интерфейса, оформленного как CGI-программа, выполняемая на сервере. В этом случае *весь* код инструментария выполняется на сервере, а клиент представляет собой стандартный Web-браузер, не требующий поддержки Java или ActiveX.

1.3.4.2. Модель клиентских вычислений (толстого клиента)

В модели клиентских вычислений (см. раздел 1.2.1) на сервере содержится только база знаний, а процессор вывода, рабочая память и источник знаний о задаче работают на клиентской ЭВМ. Возможна также ситуация, когда компонент поддержки базы знаний также функционирует на клиентской ЭВМ, но в начале работы весь текст базы знаний передается по сети на клиентский компьютер (например, по протоколу

HTTP). Такой вариант не является предпочтительным, так как сетевой трафик не минимизируется (пересылается весь текст базы знаний, в то время как при использовании специализированного компонента необходимо было бы передать только релевантные знания), однако он позволяет избежать запуска на сервере каких-либо нестандартных процессов и является единственным приемлимым (хотя и неудачным) способом использования инструментария без запуска на удаленной ЭВМ специализированных компонентов.

Рис. 10. Модель клиентских вычислений на базе инструментария

1.3.4.3. Модель вывода по распределенным базам знаний с непересекающимися доменами

Основное преимущество инструментария состоит в том, что с его помощью легко реализуются конфигурации с множественными базами знаний. При этом можно комбинировать базы знаний на уровне динамических знаний, используя один процессор вывода для проведения вывода над объединением правил из нескольких баз знаний (см. рис. 11).

Так как продукционные базы знаний с учетом принятой в инструментарии стратегии выбора правил из конфликтного множества (см. раздел 1.3.3) являются весьма чувствительными к порядку следования правил, то простое объединение баз знаний с пересекающимися множествами атрибутов (т.е. когда предметные области или домены знаний пересекаются) может привести к непредсказуемым результатам. Поэтому такая конфигурация может применяться тогда, когда различные базы знаний

Рис. 11. Вывод по распределенным базам знаний с непересекающимися доменами

отвечают за непересекающиеся домены.

В качестве примера рассмотрим гипотетическую систему полной диагностики человека. В этом случае каждый врач-специалист может создавать и поддерживать свою базу знаний о заболеваниях в его профессиональной области, и консультация по набору таких баз позволит провести общую диагностику. Конечно, все базы знаний должны быть согласованы по "внешним" именам атрибутов, обозначающих те или иные исходные данные о человеке (симптомы), и выходные диагнозы системы, т.е. иметь общую **онтологию**.

1.3.4.4. Модель распределенного вывода

В предыдущем примере вывод по распределенным базам знаний проходил централизованно. Инструментарий позволяет также проводить распределенный вывод, когда процессор вывода совместно с базой знаний функционирует на отдельной ЭВМ, при этом между ЭВМ передаются результаты вывода, т.е. статические знания о состоянии задачи (см. рис. 12).

Такой подход также может успешно применяться для баз данных с непересекающимися доменами, при этом, как правило, уменьшается сетевой трафик, и появляется возможность использовать для различных баз знаний различные процессоры вывода.

Рис. 12. Модель распределенного вывода

1.3.4.5. Модель с совпадающими доменами

Вернемся снова к примеру с комплексной диагностикой заболеваний, рассмотренному чуть раньше в разделе 1.3.4.3. Пусть теперь мы имеем несколько специалистов одного профиля, у каждого из которых имеется своя база знаний. Эти базы знаний описывают одну предметную область, и поэтому их домены знаний и множества атрибутов совпадают (для этого безусловно необходимо согласование имен атрибутов, обозначающих одно и тоже понятие). В этом случае инструментарий позволяет произвести диагностику пациента обоими базами знаний с последующим сравнением результатов. Для этого используется распределенный вывод, результаты которого поступают на специализированный компонент для сравнения результатов и выдачи комплексного решения.

1.3.4.6. Комплексный пример использования инструментария

Хотелось бы еще раз подчеркнуть, что рассмотренные конфигурации являются типовыми и отнюдь не исчерпывают возможностей инструментария. Рассмотрим еще один пример.

Пусть имеется несколько компьютерных фирм по продаже компью-

теров и комплектующих. Каждая такая фирма может использовать набор баз знаний и баз данных по имеющимся в наличии комплектующим для выбора оптимальной конфигурации и вычислению цены компьютеров по некоторым исходным требованиям заказчика (аналогично тому, как это делала известная система XCON фирмы Digital [13]). При этом для упрощения модификации базы знаний удобно разнести ее по раздельным базам знаний по каждому из типов комплектующих: процессоры, материнские платы и т.д. (см. рис. 13) Таким образом, получаем рассмотренный в разделе 1.3.4.3 случай баз знаний с непересекающимися доменами и, проводя вывод в таких базах знаний, мы сможем проводить функционально-стоимостной анализ готовых компьютерных систем.

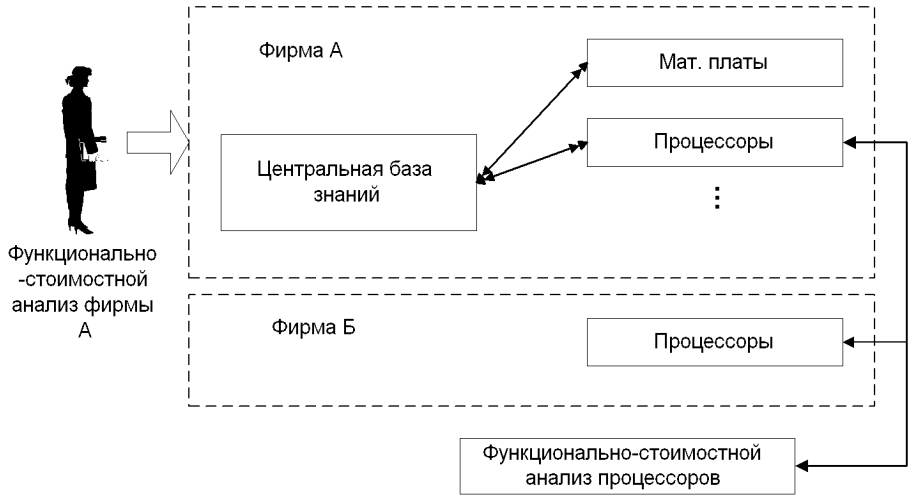

Рис. 13. Пример комплексной конфигурации

Теперь предположим, что мы хотим провести сравнительный анализ цен на процессоры в различных фирмах, использующих такой подход для выбора конфигураций. В этом случае мы можем использовать те же самые базы знаний как набор баз с совпадающими доменами и провести сравнительный анализ как это описано в разделе 1.3.4.5. Более того, мы также можем использовать весь набор баз знаний во всех фирмах для проведения автоматического сравнительного стоимостного анализа компьютеров, удовлетворяющих определенным требованиям, определенным образом объединив компоненты инструментария и используя одни и те же базы знаний. Единственное требование — это согласованность всех баз знаний по "внешним" атрибутам.

Аналогичный пример можно было бы привести и по отношению к комплексной диагностике заболеваний, как это делалось в разделе 1.3.4.3 на стр. 41, и по отношению ко многим другим предметным областям.

Глава 2.

Вычислительная часть

2.1. Выбор средств и среды для построения инструментария

Рассмотренная в предыдущей части архитектура инструментария представляет собой набор функционально законченных компонентов, которые, будучи объединены в единую конфигурацию, образуют распределенную интеллектуальную систему. Компоненты в процессе вывода взаимодействуют между собой, обмениваясь данными различной природы, быть может, по сети. Таким образом, система на базе инструментария представляет собой типичную распределенную гетерогенную систему.

2.1.1. Основные требования к инструментарию

В настоящее время существует множество сред для построения распределенных систем, поддерживающих те или иные языки программирования. От удачного выбора средств реализации инструментария будет в большой степени зависеть его практическая применимость и гибкость.

При выборе средств учитывались следующие соображения:

Многоплатформенность . Инструментарий должен быть легко переносим с одной платформы на другую, чтобы его можно было легко интегрировать в сетевые конфигурации с различными вычислительными узлами.

Открытость и расширяемость . Инструментарий должен допускать добавление новых компонентов для специализированных функций и замену основных компонентов инструментария аналогичными компонентами пользователя.

Многоязычие . Желательно обеспечить возможность реализации дополнительных компонентов инструментария на различных языках программирования. Это, во-первых, позволит интегрировать инструментарий с информационными системами, реализованными на разных языках, а также обеспечит возможность написания высокопроизводительных компонентов на компилируемом языке в сочетании с более гибкими компонентами на интерпретируемых языках.

Функционирование в среде интернет . Инструментарий должен ориентироваться на семейство протоколов TCP/IP, чтобы обеспечить его использование в среде интернет или интранет-сетей.

Простота использования средств удаленной консультации из разрабатываемых информационных систем, а также простота построения на базе инструментария различных конфигураций интеллектуальных систем.

Для достижения поставленных целей следует внимательно отнестись к выбору среды для построения инструментария, а также к выбору базового языка и системы программирования.

2.1.2. Среды построения распределенных систем

В настоящее время распределенные информационные и программные системы приобретают широкое распространение. Не удивительно, что существует достаточное количество сред для построения распределенных систем, которые обеспечивают примерно одинаковую функциональность.

2.1.2.1. Протокол RPC фирмы Sun Microsystems

Протокол удаленного вызова процедур RPC (Remote Procedure Call) является первым из предложенных протоколов для обеспечения удаленного вызова. Протокол базируется на т.н. внешнем представлении данных XDR (eXternal Data Representation), с помощью которого параметры процедур преобразуются в последовательность байт для передачи на удаленную ЭВМ, на которой собственно и выполняется процедура. Процесс преобразования параметров к внешнему представлению производится автоматически программным кодом, который генерируется специальным препроцессором по шаблону процедуры; с точки зрения пользователя процесс вызова удаленной процедуры практически прозрачен. Протокол RPC получил широкое распространение в UNIX-подобных системах, хотя существуют его реализации и для персональных ЭВМ. Протокол обеспечивает только вызов удаленных процедур и не поддерживает классовой или компонентной абстракции [26].

2.1.2.2. Распределенное окружение DCE

Распределенное окружение DCE представляет собой распределенную среду на базе UNIX-систем, включающую в себя, помимо сервисов удаленного вызова, множество других функций: поиск удаленных объектов по именам, запуск удаленного сервера для предоставления требуемых услуг и др. В UNIX-системах с графическим интерфейсом на базе DCE построено объектное окружение CDE.

2.1.2.3. Java RMI

Современные парадигмы распределенных вычислений распространили удаленный вызов процедур на объектную идеологию. Механизм удаленного вызова методов RMI (Remote Method Invocation) является встроенным в современные версии языка программирования Java. Идеология RMI в значительной степени напоминает RPC, также используется препроцессор для автоматической генерации файлов-заглушек и встроенный в язык механизм сериализации для представления передаваемых параметров [23].

Использование RMI в реальных проектах на Java крайне привлекательно, так как RMI является стандартом языка и входит во все последние его реализации. Кроме того RMI является многоплатформенным, настолько, насколько это касается Java вообще. Однако, из-за ограничения на язык программирования спектр применимости системы на основе RMI в реальных информационных системах резко сужается.

2.1.2.4. Распределенная компонентная модель DCOM Microsoft

Многие современные продукты в среде Microsoft Windows реализуются с использованием компонентной модели объектов COM (Component Object Model). В этой модели программная система строится из компонентов, под которыми понимается часть программного кода, обращение к которой производится через определенный интерфейс (набор процедур). COM-компоненты являются **повторно используемыми** (reusable) в различных приложениях и, по сути дела, представляют собой функционально-законченные единицы, из которых строится приложение [25].

