УДК 681.324.06 + 62:007.51

ИНСТРУМЕНТАРИЙ JULIA ДЛЯ СОЗДАНИЯ РАСПРЕДЕЛЕННЫХ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ НА ОСНОВЕ ПРОДУКЦИОННО- ФРЕЙМОВОГО ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Д.В. Сошников

В статье рассматриваются аспекты построения распределенных интеллектуальных систем для накопления и удаленного использования знаний на основе продукционно-фреймового представления. Предложенная модель распределенной фреймовой иерархии с формальной точки зрения является статической делиберативной многоагентной архитектурой, однако для нее характерно унифицированное представление знаний во всех узлах системы, как для внутренней обработки, так и для обмена. Таким образом, данная модель является автоонтологичной в том смысле, что применяемые для решения задач знания и эксплицитная спецификация концептов предметной области и их свойств не разделяются. В статье описан инструментарий JULIA (Java Universal Library for Intelligent Applications), реализующий предлагаемую архитектуру построения распределенных интеллектуальных систем.

Введение

В настоящее время в области информационных технологий наблюдается резкое повышение интереса к построению распределенных приложений и систем, которые могут использоваться в сетевом окружении группой пользователей. Это, безусловно, связано как с совершенстованием технической базы предприятий и повсеместным внедрением компьютерных сетей для повышения эффективности работы и автоматизации бизнес-процессов, так и с распространением и популяризацией всемирной компьютерной сети Интернет.

В то время как для таких областей, как создание распределенных баз данных и информационных систем существуют достаточно развитые и изученные технологии и методы, построение распределенных интеллектуальных систем находится в настоящее время под пристальным вниманием, так как такие системы могут открыть дорогу к разделению и совместному использованию *знаний* в компьютерных сетях [1]. Все большее использование знаний с возможностью обмена ими по сети позволят поднять содержательность информационных ресурсов сети на принципиально новый уровень, так как сетевые ресурсы будут содержать не слабоструктурированную текстовую информацию, а представленные определенным образом знания, пригодные для использования в процессе логического вывода для решения задач пользователей.

Другой важный аспект в создании систем, основанных на знаниях — это их интеграция с традиционными информационными системами, возможность совместного использования императивной и объектно-ориентированной парадигм программирования с интеллектуальными представлениями знаний, а также прозрачный доступ интеллектуальной компоненты к реляционным базам данных. Многие оболочки экспертных систем (GURU, VP Expert, и др.)

ориентированы на диалоговый режим работы, и не допускают использования режима консультации из внешней программы. Более современные системы (CLIPS [2], JESS [3], а также AMZI Prolog [4]) допускают вызов ядра системы из программ на традиционных языках программирования, однако степень интеграции интеллектуальной и императивной компоненты все равно остается недостаточной (как правило, отсутствует эффективная возможность вызова императивной компоненты из базы знаний).

С другой стороны в практических задачах часто возникает необходимость интеграции механизма вывода с базами данных, например для проведения экспертных консультаций на основе содержащихся в них данных. Существующие на данный момент подходы (активные базы данных [5] и легковесные дедуктивные базы данных [6,7]) требуют применения специализированных СУБД с соответствующим языком представления знаний и механизмом вывода, исключая таким образом использование существующего сервера баз данных или имеющихся баз знаний.

Таким образом представляется актуальным разработка методологии создания распределенных систем на базе унифицированного представления знаний [8], которое будет с одной стороны обеспечивать прозрачную интеграцию с объектной и компонентной моделями прострoения программных систем, с реляционными БД, а с другой допускать распределение фрагментов знаний по сети с последующим распределенным выводом для решения некоторой прикладной задачи.

Методология построения распределенных интеллектуальных систем на основе распределенной фреймовой иерархии

В рамках распределенного искусственного интеллекта [9] в настоящее время наибольшее распространение получила многоагентная архитектура [10], в которой интеллектуальное поведение системы в целом достигается за счет взаимодействия набора автономных агентов, которые в свою очередь могут быть слабоинтеллектуальными (реактивными) или полноценно-интеллектуальными (делиберативными, основанными на явном представлении знаний предметной области). Различают целые классы различных по природе агентных архитектур, которые обычно классифицируют по некоторым свойствам (мобильные или статические, обучающиеся или нет и т.д.) [10].

Для успешного взаимодействия агентов помимо общего внешнего языка представления знаний (например KQML [11]) эксплицитно задается концептуализация предметной области — онтология [12], обычно представляющая собой таксономию понятий, расширенную некторыми правилами-аксиомами. Эти правила специфицируются на некотором языке представления знаний (например, KIF [13]), однако агенты могут лишь подразумевать эти знания неявно, не используя их для логического вывода. Более того, природа агентов может быть весьма различной в рамках

одного агентного сообщества (например, реактивные агенты могут вообще не использовать явное предствление знание и логический вывод), в этом случае онтологии скорее выполняют роль общего словаря объектов предметной области.

В то время как одним из основных характеристик агентной архитектуры является автономность и атомарность агентов, интерес представляет также альтернативный подход [1], при котором распределенная интеллектуальная система строится из неинтеллектуальных компонентов, соответствующих классическим компонентам интеллектуальных систем: процессоров вывода, источников знаний, доски объявлений (соответствующей рабочей памяти в классической экспертной системе), интерфейса пользователя и др. В этом случае компоненты обмениваются по сети статическими (например, парами атрибут-значение) или динамическими (продукционными правилами) знаниями, которые могут комбинироваться, образуя различные конфигурации обмена знаниями по сети. В то время как в агентной архитектуре внутренняя структура логического вывода каждого агента (и, соответственно, статические знания и накопленный опыт) скрывается от внешнего окружения, в компонентной архитектуре присутствует полный обмен знаниями, благодаря которому реализуется распределенный вывод и распределенное решение задач.

