Лекция 3.

Раздел 3. Средства реализации СИИ: языки программирования, оболочки и среды разработки

Тема 7. Классификация инструментария СИИ.
Тема 8. Языки, надстройки, оболочки, среды разработки. Языки Лисп и Ским. Язык Пролог.

План лекции.

1. Инструментарий СИИ.
2. Языки Лисп (Lisp) и Ским (Scheme).
3. Язык Пролог (Prolog).

Основная часть.

Программные средства инженерии знаний и реализации интеллектуальных информационных систем (ИИС) можно разделить на следующие группы:

• универсальные языки программирования (в том числе традиционные);

• универсальные языки представления знаний и оболочки.

ИИС представляют собой некоторый программный комплекс, позволяющий решать производственные и экономические задачи на уровне человека – оператора или управленца (эксперта). Однако очевидно, что любую программу можно написать на машинно–ориентированном языке (ассемблере) или на универсальном языке высокого уровня (ПЛ/1, Си, Бейсик, Алгол, Ада, Фортран, Паскаль и т.д.). В этой связи возникает вполне справедливый вопрос: зачем рассматривать специализированные средства, для изучения которых требуется определенное время, если универсальным языком высокого уровня (либо языком ассемблера) владеет практически любой программист? Ответ на этот вопрос взят из практики: процесс программирования систем ИИ на специализированных средствах занимает в 2-3 раза меньше времени, чем на универсальных. Однако следует всегда помнить, что параметры эффективности (объем памяти и быстродействие) ИИС, реализованных на базе специализированных средств, в большинстве случаев ниже, чем при реализации ИИС на универсальных средствах.

Еще одним фактором, существенным для выбора ИИС инструментальных программных средств при разработке ИИС, является потенциальная возможность взаимодействия с программными средствами, используемыми на различных уровнях иерархии интегрированных корпоративных информационных систем.

Оптимальным решением задачи выбора программных средств для реализации ИИС следует считать следующее: первый прототип (или прототипы: исследовательский, демонстрационный) реализуется на специализированных средствах. В случае достаточной эффективности этих средств на них могут быть написаны действующий прототип, и даже промышленная система. Однако в большинстве случаев прототип следует «переписать» на традиционных программных средствах.

Рассмотрим наиболее известные и широко применяемые программные средства интеллектуальных систем.

Язык ЛИСП – один из наиболее распространенных языков искусственного интеллекта. После появления ЛИСПа различными авторами был предложен ряд других алгоритмических языков, ориентированных на решения задач в области искусственного интеллекта, среди которых можно отметить Плэнер, Снобол, Рефал, Пролог, Смолтолк.

Язык ЛИСП (LISP) был разработан в 1958 году американским ученым Джоном Маккарти как функциональный язык, предназначенный для обработки списков (Английское название LISP, являющееся аббревиатурой выражения LISt Processing, обработка списков, хорошо подчеркивает основную область его применения). Lisp - означает "лепетать".

В основе языка лежит математический аппарат:

- лямбда-исчисление Черча
- алгебра списочных структур
- теория рекурсивных функций

Долгое время язык использовался узким кругом исследователей. Широкое распространение язык получил в конце 70-х - начале 80-х годов с появлением необходимой мощности вычислительных машин и соответствующего круга задач. В настоящем - Лисп одно из главных инструментальных средств систем искусственного интеллекта. Он принят как один из двух основных ЯП для министерства обороны США (система AutoCAD разработана на Лиспе). Популярность ЛИСПа объясняется следующими причинами:

1. ЛИСП ориентирован на работу с символьной информацией, а процесс решения большинства задач искусственного интеллекта сводится к обработке такой информации.

2. ЛИСП представляет собой интерпретирующую систему, а это позволяет значительно облегчить и ускорить процесс создания сложных программных комплексов в интерактивном режиме.

3. Идеология ЛИСПа проста: данные и программы представляются в нем в одной и той же форме. Благодаря такой унификации представления данные могут

интерпретироваться как программа. А любая программа может быть использована как данные любой другой программой.

4. Язык ЛИСП является языком функционального программирования. Применение функциональных языков открывает широкие перспективы, позволяя пользователю описывать скорее природу своих задач, чем способ их решения.