DCOM (Распределенный COM, Distributed COM) предоставляет возможность строить распределенные приложения путем запуска компонентов приложения на различных ЭВМ. Средства поддержки DCOM изначально содержатся в Microsoft Windows, таким образом эта модель оказывается идеальной для построения новых и распределения старых Windows-приложений. Однако на других платформах DCOM не получил широкого распространения.

2.1.2.5. Архитектура распределенных систем CORBA

Общая архитектура брокеров объектных запросов CORBA (Common Object Request Broker Architecture) представляет собой еще одну альтернативную DCOM компонентную модель построения распределенных систем. CORBA — это попытка стандартизовать методы построения больших распределенных систем, проводимая консорциумом OMG (Object Management Group). OMG состоит из более чем 250 крупнейших компаний и корпораций в основном в области разработки программного обеспечения, за исключением Microsoft. Стандарт CORBA включает в себя

не только компонентную модель для непосредственного взаимодействия частей распределенных систем, но и спецификацию набора стандартных компонентов (CORBA Services, CORBA Facilities), облегчающих построение таких систем. Эти компоненты включают в себя, например, Naming Service, позволяющий производить поиск компонентов в распределенной среде по их многоуровневому имени, Event Service, позволяющий организовать асинхронную доставку сообщений между компонентами и другие сервисы.

Распределенные системы в архитектуре CORBA строятся из компонентов, обращение к которым возможно только через набор функций, называемый **интерфейсом**. Интерфейс описывается на универсальном **языке описания интерфейсов IDL** (Intefrace Definition Language), который является общим для всех конкретных языковых реализаций CORBA. Спецификация CORBA включает в себя т.н. **отображения** (Language Mappings) IDL на конкретные языки программирования, согласно которым описание интерфейса на IDL преобразуется к набору классов, объектов или модулей на целевом языке программирования.

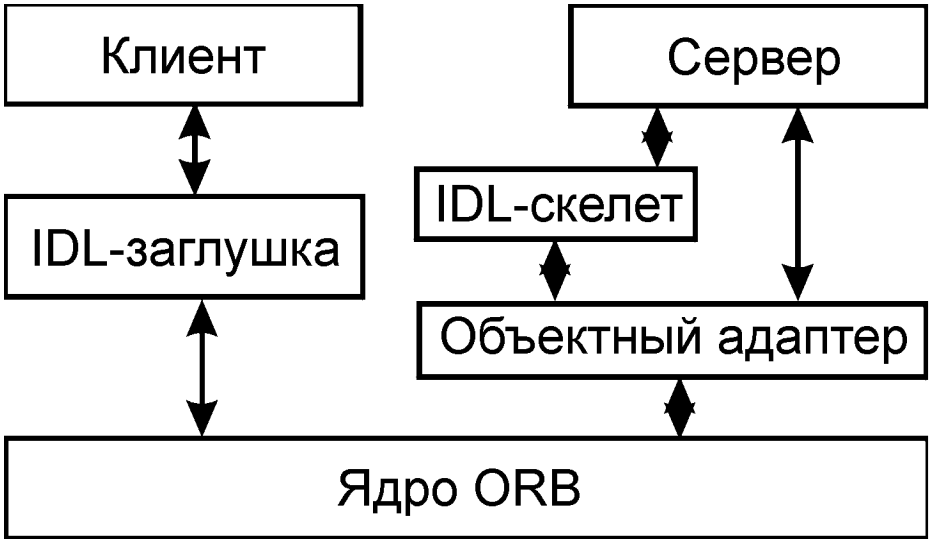

Рис. 14. Взаимодействие клиента и сервера в архитектуре CORBA при статическом вызове.

Взаимодействие объекта-клиента и объекта-сервера происходит через **брокер объектных запросов** (ORB), который является основным компонентом в архитектуре CORBA и обеспечивает взаимодействие и совместную работу остальных систем (см. рис. 14) [20]. Клиент-

серверные компоненты пользователя могут взаимодействовать через ORB напрямую, однако, в большинстве случаев, для упрощения задачи программирования используются автоматически генерируемые по описанию интерфейса на IDL **заглушка** и **скелет**.

В объектно-ориентированном языке скелет обычно представляет собой класс, от которого наследуется класс-сервер. Наследуемый класс должен переопределять все необходимые функции, описываемые в IDL-спецификации компонента, таким образом непосредственно задавая функциональность сервера. В языках без множественного наследования при необходимости включения основного класса-сервера в иерархию классов может использоваться механизм аггрегирования (т.н. tie-механизм) для взаимодействия со скелетом.

Заглушка представляет собой класс, используемый клиентом в качестве экземпляра удаленного компонента. При необходимости обратиться к удаленному компоненту пользователь прозрачным образом обращается к соответствующим методам экземпляра заглушки, которая, взаимодействуя с ORB, автоматически осуществляет удаленный вызов. Таким образом, для реализации простейшего механизма удаленного вызова необходимо специфицировать интерфейс компоненты-сервера на IDL, сгенерировать заглушку и скелет, реализовать класс-сервер, унаследованный от скелета, и далее использовать в программе-клиенте экземпляр заглушки для доступа к удаленным методам. Помимо данного достаточно простого механизма CORBA поддерживает также **динамическое связывание** с использованием **репозитария интерфейсов**, однако, данный подход не используется при построении инструментария и здесь не рассматривается.

Все компоненты CORBA (называемые также CORBA-объектами) функционируют как бы в едином пространстве, объединенном средствами ORB. Для общения по сети используется **протокол общения брокеров объектных заявок** GIOP (Generic Inter-Orb Protocol) и его разновидность для TCP/IP сетей IIOP (Internet Inter-Orb Protocol), который обеспечивает интероперабельность брокеров от разных производителей [21]. В данном пространстве каждый CORBA-объект идентифицируется **объектной ссылкой** (Object Reference) — уникальным иден-

тификатором, позволяющим сослаться на конкретный объект с целью удаленного вызова.

Существенными преимуществами CORBA по сравнению с другими моделями, обусловившими выбор CORBA как основной модели для реализации инструментария, являются следующие:

- CORBA является полноценным стандартом построения распределенных систем, в отличие от DCOM, который является стандартом де факто.
- CORBA является многоплатформенной и многоязычной средой. Существуют реализации CORBA для подавляющего большинства современных платформ, систем и языков программирования.
- CORBA широко распространена. Поддержка CORBA интегрирована в такие средства разработки, как Borland Delphi 4.0, Java Development Kit 1.2 и др. Существуют реализации CORBA в рамках GNU-проекта (mico, ORBIt) [24], а также коммерческие реализации для Borland C++, Microsoft C++ и Java (ORBIX, VisiBroker). Поддержка CORBA включена в состав среды Java для Netscape Communicator 4, что позволит легко использовать CORBA-апплеты через интернет.

Таким образом, использование CORBA для реализации инструментария позволит достичь многоплатформенности, многоязычия в написании дополнительных компонентов, а также позволит компонентам инструментария функционировать через интернет по протоколу IIOP.

2.1.3. Выбор языка программирования

CORBA поддерживает отображение в большинство распространенных языком программирования, включая C++, Java и Object Pascal (отображение реализовано в Delphi 4.0 но не стандартизовано). Кроме того, существуют реализации CORBA для таких языков, как Perl и Eiffel. Поэтому после разработки интерфейсов компонентов на IDL сами компоненты можно реализовывать на любом из поддерживаемых CORBA языках программирования.

Для реализации большинства компонентов инструментария в рамках дипломного проекта был выбран язык Java по следующим причинам:

- Java является многоплатформенным языком
- В Java имеются стандартные библиотеки языка, существенно облегчающие программирование
- CORBA является встроенной в стандарт языка начиная с версии 1.2
- Система программирования Java является бесплатно-распространяемой

2.1.4. Выбор реализации CORBA

Для языка Java имеется несколько реализаций CORBA, среди которых следует отметить следующие:

JavaIDL является стандартом библиотеки языка Java. Данная реализация распространяется вместе с JDK 1.2 и в настоящее время (версия 1.2 beta 3) реализует достаточно полное подмножество стандарта CORBA без динамического вызова и репозитария интерфейсов и без поддержки стабильных persistent-объектов. В поставку входит сервис имен Naming Service. Именно эта реализация была выбрана в качестве основной для реализации инструментария из-за своей свободной распространяемости и стандартности.

VisiBroker фирмы Inprise. Данная система является чрезвычайно распространенной реализацией CORBA, которая включена в поставку Java runtime Netscape Communicatior 4. По возможностям богаче JavaIDL, включает больше сервисов. Однако есть высокая вероятность, что в рамках Интернет VisiBroker будет вытеснен более стандартной JavaIDL, которая стремительно набирает новые возможности. Данная реализация распространяется на коммерческой основе. Разновидность VisiBroker входит в состав Delphi 4.0.

OrbixWeb фирмы IONA Technologies. Данная реализация является одной из первых реализаций CORBA, однако, в настоящее время она

вытеснена другими коммерческими системами. Данная реализация распространяется на коммерческой основе с предоставлением 30-дневной демо-версии.

Другие реализации CORBA включают в себя несколько систем, распространяемых по лицензии GNU General Public Licence или им подобным. Из-за низкой распространенности этих систем и некоторых упрощений реализации их использование оказалось непредпочтительным.

В рамках процесса выбора реализации CORBA были проведены тесты на интероперабельность различных CORBA-реализаций. Оказалось, что системы JavaIDL и VisiBroker хорошо взаимодействуют друг с другом, что позволило использовать стандартную для Java реализацию JavaIDL совместно с широко распространенной для C++ реализацией VisiBroker для построения инструментария.

2.2. Реализация компонентов инструментария

В разделе 1.3.2 (стр. 33) были описаны основные компоненты инструментария, позволяющие легко строить на своей основе комплексные конфигурации логического вывода. Для реализации инструментария на основе CORBA прежде всего необходимо специфицировать интерфейсы каждого из компонентов на IDL, а также определить их функциональность. Такая спецификация в целом полностью определит структуру и возможности системы и будет представлять собой (совместно с общими соображениями по компонентной структуре инструментария, описанными в первой части) основную ценность данной работы. Далее останется реализовать соответствующие компоненты на любом поддерживаемом CORBA языке программирования, что является более простой задачей, так как отдельные компоненты представляют собой достаточно узкоспециализированные программные единицы сравнительно низкого уровня детализации.

2.2.1. Взаимодействие основных компонентов инструментария

Как было отмечено в разделе 1.3.2, основными компонентами инструментария являются процессор вывода, база знаний и эталонная доска объявлений.

Основная работа по проведению рассуждений возложена на процессор вывода. По мере необходимости процессор вывода запрашивает правила у базы знаний и информацию о задаче у рабочей памяти. При этом в выбранной модели представления знаний для проведения шага вывода нет необходимости знать все релевантные правила, а достаточно выбрать первое применимое правило из конфликтного множества.

С учетом вышесказанного взаимодействие процессора вывода с базой знаний строится на основе **транзакций**. В рамках одной транзакции процессор вывода запрашивает (при помощи специальной процедуры открытия транзакции) у базы знаний правила из конфликтного множества и затем получает их по-очереди до тех пор, пока одно из правил не будет успешно применено. После этого процессору вывода нет необходимости получать остальные правила, и транзакция завершается.

Для простой модели представления статических знаний в виде пар атрибут-значение для выделения конфликтного множества достаточно указать целевой атрибут. Таким образом, при открытии транзакции указывается целевой атрибут, и в конфликтное множество попадают все правила, правая часть которых содержит этот атрибут. Порядок передачи правил из конфликтного множества определяет приоритет их рассмотрения в процессе вывода. Таким образом, база знаний может реализовывать различные стратегии приоритетности правил: в порядке следования в исходном тексте базы знаний, в порядке веса правил по некоторому критерию и т.д.