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

Предлагаемая методология основана на гибридной архитектуре иерархического комбинирования узлов интеллектуальной системы с фреймовой моделью представления знаний [14]. Каждый из узлов содержит в себе некоторый участок фреймовой иерархии, т.е. свои базу знаний, процессор вывода и рабочую память, называемые **фрейм-миром** (frame world). Возможны различные модели комбинирования фреймовых суб-иерархий на распределенных узлах сети: все миры могут объединяться в единую фреймовую иерархию с общим родителем, некоторый фрейм может по-очереди присоединяться (наследоваться) к различным иерархиям (**псевдомножественное наследование**), образуя аналог модели доски объявлений [15], или же фрейм-миры могут взаимодействовать более сложным образом, представляя собой сеть с распределенными статическими знаниями. Все фрейм-миры, участвующие в некотором общем процессе вывода, образуют **кластер**, или **распределенную фреймовую иерархию** [16].

Удаленное наследование фреймов обеспечивает передачу знаний по сети за счет удаленного вызова, при котором по сети реально передаются только значения атрибутов, а логический вывод проводится на ЭВМ, располагающей соответствующими динамическими знаниями (правилами). Однако в предлагаемой модели может быть реализовано и **мобильное наследование** с передачей динамических знаний фрейма-родителя по сети для использования локальным процессором вывода. Предложенная архитектура может быть также дополнена средствами для обмена динамическими знаниями в виде произвольных продукционных правил прямого и обратного вывода по сети и использования **репозитария знаний** с загрузкой правил по мере необходимости в процессе вывода (on-demand rule loading), что позволяет строить более эффективные приложения даже в двухуровневой модели клиент-сервер.

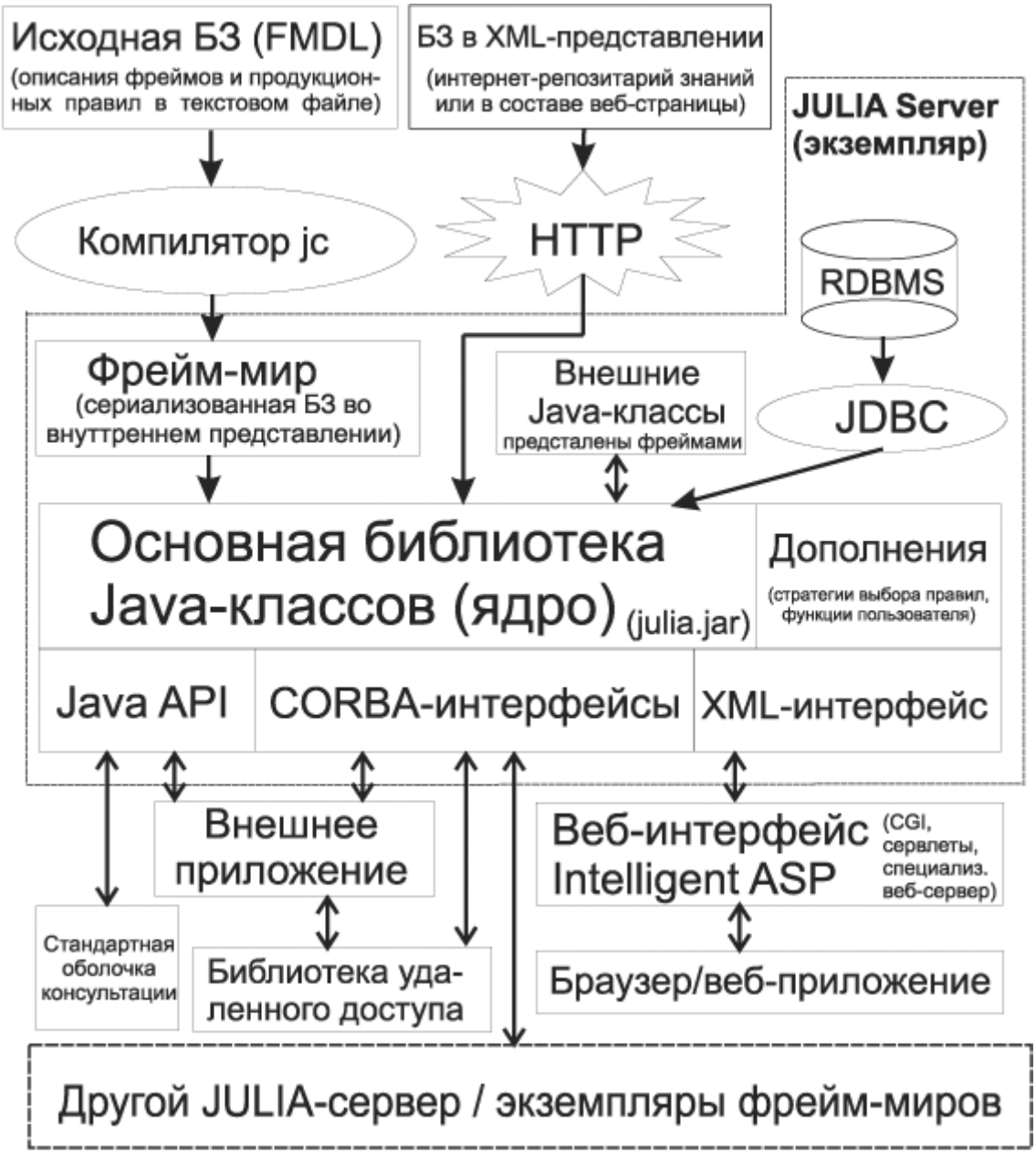
Использование фреймового представления знаний позволяет также достаточно гибко обеспечивать доступ процессора вывода к реляционным базам данных и алгоритмическим расширениям, реализованным на традиционных (объектно-ориентированных) императивных языках программирования. Таблицы реляционных БД представляются во фреймовой модели в виде семейства фреймов (экстенсионала, фрейм-класса) с общим родителем; при этом значения слотов фреймов берутся из реляционной таблицы по необходимости в процессе обратного вывода, а выведенные значения могут либо сохраняться в таблице, либо находится в памяти (в зависимости от наличия в таблице столбца с соответствующим именем). Доступ к алгоритмическим расширениям осуществляется с помощью специальных **внешних фреймов**, представляющих собой интерфейс с методами некоторого класса или компонента (например, Java-класса, COM-объекта или компонента JavaBeans).