Язык ЛИСП имеет множество диалектов, некоторые специалисты склонны отдавать предпочтение диалекту Common Lisp в качестве будущего стандарта языка ЛИСП.

Язык ЛИСП может служить основой для обучения методам искусственного интеллекта, исследованиям и практическому применению в этой области.

Основные особенности языка ЛИСП. От других языков программирования ЛИСП отличается следующими свойствами:

1. *Одинаковая форма данных и программ.* И то и другое представляется списочной структурой, имеющей одинаковую форму. Таким образом, программы могут обрабатывать и преобразовывать другие программы и даже самих себя. В процессе трансляции можно введенное и сформированное в результате вычислений выражение данных проинтерпретировать в качестве программы и непосредственно выполнить. Универсальный единообразный и простой ЛИСПовский синтаксис списка не зависит от применения, представления и абстракции[1].

2. *Хранение данных, не зависящее от места.* Списки, представляющие программы и данные, состоят из списочных ячеек, расположение и порядок которых не существенны. Структура списка определяется логически на основе имен символов и указателей. Добавление новых элементов в список или удаление из списка может производиться без переноса списка в другие ячейки памяти. Резервирование и освобождение могут в зависимости от потребности осуществляться динамически, ячейка за ячейки.

---

[1] Абстракция (от латинского abstraction – отвлечение) – это отвлечение в процессе познания от несущественных сторон, свойств, связей объекта (предмета, явления) с целью выделения их существенных, закономерных признаков. Теоретическое обобщение является результатом абстрагирования, например, цвет, красота, кривизна. В программировании (объектно-ориентированном) абстракция – использование только тех характеристик объекта, которые с достаточной точностью представляют его в данной системе. Идея заключается в том, чтобы представить объект минимальным набором полей, методов и при этом с достаточной точностью для решаемой задачи.

3. *Автоматическое и динамическое управление памятью.* Пользователь не должен заботится об учете памяти, так как система резервирует и освобождает память автоматически в соответствии с потребностью. Когда памяти не хватает, запускается специальный «мусорщик». Он перебирает все ячейки и собирает являющиеся мусором ячейки в список свободной памяти для того, чтобы их можно было использовать заново. В современных ЛИСП-системах выполнение операции сбора мусора занимает от одной до нескольких секунд. В задачах большого объема сборщик мусора запускается весьма часто, что резко ухудшает временные характеристики прикладных программ. Во многих системах мусор не образуется, поскольку он сразу же учитывается. Управление памятью не зависит от физического расположения, поскольку свободная память логически состоит из цепочки списочных ячеек.

В первую очередь данные обрабатываются в оперативной и виртуальной памяти, которая может быть достаточно большой. Файлы используются в основном для хранения программ и данных в промежутках между сеансами.

4. *Функциональная направленность.* В результате каждого действия возникает значение. Значения становятся элементами следующих действий и конечный результат всей задачи выдается пользователю, т.е. результатом вычислений могут быть новые функции. Обойти это можно только при помощи специальной функции QUOTE.

5. *ЛИСП – бестиповый язык.* В ЛИСПе имена символов, переменных, списков, функций и других объектов не закреплены предварительно за какими-нибудь типами данных. Типы не связаны с именами объектов данных, а сопровождают сами объекты. Переменные в разные моменты времени могут представлять разные объекты, поэтому ЛИСП в этом смысле и является бестиповым языком.

Динамическая проверка типа (осуществляемая лишь в процессе исполнения) и связывание допускают разностороннее использование символов и гибкую модификацию программ. Функции можно определять практически независимо от типов данных, к которым они применяются.

Одним из общих принципов развития ЛИСП-систем было свободное включение в язык новых возможностей и структур, если считалось, что они найдут более широкое применение. Это было возможно в связи естественной расширяемостью языка.

В более новых ЛИСП-системах возможно факультативное определение типов. В этом случае транслятор может использовать эту информацию для оптимизации кода. В ЛИСП-машинах проверка осуществляется на уровне аппаратуры.

6. *Интерпретирующий и компилирующий режимы работы.* Программы не нужно транслировать, их можно исправлять в процессе исполнения. Если какой-то участок

программы отлажен и не требует изменений, то его можно оттранслировать, тогда он выполняется быстрее. В одной и той же программе могут быть транслированные и интерпретированные функции. Транслирование по частям экономит усилия программиста и время вычислительной машины.