Специфика обратного вывода такова, что в ходе рассмотрения применимости правила возникает необходимость получать значения других атрибутов, что в свою очередь может потребовать дальнейшего вывода и инициирования новых транзакций. Таким образом база знаний должна поддерживать вложенные транзакции, при этом храня для каждой транзакции соответствующее конфликтное множество (или ссылку на

него) и текущее запрошенное правило.

Множество значений всех атрибутов для решаемой задачи хранится в эталонной доске объявлений. При необходимости установить значение некоторого атрибута (текущей цели вывода) процессор вывода запрашивает эталонную доску объявлений. Если значение там присутствует — оно передается процессору вывода и запоминается в локальной кэш-копии рабочей памяти. Таким образом, в дальнейшем если значение атрибута потребуется для других шагов вывода дополнительного обращения к эталонной доске объявлений не потребуется. Если же значение не найдено, то процессор вывода производит необходимые шаги логического вывода, и, если значение удается установить, запоминает его в локальной рабочей памяти и заносит на эталонную доску объявлений. Таким образом, основные функции эталонной доски объявлений — поддерживать множество пар атрибут-значение и по запросу выдавать или устанавливать значения атрибутов.

Эталонная доска объявлений может также служить для отработки **глобальных ограничений** (Global Constraints). Глобальные ограничения — это общие для всей задачи ограничения на значения атрибутов и их комбинаций. Например, фасетное ограничение может ограничивать диапазон изменения того или иного атрибута. Такие ограничения заносятся на доску объявлений и проверяются каждый раз при добавлении нового значения. Если ограничения не выполняются — операция добавления значения оказывается неуспешной, и процессор вывода переходит к поиску альтернативных решений.

Процесс решения конкретной задачи требует создания определенной конфигурации компонентов и некоторых специальных действий. Для координации всех функций и обеспечения упрощенного интерфейса к инструментарию со стороны внешних CORBA-объектов служит компонент, называемый **менеджером**.

Далее мы подробнее рассмотрим интерфейсы перечисленных компонентов и других компонентов инструментария.

2.2.2. Фабрики компонентов

Для запуска на определенном компьютере того или иного компонента инструментария необходимо, чтобы на этом компьютере функционировал некоторый процесс-сервер. Таким образом, необходим механизм, который будет создавать соответствующий CORBA-объект и передавать ссылку на него клиенту.

Некоторые реализации CORBA допускают создание т.н. **постоянных** (persistent) объектов, которые автоматически запускаются при указании требуемой ссылки. Однако даже в этой ситуации возникает проблема, когда сразу несколько клиентов пытаются получить доступ к одному CORBA-объекту.

Для решения этой проблемы для каждого компонента инструментария создается парный **компонент-фабрика** — компонент, по запросу к которому на сервере запускается копия требуемого компонента, и ссылка на нее передается клиенту. Таким образом, для каждого клиента создается своя копия объекта-сервера, которая освобождается после использования. На сервере все время запущены фабрики доступных компонентов, ссылки на которые регистрируются при запуске сервера со специальным компонентом — менеджером.

Для уникальной идентификации компонентам при запуске дается имя. Например, в некоторой конфигурации на одном или различных компьютерах могут одновременно запускаться несколько баз знаний – в этом случае каждой из них дается уникальное мнемоническое имя.

2.2.3. Основные структуры данных инструментария

В процессе вывода компоненты инструментария обмениваются статическими и динамическими правилами, а также другой информацией. Для этого соответствующие структуры данных описываются на IDL и затем участвуют в интерфейсах компонентов как параметры или возвращаемые результаты.

2.2.3.1. Структура статических знаний

Представление статических данных основано на множестве пар атрибут-значение:

```
typedef sequence<String> StringSeq;

struct AVPair
{
 String A;
 String V;
};

struct Comparison : AVPair
{
 long CompareType;
};

typedef sequence<AVPair> AVPairSeq;
typedef sequence<Comparison> ComparisonSeq;
```

Таким образом **AVPair** представляет собой одну пару атрибут-значение, **AVPairSeq** — целый набор пар, описывающих состояние задачи или его подмножество. Аналогично **Comparison** — пара атрибут-значение с оператором сравнения, задающие условие, **ComparisonSeq** — набор таких условий.

2.2.3.2. Структура динамических знаний

Динамические знания задаются набором продукционных правил следующей структуры:

```
struct Rule
{
 long RuleType;
 ComparisonSeq IF;
 AVPairSeq THEN;
};
```

Левая часть правила (IF-часть) представляет собой набор сравнений (условие применимости), правая — в общем случае набор присваиваний (новых пар атрибут-значение). Тип правила указывает на его применимость только в прямом выводе (т.н. демоны [4]), только в обратном выводе, или в обоих типах вывода (комбинированные правила).

Данные структуры данных не являются CORBA-объектами, так как они представляют собой данные, передаваемые между компонентами инструментария. При генерации Java-кода с помощью IDL-препроцессора этим структурам соответствуют некоторые Java-классы. Для их эффективного использования в программе создаются классы-надстройки, включающие в себя помимо полей данных функции их обработки, печати и т.д.

2.2.4. База знаний (Источник правил)

Основное предназначение базы знаний — предоставлять по запросу процессора вывода необходимые правила из конфликтного множества. В разделе 2.2.1 была подробнее описана транзакционная схема взаимодействия процессора вывода и источника правил.

Основные функции, отвечающие за предоставление знаний и обработку транзакций:

```
TransactionID StartTransaction(in String goal);
Rule NextRule(in TransactionID TD);
boolean HasMoreRules(in TransactionID TD);
void EndTransaction(in TransactionID TD);
```

Следующие функции носят служебный характер и предназначены для получения списка атрибутов и для определения требуемого типа процессора вывода (прямой, обратный, комбинированный) соответственно:

```
StringSeq GetDirectory();
long GetInfType();
```

Наконец фабрика баз знаний описывается следующим интерфейсом:

```
interface RuleProviderFactory
{
 RuleProvider Create(in String name);
};
```

Данный интерфейс описывает общие функции для всех источников правил. Конкретные реализации источников правил могут различаться внутренним представлением правил для хранения (в памяти либо в базе данных, объектной или реляционной), способом занесения правил (из текстового файла, с помощью графического интерфейса и др.) и прочими деталями реализации. В рамках инструментария реализован простой источник правил, считывающий исходную базу знаний из текстового файла (**FileRuleProvider**) и источник правил, оперирующий с базой правил из объектной базы данных на базе ObjectStore PSE (**PSERuleProvider**).

2.2.4.1. Источник данных о задаче. Интерфейс пользователя.

При консультации с использованием обратного вывода процессор вывода сам запрашивает значения требуемых атрибутов. Для предоставления в ходе консульнации этих данных о задаче служит рассматриваемый компонент. С задачей обычно связывается один источник знаний, но в принципе возможно объединение нескольких таких компонентов через т.н. Fork-компонент (см. раздел 2.2.8).

Интерфейс источника знаний содержит одну функцию для получения значения атрибута по его имени и описанию:

```
interface InfoProvider
{
 String Get(in String attrib, in String text,
            in StringSeq choices);
};
```

При вызове функции может также предоставляться набор значений для выбора.

В интерактивной консультации в роли источника данных о задаче выступает интерфейс пользователя, хотя в общем случае в роли источника данных может выступать, например, база данных или агент извлечения данных из web-страниц. Для интерактивных консультаций понятие источника данных расширяется унаследованным от него компонентом пользовательского интерфейса, который, помимо указанной функции, предоставляет функции печати текущих сообщений о процессе вывода:

```
interface UserInterface : InfoProvider
{
 void Print(in String s);
 String Read();
};
```

Как правило, все компоненты инструментария могут работать с любым из этих компонентов.

Источник знаний о задаче или пользовательский интерфейс не имеют соответствующих фабрик, так как они, как правило, создаются как CORBA-объекты в приложении, использующем инструментарий, или в библиотеке. Ссылка на один из этих объектов передается процессору вывода или менеджеру в начале консультации.

2.2.4.2. Эталонная доска объявлений. Контекст задачи.

Статические данные о решаемой задаче поддерживаются на эталонной доске объявлений, к которой процессоры вывода обращаются для получения значений атрибутов, отсутствующих в локальной копии состояния задачи. Таким образом, основная функция эталонной доски объявлений — поддержка таблицы атрибут-значение:

```
interface ProblemState
{
 exception ValueAlreadySet {} ;
 String Get(in String attrib);
 boolean Set(in String attrib, in String value)
   raises (ValueAlreadySet);
```

```
};
```

Также, как было отмечено в разделе 2.2.1, на эталонную доску объявлений возлагаются функции проверки глобальных ограничений. Для этого предусматриваются дополнительные функции установки ограничений:

```
void SetFacetConstraint(in String attribute,
                        in String L, in String R);
void SetSetConstraint(in String attribute, in StringSeq set);
void SetAndConstraing(in ComparisonSeq cs);
```

Для решения одной конкретной задачи могут использоваться несколько баз знаний и процессоров вывода, но всегда создается одна эталонная доска объявлений. Поэтому удобно сгруппировать с эталонной доской объявлений другие данные, уникальные для решаемой задачи, такие, как используемый в выводе источник знаний, источник данных о решаемой задаче или интерфейс пользователя. Такую структуру данных будем называть **контекстом задачи**. Контекст задачи представляет собой наследуемый от эталонной доски объявлений компонент, который полностью идентифицирует задачу, включая ее текущее состояние, используемый для диалога с пользователем интерфейс и т.д.

```
interface ProblemContext : ProblemState
{
 void SetRuleProvider(in RuleProvider RP);
 void SetInfoProvider(in InfoProvider IP);
 void SetUserInterface(in UserInterface UI);
 RuleProvider GetRuleProvider();
 InfoProvider GetInfoProvider();
 UserInterface GetUserInterface();
};
```

2.2.5. Процессоры вывода

Процессор вывода представляет собой основной компонент, управляющий процессом логического вывода в распределенной среде. Функция **SetEnvironment** служит для установления контекста задачи (т.е.

состояния задачи, источников статических и динамических знаний), а функция **Get** — собственно для получения значения целевого атрибута **goal** путем логического вывода. Функция **SetSupplementaryEngine** служит для установки вспомогательного процессора вывода при комбинированном выводе, функция **SetOptions** — для установки различных опций (например, одношагового или полного прямого вывода).

```
interface InferenceEngine
{
 String Get(in String goal);
 void SetEnvironment(in ProblemContext PC);
 void SetSupplementaryEngine(in InferenceEngine ie);
 void SetOptions(in long options);
 long GetInfType();
};
```

Инструментарий содержит два основных процессора вывода:

Процессор прямого вывода (FRIE — Forward Reasoning Inference Engine).

Процессор обратного вывода (BRIE — Backward Reasoning Inference Engine).

Распределенный вывод возможен в трех вариантах:

Комбинированный вывод. В этом случае BRIE используется как основной процессор вывода, а FRIE — как вспомогательный (ссылка на него устанавливается вызовом **SetSupplementaryEngine**. В этом случае после каждого шага основного процессора вывода вызывается вспомогательный процессор (прямого) вывода. Такая комбинация процессоров позволяет легко использовать комбинированный вывод с полным или одношаговым прямым выводом в зависимости от установленных опций.

Распределенный вывод, управляемый базой знаний. База знаний может сожержать правила, содержащие в своем составе указания

на продолжение логического вывода на другом процессоре вывода или другой базе знаний. Такие атрибуты, как правило, имеют форму универсального описателя конфигурации вывода (т.н. UICD – Unified Inference Configuration Descriptor) — строки, задающей требуемую для вывода конфигурацию компонентов (см. также раздел 2.2.7). В этом случае работа текущего процессора вывода приостанавливается, управление передается указанному процессору вывода, и после получения значения требуемого атрибута возобновляется прерванный вывод. При переходе на новый процессор вывода последнему передается контекст задачи, таким образом, он оперирует с тем же пользовательским интерфейсом и с той же рабочей памятью, что и исходный процессор.