Архитектура открытого инструментария JULIA

На основе предлагаемой методологии на языке Java реализован инструментарий JULIA (Java Universal Library for Intelligent Applications), оформленный в виде семейства классов с открытым программным интерфейсом (API). Таким образом, интеллектуальная функциональность, предоставляемая инструментарием, может легко использоваться в составе программных комплексов и информационных систем, реализованных на Java или других языках программирования. Выбор Java в качестве базового языка реализации позволяет использовать инструментарий в среде Интернет в виде апплета в модели толстого клиента, в виде сервлета (в модели тонкого клиента), или же в многоуровневых распределенных системах. В качестве протокола для организации распределенных вычислений выбрана архитектура CORBA [17], а для обмена данными и знаниями по сети во внешнем представлении — язык, основанный на XML, что позволяет интегрировать инструментарий в различные информационные системы, реализованные на разных языках программирования. В инструментарии реализован набор CORBA-объектов для

удаленного доступа к системе со стороны других JULIA-серверов, а также со стороны программных систем, поддерживающих CORBA. Сравнительно небольшой объем ядра, необходимого для работы инструментария (около 80 Кб), а также предложенный механизм загрузки правил по мере необходимости делает использование инструментария привлекательных даже в небольших проектах.

Архитектура гибридного инструментария JULIA для построения распределенных интеллектуальных систем приведена на рисунке [16]:



В основе системы лежит ядро (представляющее собой библиотеку Java-классов, распространяемую в виде одного файла `julia_rt.jar`), которое включает в себя набор классов для поддержки фреймов и наследования, библиотеку типов данных, алгоритмы реализации прямого и обратного вывода, ряд дополнительных классов для обеспечения доступа к БД, к внешним Java-классам, компонентам COM и JavaBeans, набор стратегий выбора правил при

разрешении конфликтов, библиотека основных функций и др. В отдельной библиотеке (`julia_remote.jar`) оформлены классы, обеспечивающие поддержку CORBA для распределенного вывода. Такое разделение библиотек классов позволяет использовать JULIA в качестве библиотеки для обеспечения интеллектуальности локальных приложений, в которых не нужен удаленный доступ. При этом объем требуемого файла классов существенно уменьшается, и может составлять около 70 килобайт.

В верхней части диаграммы показаны различные способы, с помощью которых система получает знания и данные для своей работы; ниже ядра — различные интерфейсы для использования библиотеки в интерактивной консультации (пользовательские интерфейсы) или из внешней информационной системы (программные интерфейсы).

Представление статических знаний

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

Фрейм рассматривается как набор **слотов**, каждый из которых может содержать значение заранее определенного или произвольного **типа**. Основными типами являются **скалярный**, который подразделяется на **числовой** (целый или с плавающей точкой), **логический** и **строковый** подтипы, **списковый** (содержащий произвольное количество элементов любого типа, в том числе спискового) и **ссылочный** (содержащий ссылку на фрейм или слот). Библиотека может расширяться другими типами данных (например, тектовым, шаблонным и др.) путем реализации соответствующих Java-классов, унаследованных от базового родительского класса `TAny`.

Каждый фрейм в процессе выполнения представляется экземпляром некоторого Java-класса, унаследованного от `Frame`. Класс фрейма определяет то, каким образом будет формироваться значения слотов в процессе вывода. Библиотека содержит специальные классы для доступа к базам данных (`DBFrame` и `DBFrameClass`), для интерфейса с объектами, реализованными на императивных языках программирования (`JavaClassFrame`, `JavaBeanFrame` и `CORBAObjFrame`). Библиотека также может быть расширена дополнительными классами фреймов, хотя необходимость в этом редко возникает, так как произвольный Java-код может быть включен в состав фреймовой модели в виде Java-фреймов.

Большинство фреймов обычно представляются классом `DefFrame`, который обеспечивает хранение значений слотов в памяти и связывание с ними процесса прямого и обратного логического вывода. С каждым слотом связывается тип содержащегося в нем значения, наборы

процедур-демонов (выполняемых при присваивании слоту значения) и процедур-запросов (выполняемых, когда требуется узнать значение слота), а также набор ограничений (constraints) на множество допустимых значений (фасетные ограничения на диапазон значений `FacetConstraint`, на принадлежность определенному множеству значений `SetContraint`, либо ограничения общего вида, представленные произвольным логическим выражением `ExprConstraint`).

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

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

База знаний обычно описывается на специализированном языке высокого уровня декларативности **FMDL** (Frame Model Definition Language), который включает в себя средства описания статических закономерностей предметной области (фреймовой иерархии) и динамических знаний в виде продукций прямого и обратного вывода. Описание на FMDL затем транслируется в семейство Java-объектов (фрейм-мир), который может непосредственно

использоваться для проведения логического вывода, либо может быть сериализован для дальнейшей загрузки и использовния в составе других приложений.

Другим способом внешнего представления знаний служит язык, основанный на XML [18], который может использоваться для сохранения всей фреймовой иерархии или ее части. Таким образом, часть знаний предметной области может сохраняться в виде XML-репозитария и загружаться по протоколу HTTP в различные фрейм-миры. Использование XML позволяет, в случае необходимости, переходить от одного XML-диалекта представления знаний к другому с помощью XSLT, обеспечивая диалог между двумя системами с несколько различным внешним представлением знаний. XML-представление, однако, не является достаточно наглядным, поэтому имеет смысл использовать FMDL для описания базы знаний, с последующей автоматической генерацией XML-представления для отдельных ее частей.

Продукционные правила и механизм обратного вывода

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

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

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

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

Для подлежащего выполнению правила производится **укороченное вычисление** условной части, которое, в свою очередь, может приводить к запросу значений других слотов, т.е. в свою очередь к рекурсивному инициированию процесса обратного вывода для других слотов. В случае успеха (в результате вычисления выражения получается истинностное значение) выполняется присваивание слоту значения, содержащегося в правой части правила (это может привести к дальнейшему логическому выводу, если в выражении в правой части в свою очередь присутствуют ссылки на другие слоты), в случае неуспеха — рассматривается следующее правило.