Однако компилирующий режим предусмотрен далеко не во всех ЛИСП-системах, его использование накладывает ряд дополнительных ограничений на технику программирования.

7. *Пошаговое программирование.* Программирование и тестирование программы осуществляется функция за функцией, которые последовательно определяются и тестируются. Написание, тестирование и исправление программы осуществляется внутри ЛИСП-системы без промежуточного использование ОС.

Вспомогательные средства, такие, например, как редактор, трассировщик, транслятор и другие образуют общую интегрированную среду, язык которой нельзя отличить от системных средств. Отдельные средства по своему принципу являются прозрачными, чтобы их могли использовать другие средства. Часто работа может производиться на разных уровнях или в различных рабочих окнах. Такой способ работы особенно хорошо подходит для исследовательского программирования и быстрого построения прототипов.

8. *Единый системный и прикладной язык программирования.* ЛИСП является одновременно как языком прикладного, так и системного программирования. Он напоминает машинный язык тем, что как данные, так и программы представлены в одинаковой форме. Язык превосходно подходит для написания интерпретаторов и трансляторов как для него самого, так и для других языков.

Scheme (Ским) - диалект языка программирования Лисп. Scheme был разработан Гаем Стилом (Guy L. Steele) и Джеральдом Сассменом (Gerald Jay Sussman) в 1975 году и представлен в серии статей, так называемых Lambda Papers, в последующие пять лет. Основные особенности Scheme:

- минимализм языка, основанный на лямбда-исчислении, которое используется для получения значительной части синтаксиса языка (11 из 23 синтаксических конструкций) из более примитивных конструкций.
- статическая лексическая область видимости: имя переменной относится к самой локальной области видимости; таким образом, код можно читать и интерпретировать вне зависимости от того, в каком контексте он будет вызываться.
- блоки, выражающиеся тремя конструкциями let, let* и letrec.

- “правильная” хвостовая рекурсия, позволяющая записывать итеративные алгоритмы более идиоматично, через рекурсию, и при этом оптимизирующая их так, чтобы поддерживать неограниченное количество активных вызовов.
- продолжения (абстрактные представления состояний программы) как объекты первого класса (процедура call-with-current-continuation).
- общее пространство имен для переменных и процедур.

При разработке Scheme упор был сделан на простоту языка. Философия языка подчёркнуто минималистская. Его цель - не сваливать в кучу разные полезные конструкции и средства, а напротив - удалить слабости и ограничения, вызывающие необходимость добавления в язык новых возможностей. В результате, Scheme содержит минимум примитивных конструкций и позволяет выразить все, что угодно путём надстройки над ними.

В качестве базовых структур данных язык использует списки и одномерные массивы («векторы»). В соответствии с декларируемым минимализмом, (пока) нет стандартного синтаксиса для поддержки структур с именованными полями.

Несколько базовых принципов языка.

1. Круглые скобки. Любое законченное выражение должно быть заключено в них. Это отпугивает тех, кто видит код впервые, но впоследствии на практике не вызывает сложностей.
2. Никаких дополнительных служебных символов. Хватит скобок. Точка с запятой, кстати, отделяет от кода комментарии.
3. Построение конструкций по типу «действие-предмет». В языке программирования это смотрится необычно, но в переводе на естественный язык вполне понятно.

Фрейм-ориентированный язык FRL. Одним из известных языков представления знаний является язык FRL (Frame Representation Language), относящийся к классу фрейм - ориентированных. Основная единица знания в таких языках – фрейм, представляющий собой информационную модель (или описание) некоторой стереотипной ситуации. «Стереотипная ситуация», является обобщением таких понятий, как действия, процессы, события, объекты, свойства, модификаторы и т.д.

Фрейм в FRL – это совокупность поименованных, ассоциативных списков, содержащая до пяти уровней подструктур. Подструктурами фреймов могут быть слоты, аспекты, данные, комментарии и сообщения. Основной структурной единицей во фрейме являются слоты, отражающие взаимосвязи между понятиями предметной области. Слот

характеризуется своим именем и значением. Имена слотов назначаются проектировщиками БЗ. Однако FRL имеет также и зарезервированные имена слотов: АКО, INSTANSE, CLAS SIFICATON. В качестве значения слотов могут выступать числа, символы, имена других фреймов, имена процедур.