Явно заданный параллельный вывод. В этом случае в правиле явно указывается, что порядок вывода правых частей не существенен, и они могут выводиться параллельно на разных явно указанных процессорах вывода. Такой подход очень удобен для случая с непересекающимися доменами (см. раздел 1.3.4.3) или с совпадающими доменами (см. раздел 1.3.4.5) и позволяет сэкономить время за счет параллельного вывода на различных удаленных ЭВМ.

Одним из предложенных расширений процесса вывода является введение т.н. **неустойчивых атрибутов** (volatile attributes). При необходимости выяснения значения некоторого обычного атрибута процессор вывода сначала анализирует наличие значения этого атрибута в рабочей памяти, а затем, в случае неудачи, инициирует логический вывод с использованием динамических знаний. Для неустойчивых атрибутов первоначального поиска значения в рабочей памяти не происходит, а всегда инициируется логический вывод. Такой подход, например, позволяет накапливать в некотором атрибуте список значений: сделав атрибут неустойчивым, соответствующие правила будут при срабатывании условий дописывать следующее значение в список, например (здесь атрибут `_диагноз` является неустойчивым):

```
IF <условие 1> THEN _диагноз = _диагноз + "воспаление легких";
IF <условие 2> THEN _диагноз = _диагноз + "простуда";
```

. . . .

В традиционных продукционных системах для составления списка значений в зависимости от комбинаций правил приходится прибегать к более сложным приемам.

2.2.6. Пользовательские расширения

Для обеспечения расширяемости инструментария в процессоры вывода введена возможность вызова т.н. **пользовательских расширений** (plugins). Эти расширения имеют общий интерфейс и вызываются из текста базы знаний с помощью конструкций, похожих на правила и позволяющих передавать расширениям параметры в различных формах представления. Имеются два типа расширений:

Универсальные расширения, которые представляют собой полноценные CORBA-объекты с интерфейсом **UserPlugin**, которые являются таким образом языково-независимыми, многоплатформенными и допускают распределенный вызов по сети.

```
interface UserPlugin
{
 String Execute(in Rule R);
}
```

Реализационно-зависимые расширения зависят от конкретной реализации инструментария и в случае с предлагаемой реализацией на языке Java представляют собой Java-классы, включаемые в состав инструментария (**стандартные расширения**) или реализуемые пользователем. Все такие расширения реализуют Java-интерфейс **com.shwars.diet.Plugins.UserPlugin**. Стандартные расширения такого типа позволяют в ходе консультации менять параметры системы, печатать сообщения и значения атрибутов и т.д. В частности, в качестве стандартного выступает расширение для доступа к базам данных, доступных через интерфейс JDBC.

2.2.7. Менеджер

Менеджер предназначен для общей координации процесса распределенного вывода. В его функции входит:

- **Регистрировать основные компоненты инструментария** (процессоры вывода и базы знаний), доступные на других ЭВМ в сети, и создавать экземпляры нужных компонентов. Для этой цели служат следующие функции (описание приводится на IDL):

```
void RegisterRuleProviderFactory(
      in RuleProviderFactory rpf);
void RegisterInferenceEngineFactory(
      in InferenceEngineFactory ief);

RuleProvider GetRuleProvider(in String Name);
InferenceEngine GetInferenceEngine(in String name);
InferenceEngine GetInferenceEngineByType(in long inftype);

StringSeq GetRuleProviders();
StringSeq GetInferenceEngines();
```

  Первые две функции позволяют зарегистрировать фабрики соответствующих компонентов. Они вызываются при запуске этих фабрик на компьютере-сервере соответствующих услуг.

  Следующие три функции позволяют получать экземпляры компонентов инструментария по имени или (для процессора вывода) по типу поддерживаемого вывода.

- **Обеспечивать простой интерфейс для внешних CORBA-объектов**. Для этой цели предусмотрены различные функции вывода, воспринимающие в качестве аргумента ссылки на компоненты (в этом случае о создании компонентов заботится вызывающая программа), имена компонентов (в этом случае необходимые компоненты создаются автоматически), имена шаблонов вывода или же строку, уникально описывающую конфигурацию вывода (т.н. UICD – Unified Inference Configuration Descriptor):

```
String GetByRef(in String theGoal,
           in RuleProvider theRuleProvider,
           in InferenceEngine theInfEngine,
           in ProblemState PS,
           in UserInterface UI);
String GetByNames(in String theGoal,
           in String theRuleProvider,
           in String theInfEngine,
           in ProblemState PS,
           in UserInterface UI);
String GetByUICD(in String theUICD);
String GetByPattern(in String PatternName);
```

Строка, описывающая конфигурацию вывода, позволяет одной строкой задавать конфигурации произвольной сложности, например:

```
c=shwars;g=diagnosis;kb=fork(pre_diag,complete_diag);ie=brie
```

В данном примере `c=` задает имя менеджера, `g=` — цели вывода, `kb=` — базу знаний (используется компонент fork для слияния динамических знаний из двух баз знаний), `ie=` задает используемый процессор вывода (обратный вывод).

- **Поддержание набора шаблонов вывода.** Для удаленного пользователя конфигурация компонентов в некоторой конфигурации компьютеров может быть неизвестна. В этом случае удобно создавать и поддерживать **шаблоны вывода**, которые будут представлять собой синонимы для определенных конфигураций. Соответствующие функции менеджера включают в себя регистрацию шаблонов указанием UICD, комбинации имен компонентов, ссылок на фабрики и т.д.

2.2.8. Fork-компоненты

Из соображений универсальности и простоты реализации процессоры вывода могут работать с одним источником данных о задаче и с

одним источником знаний. Для вывода по нескольким базам знаний необходимо "на лету" комбинировать динамические знания из различных баз знаний по некоторому критерию. Для этой цели служит т.н. fork-компонент. Точно также для подключения нескольких источников данных необходим fork-компонент для комбинирования статических знаний.

Fork-компонент для комбинирования динамических знаний (DynaFork) реализует интерфейс **RuleProvider** (см. раздел 2.2.4). При запуске этому компоненту указываются ссылки или имена источников знаний, правила из которых должны комбинироваться. Возможно реализовать несколько различных DynaFork-компонентов, использующих различные стратегии комбинирования правил. Простейший такой компонент включен в инструментарий под именет **com.shwars.diet.RuleProvider.Fork**.

Аналогичным образом устроен компонент для комбинирования статических знаний (StatFork). Типичным использованием такого компонента может быть случай, когда данные о задаче сначала ищутся в базе данных, а уж затем, в случае неудачи, запрашиваются у пользователя.

2.2.9. Утилиты инструментария

В комплект инструментария входят также утилиты для доступа к различным функциям из командной строки. В их числе:

RuleList — распечатка правил из удаленной базы знаний в удобочитаемом виде.

server — Основная интерактивная утилита инструментария для запуска всех процессов, образующих серверные компоненты на некоторой ЭВМ. Конфигурация запускаемых компонентов устанавливается в конфигурационном файле **server.conf**.

consult — Интерактивная консультационная оболочка с текстовым интерфейсом.

consultApp — Интерактивная консультационная оболочка с графическим интерфейсом: приложение или апплет.

Таким образом, для конфигурирования группы ЭВМ для распределенного вызова достаточно, установив на каждую ЭВМ инструментарий, соответствующим образом исправить файлы конфигурации **server.conf** на каждой из ЭВМ и запустить на каждой из ЭВМ утилиту **server** (при этом, разумеется, должны присутствовать указанные в конфигурации файлы баз знаний). После этого можно запускать вывод утилитами **consult** или **consultApp**.

2.2.10. Библиотеки для доступа к удаленной консультации

Для использования инструментария из других прикладных информационных систем были разработаны и реализованы рассмотренные ниже компоненты.

2.2.10.1. Java API удаленной консультации

Для доступа к инструментарию из программ на Java служит соответствующий набор классов в рамках пакета **com.shwars.diet.API**. При этом для доступа к конкретным компонентам можно использовать непосредственно менеджер, определив ссылку на него низкоуровневыми функциями **ClusterManagerAPI**, или же использовать более высокоуровневые функции, примерно соответствующие функциям менеджера. Для использования API пользователь должен создать класс, реализующий интерфейс **com.shwars.diet.API.UserInterface**, и вызвать одну из функций **Get**, передав созданный класс в качестве параметра. При создании интерактивных приложений в качестве реализации интерфейса может выступать само приложение или апплет.

2.2.10.2. DLL для удаленной консультации

Для доступа к инструментарию из приложений на других языках без использования CORBA на платформе Windows служит динамическая библиотека **remdiet.dll**. Эта библиотека создана с использованием VisiBroker для C++ в системе программирования Microsoft Visusal C++ 5.0. Для использования библиотеки программа пользователя должна

зарегистрировать CALLBACK-процедуры, соответствующие функциям пользовательского интерфейса, и вызвать одну из процедур **Get** из библиотеки.

2.2.10.3. Использование инструментария из системы программирования Borland Delphi

В настоящее время для быстрого прототипирования многих информационных систем служит система Borland Delphi. Для осуществления удаленной консультации с использованием инструментария из Delphi возможно 3 способа:

- Использование библиотеки **dietrem.pas**, представляющей собой инкапсуляцию динамической библиотеки **remdiet.dll**. В этом случае для консультации приходится использовать практически те же шаги, что и при использовании DLL.

- Реализация Delphi-компонента. При этом компонент также использует **remdiet.dll**, но процесс включения в состав приложения удаленной консультации становится более визуальным. Необходимо в свойствах компонента указать CALLBACK-процедуры соответствующего типа или же Datasource для автоматического соединения с некоторой таблицей в Delphi.

- Использование встроенной реализации CORBA для доступа непосредственно к менеджеру инструментария. Этот подход открывает наиболее широкие возможности, так как становится возможным реализовывать на Delphi непосредственно компоненты инструментария, однако, данный подход в рамках дипломной работы реализован не был ввиду того, что реализация CORBA в Delphi весьма отличается от стандарта и пока еще не полностью устоялась.

2.2.11. Дополнительные компоненты для создания встраиваемых экспертных систем

При создании реальных информационно-интеллектуальных систем с удаленной консультацией, особенно систем коммерческой направленно-

сти, может оказаться удобным снабдить систему некоторой **встроенной** (embedded) экспертной системой с ограниченными возможностями или недостаточно высоким качеством консультации, при этом предоставив ему доступ к более качественной, обновляемой базе знаний посредством удаленной консультации.

Для создания встраиваемых экспертных систем были предложены различные подходы, один из которых состоит в оформлении экспертной системы в виде библиотеки, подключаемой к основной программе [5, 6]. При этом система содержит в явном виде локальную копию базы знаний с возможностью модификации пользователем. Такой подход, по сути дела, представляет собой интеграцию полноценной экспертной системы в информационную.

В данной работе предлагается альтернативный подход, который состоит в генерации по базе знаний программного кода на языке высокого уровня, выполняющего процесс вывода. При этом правила базы знаний переводятся во фрагменты кода, и полученный код включается в информационную систему как неотъемлимая часть. Интеллектуальная компонента в такой информационной системе является *полностью встроенной*, а не представляет собой достаточно сложную самостоятельную систему.

К достоинствам такого подхода можно отнести:

- Для сравнительно небольших баз знаний (а к таковым как правило и относятся тот класс "полудемонстрационных" систем, которые удобно делать встроенными) объем полученного программного кода может уступать полнофункциональной экспертной системе с раздельным процессором вывода и базой знаний за счет того, что экономится часть системы, ответственная за операции ввода-вывода, перевод правил во внутреннее представление и т.д.