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

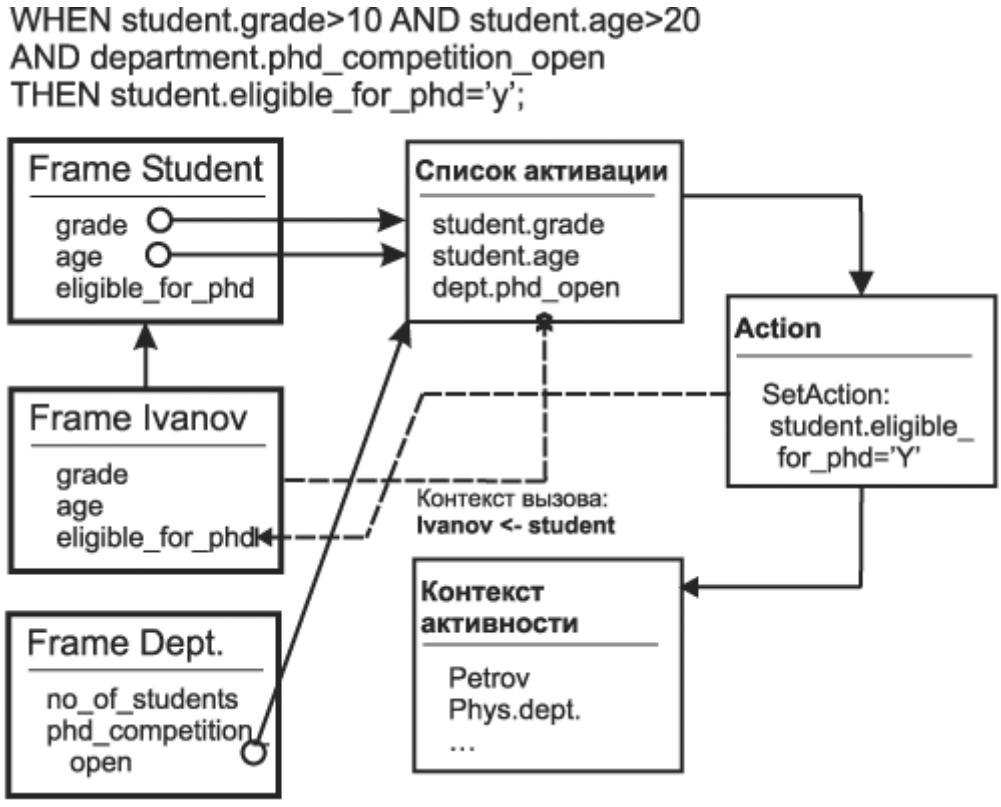
В любом случае, после присваивания слоту значения производится проверка ограничений (constraints), связанных со слотом (**фасетные ограничения**, facet constraints), с фреймом или со всей фреймовой иерархией (**референциальные ограничения**, referential constraints). Ограничения могут накладываться на тип присваиваемого значения, на диапазон или на множество значений, а также в форме произвольных выражений или определенным образом унаследованных Java-классов. Если вновь присвоенное значение не удовлетворяет ограничениям — оно обнуляется (слоту присваивается неопределенное значение), и применение процедур продолжается, таким образом производится откат назад (backtracking) и применение других правил. В текущей реализации инструментарии роль ограничений сводится к проверке и обеспечению отката назад при поиске решения.

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

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

Процедуры-демоны и механизм прямого вывода. Комбинированный вывод.

Правила прямого вывода при трансляции преобразуются в сеть, наподобие используемой в семействе алгоритмов Rete [19] и TREAT [20], в конечных узлах которой расположены, с одной стороны, процедуры-демоны, присоединенные к соответствующим слотам, входящим в левую часть выражения, а с другой — процедуры, выполняемые при истинности данного выражения (см. рисунок).



При изменении значения некоторого слота активизируются все связанные с ним процедуры-демоны, которые пытаются вычислить выражения, находящиеся в левой части продукционных

правил. При этом выражения (представленные соответствующими древовидными структурами в памяти) вычисляются по специальному алгоритму с запоминанием промежуточных результатов в узлах-операциях дерева, которые в данном случае играют роль β-памяти, применяемой в алгоритме Rete. Необходимость в α-памяти в используемом варианте алгоритма отсутствует (в качестве α-памяти используются сами слоты), так как унификация правила со значениями в рабочей памяти как таковая не производится, а заменяется неявной унификацией по наследованию, которая достигается за счет вызова процедур-демонов всех родительских фреймов с передачей текущего фрема (вызвавшего активацию) в качестве контекста вызова.