Фреймы в FRL строятся с помощью процедуры FASSERT.

В FRL имеется семь зарезервированных аспектов: VALUE, DEFAULT, IF-NEEDED, IF-ADDED, IF-REMOVED, ¤IF-INSTANTIAD, REQUIRE. Данные из аспекта VALUE интерпретируются как значение слота, а из аспекта DEFAULT – как значение по умолчанию. Остальные пять аспектов связывают с фреймом процедуральные знания. Процедуры из аспекта IF-ADDED активизируются в том случае, если в слот добавлено новое данные; из аспекта IF-REMOVED – если из слота удаляется те или иные данные. Процедуры из аспекта IF-NEEDED запускаются при создании экземпляров фрейма. Аспект REQUIRE содержит процедуры, которые ограничивают значение слота.

Важным свойством FRL является наличие в нем встроенного механизма «наследования свойств». Суть этого механизма заключается в следующем. Все понятия предметной области в БЗ организовываются в виде иерархической классификационной системы, где каждое понятие связывается с помощью специальных отложений с более конкретными. Для реализации этих отложений существуют слоты АКО и INSTANSE. Слот АКО связывает понятие с более общим (родовым). Слот INSTANSE связывает понятие с более конкретным (видом). Свойства присущие всему классу, описывают только во фрейме класса, а остальные фреймы этого класса могут наследовать это свойство в случае надобности.

Процедуры обработки FRL подразделяются на независимые и присоединенные. Независимо от типа эти процедуры пишутся обычно на языке реализации самого FRL. На сегодняшний день большинство FRL – систем написаны на LISP.

Язык Пролог. *Основной принцип использования языка Пролог* состоит в том, что нужно подробно, на логически точном языке, описать условие задачи. Решение ее получается в результате определенного рутинного процесса, который исполняется компьютером. В этом заключается принципиальное отличие Пролога от традиционных языков программирования, которые требуют описания того, как должен быть вычислен результат, или другими словами, требуют описания процедуры решения задачи.

При программировании на Прологе усилия программиста должны быть направлены на описание логической модели фрагмента предметной области решаемой задачи в терминах объектов предметной области, их свойств и отношений между собой, а не деталей

программной реализации. Фактически Пролог представляет собой не столько язык для программирования, сколько язык для описания данных и логики их обработки. Программа на Прологе не является таковой в классическом понимании, поскольку не содержит явных управляющих конструкций типа условных операторов, операторов цикла и т. д. Она представляет собой модель фрагмента предметной области, о котором идет речь в задаче. И решение задачи записывается не в терминах компьютера, а в терминах предметной области решаемой задачи, в духе модного сейчас объектно-ориентированного программирования.

Существенной возможностью логических языков программирования является интерпретация типов алгоритмов обработки информации: линейного, разветвляющегося и циклического, а также механизма подпрограмм.

Прологу присущ ряд механизмов, которыми не обладают традиционные языки программирования: сопоставление с образцом, вывод с поиском и возвратом. Еще одно существенное отличие заключается в том, что для хранения данных в Прологе используются списки, а не массивы. В языке отсутствуют операторы присваивания и безусловного перехода, указатели. Естественным и зачастую единственным методом программирования является *рекурсия.* Поэтому часто оказывается, что люди, имеющие опыт работы на процедурных языках, медленней осваивают *декларативные языки*, чем те, кто никогда ранее программированием не занимался, так как Пролог требует иного стиля мышления, отказа от стереотипов императивного программирования.

Основные приложения, которые связывают с языком Пролог это, как правило, *разработки в области искусственного интеллекта и экспертных систем.*

Еще одним применением Пролога является интерпретация трансляторов с языков программирования. С помощью выразительных средств синтаксиса этого языка удается достичь высокой степени лаконичности изложения, это делает возможным описать логику работы интерпретатора с помощью нескольких строк текста.

Области, для которых Пролог не предназначен: большой объем арифметических вычислений (обработка аудио, видео и т.д.); написание драйверов.