- Так как правила транслируются в сложнопонимаемый код на языке программирования, то задача выделения исходного текста базы знаний из готовой системы существенно усложняется.

- Программная система со встроенной таким образом интеллектуальной компонентой является неделимым продуктом и, следова-

тельно, проще в установке, так как не требуется устанавливать дополнительные runtime-компоненты экспертной системы.

Для реализации такого подхода в рамках инструментария был реализован ряд утилит (в составе пакета **com.shwars.diet.trans**) для преобразования потока динамических знаний в программный код на различных языках программирования, соответствующий проведению над данными знаниями обратного вывода. Сравнительная легкость реализации таких утилит и компактность генерируемого ими кода объясняется простотой используемой стратегии вывода и механизмов представления знаний. К этим утилитам относятся **genDelphi** (выходной код представляет собой ObjectPascal с использованием Delphi Runtime) и **genC** (для генерирования кода на языке Си).

2.3. Использование инструментария для решения прикладных задач

Разработанный в рамках данной работы инструментарий является инструментом для создания на его основе конкретных прикладных систем. В данном разделе будут кратко описанные созданные автором проекты, выполненные на основе инструментария, а также соображения по реализации некоторых других проектов с использованием распределенного вывода.

2.3.1. Модельная база знаний по диагностике больных с заболеваниями предстательной железы

В качестве модельной базы знаний для апробации инструментария была разработана база знаний диагностики больных с заболеваниями предстательной железы. Разработка выполнялась совместно с врачом Государственной клинической больницы им. С.П.Боткина Лукьяновым И.В. (эксперт) и аспирантом МАИ Заведеевым И.А. (инженер по знаниям) в рамках договора.

База знаний состоит из трех независимых частей, оформленных в виде индивидуальных компонент:

- Выработка предварительного диагноза на основании субъективных симптомов пациента.
- Выработка уточненного диагноза на основании осмотра и инструментальных тестов.
- Выработка рекомендаций к лечению.

База знаний представляет собой законченный модуль по диагностике определенного класса заболеваний и может совместно с аналогичными модулями другой тематики выступать как составная часть системы общей диагностики.

2.3.2. Информационно-учетная система диагностики больных с заболеваниями предстательной железы

На основе разработанной базы знаний совместно с Лукьяновым И.В. (эксперт) и Заведеевым И.А. (инженер по знаниям) была создана информационно-учетная система диагностики больных с заболеваниями предстательной железы, которая в настоящее время используется в лечебной практике отделения урологии ГКБ им. С.П.Боткина [2]. Система осуществляет идентификацию заболевания простаты с подробной постановкой диагноза и выработкой рекомендаций по применению различных методов оперативного (трансуретральная резекция, лазерное воздействие либо открытое вмешательство) и консервативного лечения.

Система содержит интуитивный пользовательский интерфейс, обеспечивает программную поддержку базы данных пациентов, легко интегрируется в существующий документооборот и позволяет проводить диагностику и выработку рекомендаций на основании содержащихся в базе данных сведений с использованием встроенной базы знаний или удаленной консультации на основе инструментария. Встроенная база знаний использует оригинальную методику, описанную в разделе 2.2.11.

Более подробная информация о системе содержится в [2].

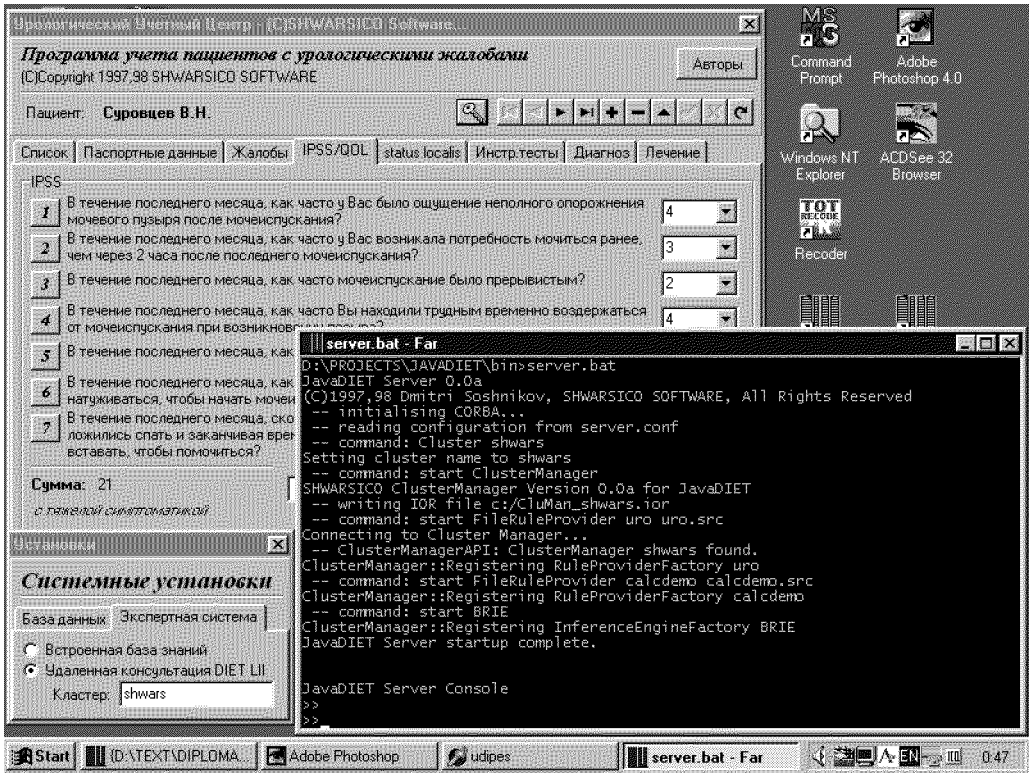

Рис. 15. Экранные формы учетно-диагностической программы

2.3.3. Об интеллектуальной навигации в Интернет

В качестве еще одной потенциальной области применения инструментария приведем пример интеллектуального поиска в Интернет. Этот пример приводится лишь как возможное направление дальнейшей работы по применению инструментария в реальных проектах, и соответствующая технология не была апробирована автором.

Одна из основополагающих проблем всемирной паутины — сложность поиска нужной информации. Странички всемирной паутины особенны тем, что, помимо собственно наполнения, они содержат некоторую дополнительную семантическую метаинформацию (знания) о наполнении в виде ссылок на другие релевантные ресурсы. Таким образом, весь набор информации в сети представляет собой сложную структуру в виде многофункционального графа, дуги которого (связи) имеют разную природу и могут описывать различные семантические отношения: синонимию, омонимию, таксономию и др.

Однако, такое представление не помогает, когда нужно найти вполне конкретную информацию. В этом случае приходится прибегать к кон-

текстному поиску, который не всегда дает хорошие результаты из-за возможной синонимичности критериев поиска. Поиск нечеткой информации (т.е. информации, для которой сложно сформулировать критерии поиска в виде ключевых слов) занимает долгое время, так как необходимо путешествовать по целому ряду вершин графа.

Одним из подходов к решению описанных проблем может быть представление знаний о наполнении страниц *в явном виде*, а не только косвенно в виде ссылок. Такой подход будет включать в себя поддержание на каждом узле локальной базы знаний о наполнении узла в соответствии с некоторым стандартом (онтологией). Такие базы знаний должны оперировать некоторыми общими входными атрибутами, задающими критерии поиска (например, классификация области знаний), а в процессе идентификации подходящих ресурсов задавать наводящие вопросы. Распределенный вывод в таких базах знаний позволит находить действительно релевантные странички даже по нечетким запросам пользователей.

Основная проблема описанной методики состоит в необходимости создания базы знаний об узлах сети, что представляет собой чрезвычайно трудоемкую задачу. Вторая проблема — неэффективность распределенного вывода при большом числе узлов — может быть частично решена за счет использования вывода на стороне сервера (в этом случае каждому серверу навигационных баз знаний придется еще и поддерживать свой процессор вывода), а также за счет выбора подсегментов баз знаний на начальном этапе в зависимости от тематической области поиска.

Рассмотренная технология пока еще далека от практического применения в глобальном масштабе сети, однако может вполне применяться для интеллектуальной навигации в рамках одного узла. Дальнейшая разработка такой методики и создание соответствующего программного обеспечения может стать предметом самостоятельных исследований.

Глава 3.

Экономическая часть

В настоящее, трудное для нашей страны время, при создании новой системы невольно задумываешься об экономических аспектах ее применения. В условиях конкуренции трудно позволить себе заниматься созданием программного продукта, не находящего себе эффективного и прибыльного применения.

Разрабатываемый в рамках дипломного проекта продукт направлен скорее не на непосредственное использование, а служит для создания сложных распределенных систем, приносящих экономическую пользу. Поэтому в первом разделе данной главы мы рассмотрим общие соображения по поводу эффективности распределения деятельности вообще. Этот аргумент в частности покажет важность создания систем поддержки распределенной деятельности, в том числе и интеллектуальных систем. Во втором разделе будут рассмотрены более прагматические соображения о применении удаленной консультации на практике.

3.1. Об экономическом эффекте распределения

Вопрос об экономическом эффекте распределения неразрывно связан с вопросом специализации. Распределение деятельности означает в первую очередь специализацию каждого узла распределенной системы на определенной деятельности.

Согласно известному экономическому **закону относительного**

преимущества (Law of Comparative Advantage) [29], если у одного узла есть *относительное* преимущество в производстве определенного продукта (в смысле ресурсоемкости производства одной единицы), а у другого — в производстве другого продукта, то наибольшей экономической эффективности можно добиться путем полной специализации каждого узла на продукте, в котором у него есть относительное преимущество. При этом абсолютное преимущество, т.е. умение изготавливать продукт с меньшими расходами, роли не играет. Данный закон, однако, справедлив только в отсутствие транспортных расходов.

В настоящее время есть одна сфера, в которой транспортные расходы практически отсутствуют — это сфера информационных услуг. Действительно, стоимость передачи информации по географически удаленным регионам по сети Интернет чрезвычайно мала. Поэтому в сфере информационных услуг широкое распространение приобретают т.н. **виртуальные корпорации** — распределенные структуры, в которых услуги предоставляются в любом географическом регионе вне зависимости от того, в каком регионе находится поставщик услуг. В последние годы имеется тенденция распространения виртуальных корпораций в производственную и промышленную сферы, и создаются т.н. **виртуальные производственные (промышленные) корпорации** [28]. Это стало возможно благодаря развитию средств **компактного интеллектуального производства** — станков, способных воспроизвести изделие по его электронному макету без человеческого вмешательства за короткий срок.

Для обеспечения деятельности виртуальной корпорации потребуются соответствующие распределенные информационные системы, в том числе интеллектуальные. Именно распределенные интеллектуальные системы помогут автоматически выбрать оптимальное место производства того или иного продукта среди имеющихся в наличии, подсказать требуемые методы изготовления и спрогнозировать стоимость.

3.2. Экономические аспекты использования удаленной консультации

Простейшей формой обмена знаниями по сети являестя удаленная консультация. Преимущества удаленной консульнации были отмечены в разделе 1.2.1 (стр. 28). В данном разделе покажем, что удаленная консультация несет в себе экономические преимущества как для продавца услуг (разработчика базы знаний), так и для пользователя.

3.2.1. Преимущества разработчика

Основной эффект удаленной консультации для разработчика состоит в том, что, предоставляя удаленную консультацию, он не теряет контроля над самой базой знаний. В частности, интеллектуальная собственность не передается покупателю, а передается лишь однократная услуга. При продаже традиционной интеллектуальной системы разработчик получает ее стоимость однократно, и теряет контроль над дальнейшей прибылью, полученной от использования системы. При этом удаленная консультация дополнительно уменьшает проблему несанкционированного распространения, так как представляющий наибольшую ценность интеллектуальный продукт вместе с системой не распространяется.