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

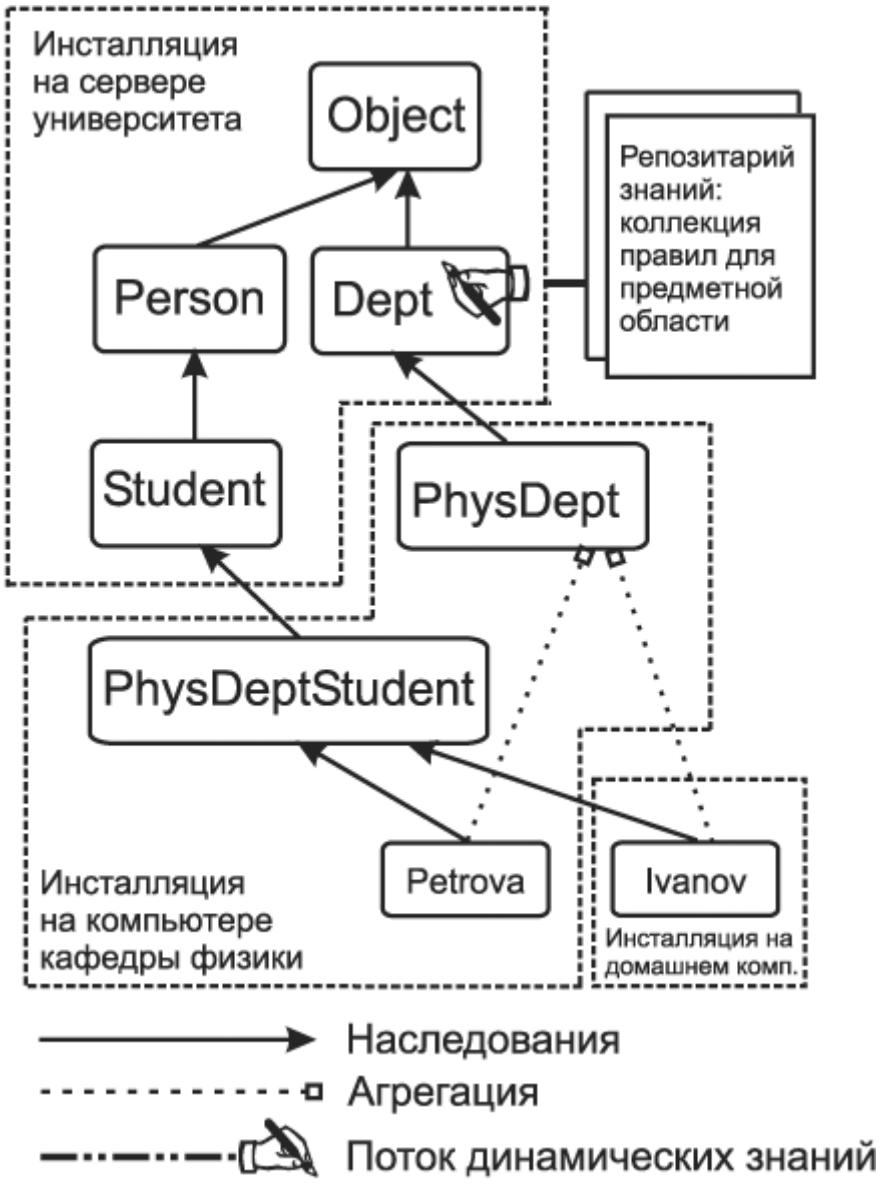
Основным в системе является обратный вывод, который инициируется при запросе значения целевого слота, и заканчивается получением его значения. В общем случае, база знаний может содержать правила как обратного, так и прямого вывода — в этом случае рассмотренный ранее механизм вывода соответствует применению исчерпывающего прямого вывода после каждого шага обратного вывода (точнее, после каждого присваивания, возникшего в результате обратного вывода).

Логический вывод в распределенной фреймовой иерархии

Распределенная фреймовая иерархия предполагает, что вся распределенная система представляет собой единую иерархию фреймов, связанную отношением наследования, которое может распространяться между узлами системы. Каждый узел содержит в себе некоторую суб-иерархию со своей базой знаний, но может ссылаться (отношением наследования или агрегации) на фреймы, расположенные на других узлах. Логический вывод в системе в целом происходит аналогично выводу в нераспределенной иерархии, с учетом удаленного вызова методов на удаленных фреймах. Такой удаленный вывов позволяет осуществлять обмен значениями слотов, и таким образом построить обмен знаниями на основе обмена статической информацией о состоянии задачи.

Так как распределенная система в свою очередь представляет собой единую фреймовую иерархию, то модель распределенного вывода представляет собой синхронный поиск в общем пространстве состояний задачи [21], т.е. общее пространство состояний распределено по

фреймовой иерархии, и соответственно поиск управляется поочередно разными процессорами. Однако из-за синхронного характера вывода (т.е. управление передается поочередно от процессора к процессору, в то время как все другие процессоры находятся в состоянии ожидания или готовности к ответу на запрос) не возникает проблем с синхронизацией и распараллеливанием вывода.



Будем называть ядро библиотеки JULIA (поддерживающее сетевые расширения) с загруженным фрейм-миром, работающее на некотором узле сети **инсталляцией**. Инсталляция включает в себя некоторую фреймовую иерархию (суб-иерархию), т.е. набор фреймов с уникальным именованием, связанных отношением наследования. С фреймами связаны соответствующие процедуры-демоны и процедуры-запросы, соответствующие некоторой базе знаний. В каждой иерархии возможно использование ссылок на **удаленные фреймы**, т.е. на фреймы, принадлежащие фрейм-мирам других инсталляций. Пример распределенной фреймовой иерархии приведен на рисунке.

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

Библиотека поддерживает два вида ссылок на удаленные фреймы: **статические ссылки** или раннее связывание, и **динамические ссылки** или позднее связывание. В первом случае еще на этапе процессирования исходного текста на FMDL создается фрейм-посредник, который жестко связывается со ссылкой на удаленный фрейм. В случае динамического связывания фрейм-посредник создается только в процессе выполнения, и динамически находит удаленный фрейм, используя доступные средства именования.

В распределенной фреймовой иерархии различают два вида ссылок на удаленные фреймы: наследование от удаленного фрейма и агрегация удаленного фрейма. В свою очередь в случае агрегации удаленный фрейм может использоваться в посылках правила прямого вывода, либо в выражении. Последний случай не представляет особого интереса, так как мало чем отличается от удаленного вызова с целью получения значения слота — следует однако иметь в виду, что такой удаленный вызов зачастую приводит к процессу обратного вывода. Если в случае обратного вывода встречаются правила, запрашивающие значения у пользователя — осуществляется обратный вызов (callback) вызывавшей инсталляции, которая и осуществляет взаимодействие с пользователем. Таким образом, в стандартном случае исходные данные запрашиваются на одной ЭВМ (что обычно и требуется); однако всегда возможно использовать процедуры, которые будут осуществлять распределенный сбор данных, что характерно, например, для задач распределенного интеллектуального контроля и управления [9,22].