Подробное описание языков Lisp, Scheme, Prolog см. в курсе лекций по дисциплине «Функциональное и логическое программирование» в разделе 3 (лекция 3) «Основы функционального программирования на языках Lisp, Scheme и FP» и разделе 4 (лекция 4) «Основы логического программирования на языке Prolog».

Продукционный язык OPS. Язык относится к числу продукционных. Являясь универсальным языком программирования, он в первую очередь предназначен для разработки систем ИИ, и, в частности экспертных систем. Идеология языка OPS нашла отражение в целом ряде практических реализаций, достаточно сильно отличающихся друг от друга. Одной из первых и наиболее известной является реализация OPS-5, выполненная на одной из версий Лиспа (Franz LISP). Поэтому синтаксис OPS –5 максимально приближен к синтаксису Лиспа. На языке OPS – 5 создан ряд промышленно эксплуатируемых экспертных систем для фирмы DEC с объемом баз знаний от 1000 до 5000 правил. Одной из последних, но уже достаточно широко известной реализацией является OPS-83. Особенности этой реализации – наличие некоторых конструкций, характерных для процедурных языков программирования, а также сильная типизация данных.

Говоря об общих отличительных чертах семейства языков OPS, необходимо отметить наличие:

- программного управления стратегий вывода решений;
- развитой структуры данных и принципиальной эффективности реализации.

Язык OPS имеет типичную для продукционных систем архитектуру, включающую в себя базу правил, рабочую память и механизм вывода. База правил состоит из неупорядоченной совокупности правил, рабочая память – из дискретных объектов, называемых элементами рабочей памяти. Элемент рабочей памяти может быть добавлен в рабочую память, удален из нее или модифицирован. Механизм вывода является стандартным для системы продукций циклом управления. На первой фазе цикла выбираются все правила, левые части которых сопоставились с содержимым рабочей памяти. На второй фазе правило выполняется. Встроенный в OPS механизм вывода непосредственно поддерживает только прямой вывод, однако в языке имеются средства для организации обратного и смешанного выводов.

В языке OPS допускается использование внешних процедур, реализованных на других языках программирования. Эффективность в языке OPS достигается, во–первых, за счет использования специального алгоритма быстрого сопоставления, а во-вторых, за счет компилирующей схемы, применяемой взамен более традиционной для продукционных языков интерпретирующей. Применение алгоритма быстрого сопоставления накладывает ряд ограничений, однако опыт эксплуатации языка OPS показал достаточно высокую адекватность средств языка для разработки экспертных систем.

Программа на языке OPS состоит из декларативной и продукционной частей. Вообще говоря, язык OPS очень прост: в нем всего три вида операторов: оператор описания типов данных TYPE, оператор описания классов CLASS и оператор описания правил RULE.

Декларативная часть программы содержит описание типов данных и классов элементов рабочей памяти. Элемент рабочей памяти (класс) является единственно возможным представлением данных в OPS программе. Он представляет собой фиксированную структуру, состоящую из совокупности пар «атрибут-значение» вида:

Класс

Атрибут – 1

значение

Атрибут –2

значение

Например, элемент памяти «Установка 1 предприятия» имеет вид:

ОБЪЕКТ

Имя

Установка 1

Входит в

Цех 1

Состоит из

(колонна 1, колонна 2, колонна 3)

Получает сырье от

Резервуар 5

Тип сырья

Нефть

Вырабатывает

Керосин

Поставляет продукцию

(установка 3, установка 4).

Элементарным объектом данных, имеющим значение, является атрибут. Атрибуты локализованы в пределах одного класса, некоторые атрибуты могут не иметь значения.

Каждый элемент рабочей памяти относится к определенному классу. Допустимые классы элементов рабочей памяти должны быть предварительно описаны в разделе определения классов. Определение класса задает его структуру, типы допустимых значений атрибутов, входящих в класс, и, если это необходимо, начальные значения атрибутов (значения по умолчанию).

Отметим, что все используемые типы данных (кроме встроенных) должны быть явно описаны в операторах TYPE или CLASS; особенно удобными при программировании экспертных систем являются произвольные типы данных, которые позволяют следить за

правильностью вводимых пользователем данных и допустимых значений в процессе логического вывода.