Естественно, если система не пользуется популярностью, т.е. среднее число проводимых консультаций M мало, то разработчику выгоднее продать систему целиком. В этом случае его полная прибыль (без учета расходов) на один комплект продаж будет составлять стоимость комплекта P.

В случае использования удаленной консультации, прибыль от комплекта продаж составит $P_1 + P_2 M$, где P_1—стоимость информационной части системы, P_2 — стоимость одной удаленной консультации. Естественно предположить, что $P_1 < P$, и основная прибыль в случае удаленной консультации достигается только при большом среднем числе консультаций, т.е. при популярности системы.

3.2.2. Преимущества пользователя

Для пользователя приобретение системы с возможностью удаленной консультации имеет следующие преимущества:

- Нет необходимости сразу платить большую сумму за полную систему, оплата производится постепенно.
- Расходы зависят от числа консультаций, что позволяет сэкономить при малом числе консультаций.
- База знаний разработчика может претерпевать изменения и обновления, при этом для использования новой версии нет необходимости преобретать новую программу.

Последний пункт пожалуй наиболее интересен. Традиционные системы имеют определенный срок службы T, после которого надо приобретать новую систему из-за морального устаревания старой. При этом удельные расходы пользователя на единицу времени равны $\frac{P}{T}$. В случае удаленной консультации при частоте использования F (консультаций в единицу времени) удельный расход составит

$$\frac{P_1 + P_2 F T}{T} = \frac{P_1}{T} + P_2 F$$

Так как в этом случае стоимость оболочки P_1 мала по сравнению с временем T, то удельная стоимость удаленной консультации зависит прежде всего от частоты использования консультации. Поэтому во многих случаях, когда консультация требуется очень редко (например, консультации по вопросам здоровья в период беременности) или однократно, использование удаленной консультации крайне оправданно.

Заключение

В ходе выполнения дипломной работы сделано следующее:

- Исследованы принципы построения интеллектуальных систем, способы представления знаний, архитектуры систем с коллективным выводом.
- Проведен сравнительный анализ существующих технологий построения распределенных систем.
- Рассмотрены основные подходы к построению распределенных интеллектуальных систем: удаленная консультация, агентные архитектуры.
- Предложен принцип построения распределенных интеллектуальных систем на базе многокомпонентного инструментария, оптимальные способы представления знаний в таких системах.
- В соответствии с этим принципом спроектирована архитектура инструментария, включая структуры данных для статических и динамических знаний, интерфейсы компонентов инструментария.
- На языке Java реализованы основные компоненты инструментария, достаточные для построения сравнительно развитых интеллектуальных систем.
- Предложен принцип построения встраиваемых экспертных систем на базе высокоуровневой кодогенерации.
- Разработан набор компонентов для интеграции средств удаленной консультации с использованием инструментария в информационные системы.

- Совместно с врачом высшей категории ГКБ им. С.П.Боткина Лукьяновым И.В. (эксперт) и аспирантом МАИ Заведеевым И.А. (инженер по знаниям) разработана база знаний диагностики больных заболеваниями предстательной железы.

- На базе предложенных принципов спроектирована, разработана и внедрена экспертная учетно-диагностическая система диагностики больных заболеваниями предстательной железы.

Новыми результатами, выносимыми на защиту, являются:

- Принцип построения распределенных интеллектуальных систем на базе многокомпонентного инструментария, позволяющий реализовывать различные конфигурации вывода и обмен знаниями на уровне статического и динамического представления.

- Реализованное в соответствии с этим принципом в архитектуре CORBA оригинальное программное обеспечение инструментария для построения распределенных интеллектуальных систем.

- Реализованная на базе инструментария учетно-диагностическая система, содержащая встроенную базу знаний и возможность удаленной консультации по набору распределенных баз знаний в среде интернет.

Полученные результаты могут развиваться по следующим направлениям:

- Расширение модели представления знаний, включение в архитектуру инструментария фреймовых компонентов.

- Улучшение методов логического вывода, в частности, в направлении автоматизации распараллеливания процесса распределенного вывода.

- Исследование методологий разработки и проектирования распределенных баз знаний, в том числе независимыми экспертами.

- Разработка дополнительных компонентов инструментария, например, естественно-языкового интерфейса.

- Создание на базе инструментария прикладных систем и технологий, например, интеллектуальной навигации в Интернет или синтеза программ в распределенной обучающей системе по курсу "Информатика".

Основные результаты дипломной работы были представлены на Международном научном семинаре "The 1st International Workshop on Computer Science and Information Technologies" в январе 1999 г.

Разработанная автором совместно с Лукьяновым И.В. и Заведеевым И.А. на основании полученных в работе результатов информационно-интеллектуальная учетно-диагностическая система больных с заболеваниями предстательной железы внедрена в лечебную практику отделения урологии ГКБ им. С.П.Боткина.

Публикации автора по теме дипломной работы:

1) 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.

2) Лукьянов И.В., Машкович В.Э., Заведеев И.А., Сошников Д.В. Выбор оптимального метода трансуретрального лечения больных с инфравезикальной обструкцией. Тезисы 1-ой Всеросийской научно-практической конференции "Современные эндоскопические технологии в урологии". Уральская медицинская академия, 1999. (В печати).

Список использованных источников

[1] 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.

[2] Лукьянов И.В., Машкович В.Э., Заведеев И.А., Сошников Д.В. Выбор оптимального метода трансуретрального лечения больных с инфравезикальной обструкцией. Тезисы 1-ой Всеросийской научно-практической конференции "Современные эндоскопические технологии в урологии". Уральская медицинская академия, 1999. (В печати).

[3] Веб-узел `http://www.glasnet.ru/ lukianovi`.

[4] Журавлева Т.Э. Гибридный инструментарий интеллектуальных систем на основе расширенного логического программирования. Дис-с к.ф.-м.н. — М.: МАИ, 1993.

[5] Зайцев В.Е., Лукашевич С.Ю. Инструментальные средства для построения встроенных экспертных систем // Информатика, Г3-4, 1991. стр. 30-40.

[6] Зайцев В.Е., Исаев В.К., Лукашевич С.Ю., Хмелев А.К. Опыт интеграции интеллектуальной компоненты в систему автоматизированного проектирования // Информатика, Г3-4, 1991. стр. 74-80.

[7] DESS: Document and Reporting Decision Support System. European Software Laboratories, 1993.

[8] G.F.Luger, W.A.Stubblefield. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. — Benjamin Cummings Publishing Company, 1993. (2nd Ed.)

[9] Russel S., Norvig P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 1994.

[10] Ж.-Л.Лорьер. Системы искусственного интеллекта: Пер. с фр. — М.: Мир, 1991.

[11] К. Таунсенд, Д. Фохт. Проектирование и программная реализация экспертных систем на персональных ЭВМ: Пер. с англ. — М.: "Финансы и статистика", 1990.

[12] J. Girratano, G. Riley. Expert Systems: Principles and Programming. — PWS Publishing Company, Boston, 1993. (2nd Ed.)

[13] Дж. Элти, М. Кумбс. Экспертные системы: концепции и примеры: Пер. с англ. — М.: Финансы и статистика, 1987.

[14] Осуга С. Обработка знаний: Пер. с япон. — М.: Мир, 1989.

[15] Уэно Х., Кояма Т., Окамото Т., Мацуби Б., Исидзука М. Представление и использование знаний: Пер. с япон. — М.: Мир, 1989.

[16] Нейлор К. Как построить свою экспертную систему: Пер. с англ. — М.: Энергоатомиздат, 1991.

[17] Построение экспертных систем: Пер. с англ./Под ред. Ф.Хейеса-Рота, Д.Уотермана, Д.Лената. — М.: Мир, 1987.

[18] Веб-узел `http://www.cs.umbc.edu/agents/kse/kif/`.

[19] Веб-узел `http://www.cs.umbc.edu/kqml`.

[20] С. Дунаев. Intranet-технологии. — М.: Диалог-МИФИ, 1997.

[21] Д.О.Брюхов, В.И. Задорожный, Л.А. Калиниченко, М.Ю. Курошев, С.С. Шумилов. Интероперабельные информационные системы: архитектуры и технологии. // СУБД, Ґ4, 1995.

[22] CORBA: Architechture and Specification. Object Management Group, 1995. (Доступно на сервере `http://www.omg.org`)

[23] Weber J. Special Edition: Using Java. QUE Corporation, 1996.

[24] Веб-узел `http://www.gnu.org`.

[25] Rogerson D. Inside COM. — Microsoft Press, 1997.

[26] А.М.Робачевский. Операционная система UNIX. — СПб.: BHV – Санкт-Петербург, 1997.

[27] Ивахненко А.Г., Коппа Ю.А., Степашко В.С. и др., Справочник по типовым программам моделирования. — К.: Техніка, 1980.

[28] Пирогова Н. Как создать виртуальную корпорацию // Открытые системы, Ґ1, 1998. стр. 62-66.

[29] Microsoft Encarta'98 Encyclopedia, CD-ROM edition. Microsoft Press, 1998.

Приложение А.

Тексты программ

Приложение содержит описание интерфейсов компонентов инструментария на IDL, выборочные исходные тексты компонентов инструментария (полностью тексты не приводятся ввиду большого объема, а также по соображениям нераспространения интеллектуальной собственности), примеры использования инструментария с помощью библиотеки вызовов и непосредственно обращением к CORBA-объектам.

Описание интерфейсов компонентов инструментария на IDL

```
/* ------------------------------------------------ */
/* -                                              - */
/* -   Distributed Inference Engine Technology    - */
/* -   DIET Level II Version 1.0                  - */
/* -   (C)1997,98 Dmitri V. Soshnikov             - */
/* -   (C)1997,98 SHWARSICO SOFTWARE              - */
/* -                                              - */
/* ------------------------------------------------ */
/* -                                              - */
/* - Sep 1998 : DS : 1st DIET Level II Impl       - */
/* -                                              - */
/* ------------------------------------------------ */


module IDiet
{

 // ------------------ Basic data types -------------------

 typedef string String; // String implementation


 // Attribute-Value pair
 struct AVPair
 {
  String A;
```

```
 String V;
};

// Comparison
struct Comparison
{
 String A;
 String V;
 long CompareType;
};

typedef sequence<AVPair> AVPairSeq;
typedef sequence<Comparison> ComparisonSeq;
typedef sequence<String> StringSeq;


// Production rule
struct Rule
{
 long RuleType;
 ComparisonSeq IF;
 AVPairSeq THEN;
};

typedef long TransactionID;

// ------------------ Components -------------------

interface RuleProvider
{
 TransactionID StartTransaction(in String goal);
 Rule NextRule(in TransactionID TD);
 boolean HasMoreRules(in TransactionID TD);
 void EndTransaction(in TransactionID TD);
 StringSeq GetDirectory();
 String getName();
 long GetInfType(); // 1 - back, 2 - forw - required types of Inference
};

interface RuleProviderFactory
{
 String getName();
 RuleProvider Create(in String name);
 StringSeq List();
};

interface InfoProvider
{
 String Get(in String attrib, in String text, in StringSeq choices);
};

interface UserInterface : InfoProvider
{
 void Print(in String s);
 String Read();
};

interface ProblemState
{
 exception ValueAlreadySet {} ;
 String Get(in String attrib);
 void Set(in String attrib, in String value) raises (ValueAlreadySet);
};

interface ProblemContext : ProblemState
{
 void SetRuleProvider(in RuleProvider RP);
 void SetInfoProvider(in InfoProvider IP);
```

```
 void SetUserInterface(in UserInterface UI);
 RuleProvider GetRuleProvider();
 InfoProvider GetInfoProvider();
 UserInterface GetUserInterface();
};

interface InferenceEngine
{
 long GetInfType(); // get inference type: 1 - back, 2 - forw
 String Get(in String goal);
 void SetEnvironment(in ProblemContext PC);
 String getName();
};


interface InferenceEngineFactory
{
 String getName();
 InferenceEngine Create(in String name);
 StringSeq List();
 long GetInfType(); // get inference type: 1 - back, 2 - forw
};

interface ClusterManager
{
 void RegisterRuleProviderFactory(in RuleProviderFactory R);
 RuleProvider GetRuleProvider(in String Name);

 void RegisterInferenceEngineFactory(in InferenceEngineFactory ief);
 InferenceEngine GetInferenceEngine(in String name);
 InferenceEngine GetInferenceEngineByType(in long inftype);

 String Get(in String theGoal, in String theRuleProvider,
            in String theInfEngine, in ProblemState PS,
            in UserInterface UI);

 StringSeq GetRuleProviders();
 String getName();
};
};
```