В случае наследования от удаленного фрейма обычно происходит переход процесса обратного вывода через границы инсталляции. В этом случае в алгоритме обратного вывода производится вызов процедуры вычисления значения слота для фрейма-посредника, который, в свою очередь, производит удаленный вызов соответствующей процедуры на другой инсталляции, передавая в качестве базового фрейма ссылку на текущий фрейм. В этом случае в удаленной инсталляции также создается фрейм-посредник, который связывается с базовым фреймом для данного вывода. Соответственно, процессом вывода управляют динамические правила,

расположенные на удаленной ЭВМ, но значения атрибутов присваиваются исходному базовому фрейму в процессе обратного удаленного вызова.

Таким образом, обмен по сети производится только значениями слотов — статическими знаниями о состоянии задачи — что тем не менее приводит к передаче знаний, точнее, дистанционному применению знаний или удаленной консультации. Отличие от удаленной консультации в чистом виде состоит в том, что возможно несколько уровней удаленного наследования, а также расширение модели дополнительными знаниями на каждом из уровней. Простая удаленная консультация реализуется наследованием одного фрейма от некоторого удаленного родителя (например, фрейм Ivanov на приведенном выше рисунке) с последующим запросом целевого слота, что приводит к процессу обратного вывода на удаленной ЭВМ с заполнения слотов исходного базового фрейма полученными в процессе вывода значениями.

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

Альтернативой описанному выше статическому наследованию, реализуемому удаленным вызовом, служит мобильное наследование, при котором по сети передаются правила вывода во внутреннем представлении (в виде процедур-запросов и процедур-демонов) для использования в логическом выводе на локальном узле. Мобильное наследование позволяет осуществлять доступ к удаленным репозитариям знаний без использования активного сервера, так как в частном случае возможно хранение внутреннего представления правил в виде файлов, передаваемых по обычному протоколу HTTP. По мере необходимости правила или целые фрейм-миры могут загружаться на локальную ЭВМ и использоваться в процессе вывода. Такой подход напоминает предложенный в [23] механизм комбинирования логических программ — только в данном случае осуществляется естественное комбинирование иерархических структур относительно отношения наследования.

Технически, описанный метод построения распределенной фреймовой иерархии на базе инструментария может быть основан на любом транспортном протоколе удаленного вызова, поддерживающем компонентную модель и достаточно развитые интерфейсы вызовов. К таким протоколам относятся Java RMI, Microsoft DCOM, DCE, CORBA и др. В текущей реализации использовался механизм CORBA, так как он является стандартом, и позволяет реализовывать

отдельные элементы распределенной фреймовой иерархии на других языках программирования. В частности, расширение инструментария такими средствами, как `CORBAFrame` (доступ к произвольному CORBA-объекту как к фрейму), процедурами-запросами и процедурами-демонами, вызывающими произвольный метод некоторого CORBA-объекта, позволяют получить на базе инструментария полноценную среду программирования, интероперабельную со многими современными распределенными системами, построенными на базе CORBA. Внедрение такого инструментария на CORBA-магистраль позволяет дополнять ее интеллектуальными функциями, используя при этом единый механизм удаленного вызова.

Например, в индустриальных областях задачи контроля за состоянием распределенных объектов подразумевают распределенный сбор данных и прозрачный обмен собранными и выведенными данными между узлами системы. Такие задачи как правило решаются применением централизованной модели распределения компонентов системы, в качестве которой как правило выбирают именно CORBA [22].

Использование CORBA не противоречит применению некоторого языка для обмена знаниями между фреймами. В многоагентных системах часто используется стандартный язык обмена знаниями KQML (Knowledge Query and Manipulation Language) [11] или его подмножество, позволяющий строить гибкий обмен сообщениями между агентами. В реализации инструментария проще и эффективнее не использовать специализированный язык внешнего представления, так как в качестве него неявно используется некоторое представление, генерируемое транспортным уровнем CORBA. Однако, для обеспечения совместимости инструментария с другими агентными системами возможна поддержка подмножества KQML, по крайней мере для обеспечения ответов на запросы о значениях слотов. Работа над таким развитием инструментария планируется, и позволит использовать предложенную методику распределенной фреймовой иерархии совместно с другими агентными архитектурами.

Отмеченный ранее синхронный характер логического вывода упрощает реализацию инструментария, но лишает распределенный вывод некоторой привлекательности с точки зрения эффективности. Заметим, что данный дефект не является следствием модели распределенной фреймовой иерархии — напротив, в принципе вполне допустима параллельная работа процессоров вывода в различных инсталляциях. В этом случае подразумевается, что если в некоторое выражение входят ссылки на слоты удаленных фреймов, то начинать их вычисление следует параллельно.

Следует понимать, однако, что такой параллелизм нарушает традиционную семантику обратного вывода, при которой значения, входящие в выражение, вычисляются по укороченной схеме слева направо. Зачастую создатели баз знаний имеют в виде именно такую семантику выполнения, и параллелизм может некоторым образом нарушить предполагаемый процесс

вывода. Однако в ряде случаев параллельное вычисление весьма желательно — например, если цель нашей системы — сравнить значения некоторого атрибута, полученные различными независимыми экспертными системами. Поэтому с точки зрения автора представляется наиболее разумным в явном виде включить в язык конструкции для указания возможности параллельного вычисления некоторого выражения или операции, что позволит использовать параллелизм совместно с полным вычислением выражения в том случае, где это не противоречит требуемой семантике.

Распределенное решение проблем и синтез решения

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

При рассмотрении задачи распределенного накопления и использования знаний можно выделить два типовых случаях [1]:

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

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

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

Альтернативным способом решения может быть разбиение исходных данных по набору независимых фреймов, каждый из которых наследуется от своей иерархии, и затем подвергается

логическому выводу для получения результатов. В случае обратного вывода такой подход может оказаться удобнее, так как исходные данные отсутствуют, и необходимо просто собрать интересующие атрибуты у набора фреймов, унаследованных от соответствующих иерархий.