Перейдем теперь к рассмотрению продукционной части OPS программы – раздела правил. Он записывается после декларативной части и представляет собой совокупность правил. Правило OPS состоит из заголовка и тела правила. Заголовок правила начинается со слова RULE, за которым следует имя правила и описание переменных (если они используются). Тело правила состоит из левой части, задающей условие применимости правила, и правой части, содержащей последовательность выполняемых действий. Левая часть начинается словами IF, разделителем между левой и правой частями служит слово THEN. Ниже приведен пример правила OPS:

```
RULE ОСТАНОВКА УСТАНОВКИ 1
IF
  ОБЪЕКТ
      ИМЯ              УСТАНОВКА_1
      СОСТОЯНИЕ        ОСТАНОВИЛАСЬ
AND
   ОБЪЕКТ
      ИМЯ              УСТАНОВКА_2
      СОСТОЯНИЕ        РАБОТАЕТ
AND
ОБЪЕКТ
      ИМЯ              УСТАНОВКА_3
      СОСТОЯНИЕ        РАБОТАЕТ
THEN
WRITE ∇ Установка 1 остановилась∇
WRITE ∇ Распределить ее сырье между установкой 2 и установкой 3 ∇
```

Левая часть правила состоит из одного или более условных элементов. Каждый условный элемент задает образец, который сопоставляется с содержащимися в рабочей памяти элементами. Левая часть считается сопоставленной, если одновременно сопоставились все входящие в нее условные элементы. Разделителем между условными элементами служит слово AND или ANDNUT. В левой части могут быть использованы следующие элементарные предикатное операторы: «не равно», «больше», «меньше», «не больше», «не меньше», «входит», «не входит в список», «имеет/не имеет значение» и др. Правая часть правила состоит из последовательности императивных утверждений, называемых действиями. Действия, имеющиеся в OPS, разделяются на две группы:

элементарные, обеспечивающие вывод решения, и вспомогательные, обеспечивающие ввод и другие сервисные возможности. К элементарным действиям относятся:

MARE – создание нового элемента рабочей памяти;

REMOVE – удаление элемента из рабочей памяти;

MODIFY – изменение значений атрибутов, уже находящихся в рабочей памяти

К вспомогательным действиям относятся:

HALT – явное прекращение программы;

WRITE – выдача сообщения на терминал;

PRINT – печать сообщения на печатающем устройстве;

DISPLAY – вывод на экран терминала информации из библиотеки на магнитном диске;

BIND – вызов функций или модулей на других языках программирования;

SET – динамическое изменение стратегии вывода решений или подробности объяснений решений.

Рассмотрим современные разработки средств построения интеллектуальных систем.

Объектно-ориентированный язык Visual Basic. Visual Basic язык, **поддерживающий событийно-управляемое** программирование (event-driven programming): визуальное проектирование и элементы объектно-ориентированного программирования. Выпустив в 1991 г. первую версию VB, Microsoft достаточно скромно оценивала возможности этой системы, ориентируя ее, прежде всего, на категорию начинающих и непрофессиональных программистов. Основной задачей тогда было выпустить на рынок простой и удобный инструмент разработки в тогда еще довольно новой среде Windows, программирование в которой представляло проблему и для опытных специалистов,

В 1992 г. была выпущена вторая версия, а в 1993-94 гг. - третья версия. Эта версия позволила продукту войти в число серьезных инструментальных средств программирования и значительно расширить свой рынок.

1996-97 гг. была выпущена пятая версия. В VB5 было много усовершенствований, он обеспечивал заметно более высокую производительность и предлагал долгожданный компилятор, преобразующий программу во внутренний машинный код.

В 1998 г. появился Visual Basic 6.

В течение нескольких лет идут постоянные дебаты о том, может ли Visual Basic считаться языком объектно-ориентированного программирования (ООП). С одной стороны, элементы ООП в нем были всегда, и их число росло от версии к версии. С другой - многих нужных возможностей ООП в Visual Basic не было. Появление Visual Basic.NET

должно положить конец всем этим дискуссиям, так как в нем будут реализованы все необходимые атрибуты ООП. Напомним, что модель ООП подразумевает наличие трех обязательных механизмов инкапсуляции, полиморфизма и наследования. Первые два были реализованы в предыдущих версиях и получили развитие в новой, а последний появится в ней впервые.