Исходный текст процессора обратного вывода и соответствующей фабрики

```
/* ------------------------------------------------ */
/* -                                              - */
/* -   Distributed Inference Engine Technology    - */
/* -   DIET Level II Version 1.0                  - */
/* -   (C)1997,98 Dmitri V. Soshnikov             - */
/* -   (C)1997,98 SHWARSICO SOFTWARE              - */
/* -                                              - */
/* ------------------------------------------------ */
/* -                                              - */
/* - Sep 1998 : DS : 1st DIET Level II Impl       - */
/* - Backward Reasoning Inference Engine - BRIE   - */
/* ------------------------------------------------ */

package com.shwars.diet.InferenceEngine;

import com.shwars.diet.*;
import com.shwars.util.StringArith;
import com.shwars.util.common;
```

```
import java.util.Stack;

public class BRIE
 extends com.shwars.diet.InferenceEngine.InferenceEngine
{

 public final static int infType = InferenceEngine.IE_BACKWARD;


 // Implementation of the main IE reasoning method

  public String Get(String goal)
 {
  Rule R;
  String res = null;
  comment("Reasoning for goal "+goal);
  if (!com.shwars.diet.AVPair.isVolatile(goal))
  {
   // Check if the value already exists in ProblemStateCache
   res = PSC.Get(goal);
   if (!res.equals(""))
   {
    // Attribute already there!
    comment("Attribute "+goal+" already known");
    return res;
   }
  }
  Comparison C;

  // Value is not known or is volatile
  // Let's start reasoning!

  int TID = RP.StartTransaction(goal); // Start transaction with RuleProvider
  while (RP.HasMoreRules(TID))
  {
   R = new Rule(RP.NextRule(TID)); // Obtain the envelope Rule object
   // Apply the rule
   if (R.ruleType == R.RT_ASK)
   {
    // Ask question to the user. Performs UserInterface call (inherited from generic IE)
    res = Ask(goal,R.getAskQuestion(),R.getAskChoices());
    if (res==null) continue;
    else
    {
     if (PSC.Set(goal,res)) break; // value obtained
     else continue; // fallback on constraint violation
    }
   }

   if (R.ruleType == R.RT_EXEC)
   {
    com.shwars.diet.Pluguns.PluginHelper.Execute(R); // Execute plugin
    continue;
   }

   if (R.ruleType == R.RT_CALC)
   {
    // Calculate expression
    res = Calculate(R.IF);
    if (res==null) continue;
    if (res.equals("") && com.shwars.diet.Defs.fallbackOnEmptyCalc) continue;
    break;
   }

   if ((R.ruleType & Rule.RT_IF) == 0) continue; // skip not IF-type rules

   // Here we are left with one IF-rule
   if (R.THEN.length!=1) continue; // we cannot process rules with more then one THEN-clause
```

```
  boolean fire=true;
  for (int i=0;i<R.IF.length;i++)
  {
   C = R.IF[i];
   String s1 = Get(R.IF[i].A); // evaluate subgoals
   if ((s1!=null)&&(C.compare(s1))) continue;
   else
   {
    fire=false;
    break;
   }
  }
  if (fire) // Rule fires
  {
   res = R.THEN[0].V;
   if (PC.Set(goal,res)) break;
   else continue; // fallback on constraint violation
  }
 }
 RP.EndTransaction(TID); // End the RuleProvider Transaction
 comment("Result obtained: "+goal+" = "+res);
 if (res==null) res="";
 return res;
};

// Internal routine to do expression calculation
private String Calculate(AVPair A[])
{
 int i;
 Stack S = new Stack();
 String s,u,v;
 try
 {
  for (i=0;i<A.length;i++)
  {
   s = A[i].V;
   if (common.isInteger(s)) { S.push(s); }
   else if (s.length()>1 && s.charAt(0)=='"' && s.endsWith("\""))
     { S.push(s.substring(1,s.length()-1)); }
   else if (s.equals("+"))
     { v=(String)S.pop(); u=(String)S.pop(); S.push(StringArith.Add(u,v)); }
   else if (s.equals("-"))
     { v=(String)S.pop(); u=(String)S.pop(); S.push(StringArith.Subtract(u,v)); }
   else if (s.equals("*"))
     { v=(String)S.pop(); u=(String)S.pop(); S.push(StringArith.Multiply(u,v)); }
   else if (s.equals("/"))
     { v=(String)S.pop(); u=(String)S.pop(); S.push(StringArith.Divide(u,v)); }
   else if (s.equals("."))
     { v=(String)S.pop(); u=(String)S.pop(); S.push(u+v); }
   else
   {
    S.push(Get(s));
   }
  }
  return (String)S.pop();
 }
 catch(java.util.EmptyStackException e)
 {
  return "";
 }
}

public int GetInfType()
{
 return infType;
}
```

```
 public String getName()
 {
  return "BRIE";
 }
}



/* ------------------------------------------- */
/* -                                           - */
/* -   Distributed Inference Engine Technology   - */
/* -   DIET Level II Version 1.0                 - */
/* -   (C)1997,98 Dmitri V. Soshnikov            - */
/* -   (C)1997,98 SHWARSICO SOFTWARE             - */
/* -                                           - */
/* ------------------------------------------- */
/* -                                           - */
/* - Sep 1998 : DS : 1st DIET Level II Impl      - */
/* - Generic Inference Engine Superclass         - */
/* ------------------------------------------- */

package com.shwars.diet.InferenceEngine;

public class InferenceEngine
 extends com.shwars.diet.IDiet._InferenceEngineImplBase
 implements com.shwars.diet.IDiet.InferenceEngine
{

 com.shwars.diet.IDiet.RuleProvider RP = null;
 com.shwars.diet.IDiet.InfoProvider IP = null;
 com.shwars.diet.IDiet.UserInterface UI = null;
 com.shwars.diet.IDiet.ProblemContext PC = null;

 ProblemStateCache PSC = null;

 public final static int IE_BACKWARD = 1;
 public final static int IE_FORWARD = 2;

 public String Get(String goal)
 {
  com.shwars.util.common.error("InferenceEngine","Inference is
                               not implemented in the generic stub");
  return "";
 };

 public void SetEnvironment(com.shwars.diet.IDiet.ProblemContext PC)
 {
  this.PC = PC;
  PSC = new ProblemStateCache(PC);
  RP = PC.GetRuleProvider();
//  IP = PC.GetInfoProvider();

// This version of InferenceEngine always requests and uses UserInterface
// if there is no userInterface (non-interactive consultation)
// the ProblemContext creates an empty one

  UI = PC.GetUserInterface();
 }

 public String Ask(String attr,String s, String choices[])
 {
  return UI.Get(attr,s,choices);
 }

 public int GetInfType()
 {
  return 0;
```

```
 }

 public String getName()
 {
  return "GenericInferenceEngineStub";
 }

 public void comment(String s)
 {
  if (UI!=null) UI.Print(" * "+s+"\n");
 }

}




/* ------------------------------------------------ */
/* -                                              - */
/* -   Distributed Inference Engine Technology    - */
/* -   DIET Level II Version 1.0                  - */
/* -   (C)1997,98 Dmitri V. Soshnikov             - */
/* -   (C)1997,98 SHWARSICO SOFTWARE              - */
/* -                                              - */
/* ------------------------------------------------ */
/* -                                              - */
/* - Sep 1998 : DS : 1st DIET Level II Impl       - */
/* - Backward Reasoning Inference Engine Server   - */
/* ------------------------------------------------ */

package com.shwars.diet.InferenceEngine;

import org.omg.CORBA.*;
import com.shwars.util.common;
import com.shwars.diet.*;

public class BRIESrv
 extends Thread
{
 String CName = null;
 String Name = null;
 String av[] = null;

 BRIE B = null;

 public BRIESrv(String CName, String Name, String av[], ORB orb)
 {
  /* --------------------------
  common.println("FileRuleProvider Server");
  common.println(Defs.Name+" Version "+Defs.Version);
  common.println(Defs.Copyright);
  ---------------------------- */
  this.CName = CName;
  this.Name = Name;
  this.av = av;
  common.println("Starting backwards reasoning engine "+Name+" for cluster "+CName);
  common.vprintln(" -- Instantiating BRIE...",8);
  B = new BRIE();
  try
  {
// Do we need another ORB here? Probably its irrelevant, since ORB is a pseudo anyway...
//   common.vprintln(" -- Initializing CORBA core...",8);
//   ORB orb = ORB.init(av,null);
   common.vprintln(" -- Registering BRIE with ORB...",8);
   orb.connect(B);

   common.vprintln(" -- Ready to serve requests...",8);
   common.diag("BRIE","Ready to serve requests");
```

```
  }
  catch (Exception e)
  {
   common.error("BRIESrv","Error intializing CORBA core");
   e.printStackTrace(System.out);
  }
 }

 synchronized public BRIE getBRIE()
 {
  return B;
 }

 public void run()
 {
  try
  {
     java.lang.Object sync = new java.lang.Object();
      synchronized (sync) { sync.wait(); }
  }
  catch (Exception e)
  {
   common.error("BRIESrv","Unknown Error");
   e.printStackTrace(System.out);
  }
 }
}




/* ------------------------------------------------ */
/* -                                              - */
/* -   Distributed Inference Engine Technology    - */
/* -   DIET Level II Version 1.0                  - */
/* -   (C)1997,98 Dmitri V. Soshnikov             - */
/* -   (C)1997,98 SHWARSICO SOFTWARE              - */
/* -                                              - */
/* ------------------------------------------------ */
/* -                                              - */
/* - Sep 1998 : DS : 1st DIET Level II Impl       - */
/* - Backward Reasoning Inference Engine Factory - */
/* ------------------------------------------------ */

package com.shwars.diet.InferenceEngine;

import org.omg.CORBA.*;

public class BRIEFactory
 extends com.shwars.diet.IDiet._InferenceEngineFactoryImplBase
 implements com.shwars.diet.IDiet.InferenceEngineFactory
{
  String CName, Name, av[];
  ORB orb;

  public BRIEFactory(String CName, String Name, String av[], ORB orb)
  {
   this.CName = CName;
   this.Name = Name;
   this.av = av;
   this.orb = orb;
  }

  public String getName()
  {
   return Name;
  }
```

```
  public int GetInfType()
  {
   return BRIE.infType;
  }

  public com.shwars.diet.IDiet.InferenceEngine Create(String name)
  {
   if ((!name.equals("")) && (!Name.equals(name)) && (name!=null)) return null;
   com.shwars.util.common.diag("BRIEFactory","Request for new BRIE");
   BRIESrv b = new BRIESrv(CName, Name, av, orb);
   if (b==null)
   {
    com.shwars.util.common.diag("BRIEFactory","Cannot create BRIE Instance");
    return null;
   }
   b.start();
   return b.getBRIE();
  }

  public String[] List()
  {
   String s[] = new String[1];
   s[0] = new String(Name);
   return s;
  }
};


/* ---------------------------------------------- */
/* -                                            - */
/* -   Distributed Inference Engine Technology  - */
/* -   DIET Level II Version 1.0                - */
/* -   (C)1997,98 Dmitri V. Soshnikov           - */
/* -   (C)1997,98 SHWARSICO SOFTWARE            - */
/* -                                            - */
/* ---------------------------------------------- */
/* -                                            - */
/* - Sep 1998 : DS : 1st DIET Level II Impl     - */
/* - Backward Reasoning Inference Factory Server - */
/* ---------------------------------------------- */

package com.shwars.diet.InferenceEngine;

import com.shwars.util.common;
import org.omg.CORBA.*;
import com.shwars.diet.ClusterManagerAPI;

public class BRIEFactorySrv
{
 public static void main(String av[])
 {
  com.shwars.util.common.setVerbosity(com.shwars.diet.Defs.verbosity);
  String CName = common.propArg(av,0);
  String Name = common.propArg(av,1);
  if (com.shwars.util.common.propArg(av,2)!=null || CName == null || Name == null)
  {
   common.println("Invalig invocation.");
   common.println("SYNTAX:");
   common.println(" BRIEFactorySrv [<orb.options>] <Cluster> <Name>");
   return;
  }
  common.println("Starting BRIEFactory "+Name+" for cluster "+CName);
  try
  {
   common.vprintln(" -- Initializing CORBA core...",8);
   ORB orb = ORB.init(av,null);
   common.vprintln(" -- Instantiating BRIEFactory...",8);
   BRIEFactory bf = new BRIEFactory(CName,Name,av,orb);
```

```
   common.vprintln(" -- Registering BRIEFactory with ORB...",8);
   orb.connect(bf);
   common.vprintln(" -- Connecting to Cluster Manager...",8);
   com.shwars.diet.IDiet.ClusterManager CM = ClusterManagerAPI.resolve_reference(orb,CName);
   if (CM==null)
   {
    common.error("BRIEFactorySrv","Cannot connect to cluster manager");
    return;
   }
   common.vprintln(" -- Registering with Cluster Manager...",8);
   CM.RegisterInferenceEngineFactory(bf);
   common.vprintln(" -- Ready to serve requests...",8);
   common.diag("BRIEFactorySrv","Ready to serve requests");
   java.lang.Object sync = new java.lang.Object();
      synchronized (sync) { sync.wait(); }
  }
  catch (Exception e)
  {
   common.error("BRIEFactorySrv",
         "Error intializing CORBA core, or ClusterManager not alive.");
   e.printStackTrace(System.out);
  }
 }

}
```