В случаях с совпадающими доменами требуется получить решения одной и той же задачи с использованием различных баз знаний (т.е. провести логический вывод для одного и того же набора исходных данных), с целью дальнейшего сравнения полученных результатов на основе простой процедуры или логического вывода. В этом случае требуется создать несколько копий исходного фрейма, и использовать различные копии в качестве наследников нескольких родительских иерархий. При использовании обратного вывода этот случай будет мало отличаться от предыдущего — лишь на этапе анализа данных нужно будет учитывать возможные противоречия, возникшие из-за различных решений исходной задачи различными экспертными системами.

Использование JULIA в прикладных интеллектуально-информационных системах

Инструментарий JULIA был успешно применен для решения ряда практических задач, подробное рассмотрение которых выходит за рамки данной статьи. Благодаря свой открытой архитектуре, возможно использования инструментария в качестве интеллектуальной компоненты программных систем, реализованных на языка Java, для построения систем удаленной консультации на стороне клиента в виде Java-апплета, в составе веб-приложений с использованием сервлетов или специализированной технологии IASP [24], либо в составе сложных интеллектуально-информационных систем, основанных на CORBA [25]. На основе JULIA реализованы прототипы экспертной системы по оптимизации и планированию рекламной кампании интернет-ресурсов в руссоязычном сегменте Интернет [26], система адаптивного интеллектуального тестирования в области создания веб-приложений [27], в настоящее время создается интеллектуальная система дистанционного обучения и контроля знаний по логическому программированию.

Наибольший интерес представляет разработанная интеллектуально-информационная учетно-диагностическая система больных с заболеваниями предстательной железы, внедренная в лечебный процесс ГКБ им. С.П.Боткина [28]. Информационная компонента и пользовательский интерфейс данной системы реализованы на системе программирования Borland Delphi, интеллектуальная компонента представлена встроенной базой знаний на основе высокоуровневой кодогенерации [29], или внешней базой знаний на основе полноценного JULIA-сервера. В последнем случае применение распределенной фреймовой иерархии позволяет использовать знания различных врачей-урологов для совместного решения задачи с последующим сравнением/анализом диагнозов, а также для комбинирования знаний врачей различной

специализации с целью выработки более полного диагноза на основе множества симптомов. В перспективе такой подход может лечь в основу распределенной информационной интеллектуализации лечебного процесса, в котором компьютерная сеть лечебного комплекса будет использоваться для поддержания распределенной базы знаний по всему спектру заболеваний, с возможностью гибкого доступа со стороны внутренней информационной системы автоматизации документооборота, а также извне на основе решений электронной коммерции. Аналогично, при использовании такой же технологии в других лечебных учреждениях возможна либо дальнейшая меж-институциональная специализация баз знаний при условии соблюдения общей онтологии, либо взаимное использование баз знаний для уточнения/подтверждения локального диагноза.

Выводы

Как было отмечено по ходу изложения, предложенная архитектура на базе продукционно-фреймового представления знаний, реализованная в инструментарии JULIA, позволяет достичь следующих результатов:

- Гибкая интеграция с существующими информационными системами, позволяющая расширять все функции системы произвольным Java-кодом, вызывать Java-объекты и компоненты CORBA и JavaBeans непосредственно из фреймовой модели в процессе вывода, а также наоборот встраивать интеллектуальную систему в состав более крупных программных комплексов, реализованных как на Java, так и не других языках программирования.
- Распределенное накопление знаний и гибкий обмен знаниями по сети, включающий в себя обмен статическими и динамическими знаниями, в зависимости от специфики решаемой задачи.

Помимо указанных основных результатов следует отметить, что разработанный для представления знаний язык FMDL обладает богатыми декларативными свойствами, и в общем является более наглядным (хотя, возможно, за счет некоторых потерь в выразительности), чем языки семейства OPS5, используемые в CLIPS и JESS.

Следует отметить, что выбранное фреймовое представление знаний очень удачно подходит для реализации описанных выше целей:

- Фреймовое представление знаний очень похоже на традиционные в настоящее время объектный и компонентный подходы, что позволяет использовать фреймовую модель как некоторый "общий знаменатель" при создании гибридных систем, сочетающих декларативные знания и императивные компоненты. Таким образом объекты и компоненты открытых систем в форме Java-классов, (Enterprise) JavaBeans, CORBA- и

COM-объектов могут быть представлены как фреймы в единой иерархии, и наоборот, любой фрейм иерархии может использоваться как объект, вызываемый из внешней программной системы.

- Фреймы могут эффективно использоваться для доступа к реляционным базам данных, а также к другим типам структуризованной информации (например, для анализа сетей веб-страниц [12] и др.).
- Фреймовое представление знаний предоставляет естественный способ кластеризации знаний, в особенности динамических правил прямого и обратного вывода, вокруг соответствующих фреймов в виде процедур-демонов и процедур-запросов, что в свою очередь обеспечивает естественное распределение знаний между различными узлами. Представление знаний в виде иерархических структур находит отражение в виде моделей онтологий [12,13], и модель распределенной фреймовой иерархии представляет собой один из способов реализации таксономических онтологий, совмещенных с традиционной моделью логического вывода и представления знаний. Методология создания распределенных баз знаний на основе таксономической онтологии заслуживает отдельного рассмотрения, в частности, с позиций адаптации существующих объектных методологий проектирования программных систем.

Таким образом, в инструментарии JULIA реализован целый спектр подходов к созданию гибридных распределенных интеллектуальных систем, что позволяет эффективно использовать его в различных приложениях, ориентированных на распределенное накопление и использование знаний. Безусловно, инструментарий может эффективно использоваться и для создания локальных (функционирующих на одной ЭВМ) экспертных систем, распределенных систем масштаба предприятия функционирующих на единой шине CORBA, распределенных систем информационно-интеллектуальной поддержки виртуальных корпораций, гетерогенных Интернет-приложений и др.