Visual Basic наконец-то стал полноценным объектно-ориентированным языком. Безусловно, Visual Basic.NET серьезно прибавил в мощности средств, но работать с ним будет сложнее. Ведь объектно-ориентированные методы программирования предъявляют более серьезные требования к квалификации разработчика, на которою перекладываются многие проблемы обеспечения работоспособности программы.

Дадим также описание некоторых новых элементов языка на концептуальном уровне.

Web Services - это некая принципиально новая платформо-независимая технология, связанная с использованием стандарта XML и протокола SOAP (Simple Object Access Protocol - протокол доступа к простым объектам), которая будет широко интегрирована в средства разработки. Ключевая идея состоит в создании компонентов уровня бизнес-логики**,** которые взаимодействуют с внешними объектами с помощью стандартных Web-протоколов.

Object-oriented: Для того, чтобы называться объектно-ориентированным, язык должен удовлетворять трем критериям:

1. Поддерживать инкапсуляцию(encapsulation). VB делает это с 4-й версии.

2. Поддерживать наследование (inheritance). VB 7 будет иметь полноценное наследование.

3. Поддержка полиморфизма(polymorphism). В VB это работает с 4-й версии.

Итак, в 7 версия будет вполне удовлетворять этим критериям.

Encapsulation **(**Инкапсуляция**).** Идея заключается в том, что вы можете создавать скрытый набор процедур (методов и свойств), которые формируют некий программный интерфейс. Другой код может обращаться к этим методам и свойствам, не вдаваясь в подробности их внутренней реализации.

Free-threaded (Многопоточность). Это комплексная концепция поддержки выполнения более чем одного потока заданий в одно и то же время. Например, пользователь может продолжать работать с приложением, после того как он задал операцию фоновой печати документа. Подобный режим крайне необходим для создания масштабируемых серверных компонентов и может быть полезен для реализации пользовательского

интерфейса. Создание таких вычислительных потоков выполняется примерно следующим образом:

Inheritance (Наследование). Это одно из ключевых понятий объектно-ориентированного программирования возможность использования (в том числе расширения) поведения чужого объекта. Упрощенно говоря, можно создать объект Продукт, а затем на его основе объекты Программный Продукт и Технический Продукт. Оба новых объекта будут наследовать свойства и методы объекта Продукт, и при этом вы сможете изменить поведение наследующего объекта. Visual Basic-разработчики теперь могут использовать ключевое слово Inherits для подключения процедур уже существующего класса:

Overloading. В русском языке нет соответствующего термина в данном контексте: использование одного и того же идентификатора для обозначения разных процедур. Выбор нужной процедуры выполняется в зависимости от числа и типа параметров. Это особенно полезно для создания одного свойства, поддерживающего разные типы аргументов.

Polymorphism (Полиморфизм). Возможность иметь несколько объектов разного типа, но с одинаковыми методами. Это позволяет писать код, вызывающий тот метод, который нужен в зависимости от используемого в данный момент объекта.

Structured Exception Handling (Структурная обработка особых ситуаций). Это новая структура для обработки ошибок, уже реализованная во многих языках. Она должна заменить старую и весьма негибкую (точнее, ненаглядную) конструкцию On Error Goto|Resume|Next. Новый блок содержит ключевые слова Try, Catch, Finally:

Type Safety (Контроль типов данных). Запрет неявного преобразования типов с помощью нового оператора Option Strict. Кому-то из программистов это не понравится, так как данный режим заставляет задуматься о типах переменных и использовать специальные функции при присвоении переменной значения другой переменной другого типа. Но это совершенно необходимо, если вы хотите писать надежные программы и снизить затраты на отладку приложений.

User Interface Inheritance (Наследование пользовательского интерфейса). VB7 будет включать наследование форм, т.е. создание новых форм на основе некоторых шаблонов. В отличие от существующего сегодня механизма подключения новых форм на основе шаблонов, в данном случае будет автоматически поддерживаться механизм наследования: изменения в родительском шаблоне (например, корпоративном логотипе) будут отражаться 1 в дочерних формах.