Пример использования инструментария через CORBA-объекты

```
/* ------------------------------------------------ */
/* -                                              - */
/* -   Distributed Inference Engine Technology    - */
/* -   DIET Level II Version 1.0                  - */
/* -   (C)1997,98 Dmitri V. Soshnikov             - */
/* -   (C)1997,98 SHWARSICO SOFTWARE              - */
/* -                                              - */
/* ------------------------------------------------ */
/* -                                              - */
/* - Sep 1998 : DS : 1st DIET Level II Impl       - */
/* - Demonstration example 1                      - */
/* ------------------------------------------------ */

package com.shwars.diet.demos;

import com.shwars.diet.*;
import org.omg.CORBA.*;

public class test1
{
 public static void main(String av[])
 {
  ORB orb = ORB.init(av,null); // Obtain reference to CORBA ORB

  // Obtain ClusterManager reference via ClusterManagerAPI
  com.shwars.diet.IDiet.ClusterManager CM = ClusterManagerAPI.resolve_reference(orb,"shwars");

  // Obtain references to RuleProvider and InferenceEngine
  com.shwars.diet.IDiet.RuleProvider RP = CM.GetRuleProvider("uro");
  com.shwars.diet.IDiet.InferenceEngine IE = CM.GetInferenceEngine("BRIE");

  // Create new empty problem context
  ProblemContext PC = new ProblemContext();

  // Set the problem context
```

```
  PC.SetRuleProvider(RP);
  PC.SetUserInterface(new com.shwars.diet.InfoProvider.DefUserInterface());

  // Set the inference problem context
  IE.SetEnvironment(PC);

  // Start consultation by calling Get method of InferenceEngine
  System.out.println("Diagnosis = "+IE.Get("diagnosis"));
 }
}
```

Пример использования инструментария через компонент-менеджер

```
package com.shwars.diet.demos;
import com.shwars.diet.*;
import org.omg.CORBA.*;

public class test2
{
 public static void main(String av[])
 {

  ORB orb = ORB.init(av,null); // Obtain reference to CORBA ORB

  // Obtain ClusterManager reference via ClusterManagerAPI
  com.shwars.diet.IDiet.ClusterManager CM = ClusterManagerAPI.resolve_reference(orb,"shwars");

  // Immediately start the inference by calling ClusterManager GET method
  System.out.println("Result is: "+CM.Get("diagnosis","uro","",null,
                      new com.shwars.diet.InfoProvider.DefUserInterface()));
 }
}
```

Пример использования инструментария через библиотеку DietAPI

```
package com.shwars.diet.demos;
import com.shwars.diet.*;
import org.omg.CORBA.*;

public class test3
{
 public static void main(String av[])
 {
  // Initialize the API and connect to clustermanager
  com.shwars.diet.API api = new com.shwars.diet.API.Init("shwars");

  // Register user interface class
  api.RegisterUIClass(new testUI());

  // Start the inference
  System.out.println("The result is: "+api.Get("diagnosis","uro",null);
 }
}


// Sample user interface class
package com.shwars.diet.demos;
```

```
public class testUI
{
 // Print procedure
 public void print(String s) { System.out.println(s); }

 // Get attribute value
 public String get(String goal, String text, String[] choices)
 {
  System.out.println(text);
  return com.shwars.util.common.readLn(System.in);
 }
}
```

Исходный текст библиотеки удаленной консультации dietrem.dll

```
// DIET Project
// DLL Library DIETRem.dll
// (C)1998,99 SHWARSICO SOFTWARE, All Rights Reserved
// (C)1998,99 Dmitri Soshnikov

#include <fstream.h>
#include "diet_c.hh"
#include "diet_s.hh"
#define null 0
#define OK 0
#define ERROR 1

// #define LOG "c:/dietrem.log"

CORBA::ORB_var orb;
CORBA::BOA_var boa;

#ifdef LOG
 ofstream oflog = ofstream(LOG);
#endif

typedef void PRINT_FUNC(const char *s);
typedef char* ASK_FUNC(const char*,const char*,const char*);

PRINT_FUNC *thePF = null;
ASK_FUNC *theAF = null;

IDiet::ClusterManager_var CM;

char _buf[2048];

// User Interface Class
class UserIntImpl : public _sk_IDiet::_sk_UserInterface
{
 public:

  void Print(const char* s) { (*thePF)(s); };

  char* Read() { return ""; }

  char* Get(const char* s, const char* t, const class IDiet::StringSeq& L);

};

char* UserIntImpl::Get(const char* s, const char* t, const IDiet::StringSeq& L)
{
```

```
 _buf[0]=0;
 for (int i=0;i<L.length();i++)
 {
  strcat(_buf,L[i]);
  strcat(_buf,"%");
 }
 if (strlen(_buf)>1) _buf[strlen(_buf)-1]=0;
 return (*theAF)(s,t,_buf);
}

// Initialization function
extern "C" int __stdcall Init()
{
 int argc = 0;
 char **argv = 0;
#ifdef LOG
 oflog << "DIETREM.DLL: Initializing..." << endl;
 oflog.flush();
#endif
 try
 {
  orb = CORBA::ORB_init(argc, argv);
  boa = orb->BOA_init(argc, argv);
  return OK;
 }
 catch (const CORBA::Exception& e)
 {
  return ERROR;
 }
}

// Registering Callback procedures
extern "C" void __stdcall RegisterProc(PRINT_FUNC *pf, ASK_FUNC *af)
{
 thePF = pf;
 theAF = af;
 (*thePF)("DIETREM.DLL Version 0.0a");
}

// Connecting to ClusterManager
extern "C" int __stdcall Connect(char *fname)
{
 char s[4096];
 try
 {
#ifdef LOG
  oflog << "DIETREM.DLL: Resolving clustermanager " << fname << endl;
  oflog.flush();
#endif
  strcpy(s,"c:/CluMan_");
  strcat(s,fname);
  strcat(s,".ior");
  ifstream ifs = ifstream(s);
  ifs >> s;
#ifdef LOG
  oflog << "DIETREM.DLL: IOR obtained: " << s << endl;
  oflog.flush();
#endif
  CORBA::Object_var o = orb->string_to_object(s);
  CM = IDiet::ClusterManager::_narrow(o);
  return OK;
 }
 catch (const CORBA::Exception& e)
 {
  return ERROR;
 }
}
```

```
char R[1024];

// Performing inference
extern "C" char* __stdcall Get(char *goal, char *kb)
{
  char *res;
  try {
    UserIntImpl UII;
    boa->obj_is_ready(&UII);

    IDiet::UserInterface_ptr UI = &UII;

    res = CM->Get(goal,kb,"",0,UI);
    strcpy(R,res);

  } catch(const CORBA::Exception& e) {
    return NULL;
  }
  return(R);
}
```

Исходный текст библиотеки для использования средств инструментария из ObjectPascal

```
{ ---------------------------------------------- }
{ -                                            - }
{ -   Distributed Inference Engine Technology  - }
{ -   DIET Level II Version 1.0                - }
{ -   (C)1997,98 Dmitri V. Soshnikov           - }
{ -   (C)1997,98 SHWARSICO SOFTWARE            - }
{ -                                            - }
{ ---------------------------------------------- }
{ -                                            - }
{ - Jan 1999 : DS : 1st DIET Level II Impl     - }
{ - ObjectPascal Library                       - }
{ ---------------------------------------------- }


unit dietrem;

interface

const drOK = 0;
      drErr = 1;

type TAskFunc = function (s,t,u : PChar): PChar; cdecl;
     TPrintFunc = procedure (p : PChar); cdecl;
     TInitFunc = function : Integer; stdcall;
     TConnectFunc = function (s : PChar) : integer; stdcall;
     TRegisterFunc = procedure (pf : TPrintFunc; af : TAskFunc); stdcall;
     TGetFunc = function (goal, kb : PChar) : PChar; stdcall;

var _Init : TInitFunc;
    _Connect : TConnectFunc;
    _RegisterProc : TRegisterFunc;
    _Get : TGetFunc;

procedure RegisterCallbackProc(pf : TPrintFunc; af : TAskFunc);
function Connect(cluman : String): integer;
function Get(goal, kb : String): String;

var LibHandle : integer;

implementation
```

```
uses Windows;

procedure RegisterCallbackProc(pf : TPrintFunc; af : TAskFunc);
begin
 _RegisterProc(pf,af);
end;

function Connect(cluman : String): integer;
begin
 result:=_Connect(PChar(cluman));
end;

function Get(goal, kb : String): String;
begin
 result:=_Get(PChar(goal),PChar(kb));
end;

begin
// Initialization
 libHandle:=LoadLibrary('dietrem.dll');
 if libHandle<>0 then
  begin
   @_Init := GetProcAddress(LibHandle, 'Init');
   @_Connect := GetProcAddress(LibHandle, 'Connect');
   @_RegisterProc := GetProcAddress(LibHandle, 'RegisterProc');
   @_Get := GetProcAddress(LibHandle, 'Get');
   if _Init<>drOk then LibHandle:=0;
  end;
end.
```