В то время как агентный подход в настоящее время находится в стадии бурного развития, нам кажется, что альтернативный подход, основанный на более классическом логическом выводе и явном представлении знаний будет дополнять агентные системы, в ряде случаев оказываясь более удобным для применения. Кроме того, создание расширяемой гибридной многоплатформенной библиотеки для построения интеллектуальных систем, основанной на открытых стандартах (XML, CORBA, HTTP, KQML), само по себе является весьма полезной задачей, находящей множество применений на практике.

Список литературы

1. Soshnikov D. An Approach for Creating Distributed Intelligent Systems. // Proceedings of the 1st International Workshop on Computer Science and Information Technologies. Moscow: Mephi Publishing, 1998. — pp. 129-134.
2. CLIPS: A Tool for Building Expert Systems. — http://www.ghgcorp.com/clips/CLIPS.html (23.10.2001)
3. Jess, the Expert System Shell for the Java Platform. — http://herzberg.ca.sandia.gov/jess/main.html (23.10.2001)
4. AMZI Prolog — http://www.amzi.com (23.10.2001)
5. Hanson N.H.,Widom J. An Overview of Production Rules in Database Systems. // The Knowledge Engineering Review. — 1993, V.8, №2. — pp.121-143.
6. Ramakrishnan K., Ullman, J. A Survey of Research on Deductive Database Systems. // Journal of Logic Programming. — 1995, 23(2). — pp. 125-149.
7. Dobson S. A., Burrill V. A. Lightweight databases. // Computer Networks and ISDN Systems. — 1995, V.27, No.6. — pp. 1009-1015.
8. Сошников Д.В. Построение распределенных интеллектуальных систем на основе распределенной фреймовой иерархии.// Информационные технологии в образовании. Международная научно-практическая конференция. Шахты. 2001: Тез. докл. — Шахты, 2001.
9. Искусственный интеллект: применение в интегрированных производственных системах.— М.: Машиностроение, 1991.
10. Hyacinth S. Nwana, Software Agents: An Overview. // Knowledge Engineering Review. — 1996, V. 11, No.3. — pp. 1-40.
11. Tim Finin, Yannis Labrou, James Mayfield. KQML as an agent communication language. // Software Agents. — Cambridge: MIT Press, 1997.
12. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. — СПб.: Питер, 2000.
13. Gruber T. R., A Translation Approach to Portable Ontology Specifications. // Knowledge Acquisition. — 1993, 5(2). — pp. 199-220.
14. Minsky M. A Framework for Representing Knowledge. — Cambridge: MIT Press, 1974.
15. Pfleger K., Hayes-Roth B., An Introduction to Blackboard-Style Systems Organization. // KSL Technical Report KSL-98-03. — Stanford University, 1997.
16. Soshnikov D. Software Toolkit for Buiding Embedded and Distributed Knowledge-Based Systems. // Proceedings of the 2nd International Workshop on Computer Science and Information Technologies. (Vol.1). — Ufa: USATU Publishing, 2000. — pp. 103-111.

17. Orfali R., Harkey D., Edwards J. Instant CORBA. — Wiley Computer Publishing, 1997.
18. XML Web Site. — http://www.xml.com
19. Charles L. Forgy. RETE: A fast algorithm for the many pattern / many object pattern match problem. // Artificial Intelligence. — 1982, V.19, №1. pp. 17-37.
20. Daniel P. Miranker. TREAT: A New and Efficient Match Algorithm for AI Production Systems. — San Mateo: Morgan Kaufmann Publishers, Inc., 1990.
21. Lesser, V.R. An Overview of DAI: Viewing Distributed AI as Distributed Search. // Journal of Japanese Society for Artificial Intelligence. — 1990, V. 5, № 4.
22. Heinz Worn, Thomas Laengle, Martin Albert. Distributed Diagnosis for Automated Production Cells. // Proceedings of the 2nd International Workshop on Computer Science and Information Technologies. (Vol.1). — Ufa: USATU Publishing, 2001.
23. Seng Wai Loke. Adding Logic Programming Behaviour to the World Wide Web: PhD Thesis.// Department of Computer Science, The University of Melbourne — Melbourne: 1998.
24. Soshnikov D. Technologies for Building Intelligent Web Applications based on JULIA Toolkit. // Proceedings of the 2nd International Workshop on Computer Science and Information Technologies. (Vol.1). — Ufa: USATU Publishing, 2001.
25. Сошников Д.В., Лукьянов И.В., Заведеев И.А. Интеллектуальная учетно-диагностическая программа для учета и диагностики больных заболеваниями предстательной железы.// 10-ая межд. конф. по вычислительной механике и современным прикладным программным системам, Переславль-Залесский, 1999: Тез. докл. — М.: МГИУ, 1999. — с.326-328.
26. Крастелева И.Е., Сошников Д.В. Исследование процесса продвижения интернет-ресурсов с использованием интеллектуальных технологий.// Новые информационные технологии. 9-ая международная студенческая школа-семинар, Судак, 2001: Тез. докл. — М.: МГИЭМ, 2001. — с.392-394.
27. Малкина О.И., Сошников Д.В. Создание интерактивных систем адаптивного тестирования в среде Интернет с использованием технологий искусственного интеллекта.// Новые информационные технологии. 9-ая международная студенческая школа-семинар, Судак, 2001: Тез. докл. — М.: МГИЭМ, 2001. — с.390-392.
28. Лукьянов И.В., Машкович В.Э., Заведеев И.А., Сошников Д.В. Выбор оптимального метода трансуретрального лечения больных с инфравезикальной обструкцией.// Современные эндоскопические технологии в урологии. 1-ая Всеросийская научно-практическая конференция. Тез. докл. — Уральская медицинская академия, 1999.
29. Сошников Д.В. Инструментальные средства для построения встраиваемых продукционных экспертных систем.// 10-ая юбилейная международная конференция по вычислительной

механике и современным прикладным программным системам, Переславль-Залесский, 1999: Тез. докл. — М.: МГИУ, 1999. — с.325-326.

Сведения об авторе

Сошников Дмитрий Валерьевич, аспирант кафедры вычислительной математики и программирования Московского государственного авиационного института (технического университета);
Телефон: 158-4983, e-mail: dmitri@soshnikov.com