Возможности языка Visual Basic для создания экспертных систем. В основу программного пакета было положено передовое архитектурное решение, позволяющее не

писать, а проектировать программы, подобно инженеру-дизайнеру. Иначе говоря, в нем был одним из первых реализован популярный ныне стиль визуального программирования. И ключевым словом в названии языка является Visual -экранные формы и множество встроенных компонент (текстовые, графические окна, кнопки, диалоги и т. п.) избавляют от сложностей, связанных с выводом, обработкой, обновлением всех этих элементов, что особенно важно при разработке экспертной системы, для которой легкая модифицируемость является чуть ли не наиболее важной отличительной чертой.

В Visual Basic реализовано "управление от событий", как уже отмечалось ранее, что позволяет разработчику спроектировать интерфейс пользователя максимально удобно для пользователя. После этапа визуального проектирования программист просто пишет программный код для обработки связанных с объектом событий, VB предоставляет удобную среду для разработки приложений, тестирования их работы и нахождения и исправления ошибок.

Рассмотрим основные принципы, положенные в основу Visual Basic и позволяющие максимально облегчить труд разработчика и сделать работу с написанным приложением максимально комфортным.

Объект одно из основных понятий Visual Basic и объектно-ориентированного программирования. Объектом называется некая сущность, которая, во-первых, чётко проявляет своё поведение, а во-вторых, является представителем некоторого класса подобных объектов.

Классом объектов в объектно-ориентированных языках называется общее описание таких объектов, для которых характерно наличие множества общих свойств и действий.

СОБЫТИЕ - действие или ситуация, связанная с объектом, например: щелчок мыши или нажатие клавиши. События могут инициироваться в программном коде приложения или непосредственно в системной среде.

СВОЙСТВА - определяют представление, поведение и другие черты объекта. Цвет фона формы, строка соединения (в сеансе БД), размер элементов и т.п. - все это свойства тех или иных объектов.

Свойства: Caption - имя, Enabled - доступ, Height - высота.

МЕТОДЫ - программные процедуры, которые выполняют некоторую обработку, связанную с объектом, определенные действия над объектом.

Объектно-ориентированная модель аналогична во многом фреймовой модели, реализуя обмен сообщениями между объектами, в большей степени ориентирована на решение динамических задач и отражение поведенческой модели. Отличия от фреймовой

модели заключаются в чётком понятии класс объектов и экземпляр объекта, а также в способе активации процедур объектами.

Перечислим и дадим краткую характеристику тем средствам VB которые помогают в разработке профессиональных коммерческих приложений вообще и экспертных систем в частности.

Пакет Microsoft Visual Basic представляет собой идеальную платформу для создания интерфейсов с локальными базами данных и базами данных типа клиент/сервер в среде Windows.

Visual Basic позволяет создавать различные приложения для работы с базами данных - от простейших локальных баз данных до многоуровневой архитектуры клиент/сервер, а также приложений для работы в intranet и Internet с использованием таких передовых технологий, как DHTML, XML и ASP.

Также очень важны вопросы проектирования реляционных баз данных и использования языка SQL. Язык структурированных запросов является стандартным средством для работы с базами данных и может использоваться как для интерактивной работы с базами данных, так и включаться в языки программирования. Применительно к Visual Basic SQL позволяет:

- создавать, модифицировать или удалять таблицы в базе данных Access;
- вставлять, удалять или модифицировать записи таблиц;
- получать сводную информацию о данных в таблице;
- поиск данных в одной или более таблицах по запросу.

Экспертная система, как никакая другая, должна предоставлять пользователю максимально "дружелюбный" интерфейс, поскольку она в большинстве случаев является диалоговой и требует от пользователя максимального взаимодействия. Это справедливо в большой степени для так называемых оболочек экспертных систем, правильность настройки и заполнения которых является ключевым фактором в построении адекватно реагирующей экспертной системы. В данной ситуации многое зависит от логики представления информации и управляющих элементов в программе, наличия пользовательского меню и развитой справочной системой.

Visual Basic предоставляет разработчику все средства для создания «графического интерфейса пользователя»:

- возможность построения много документального интерфейса;
- создание пользовательского меню;
- травления ACTIVE-X и многое др.

В заключении приведем некоторые подробности реализации механизма вывода с помощью деревьев.

В ЭС принято представлять процесс логического вывода в виде схемы, которая показывается деревом вывода. В системах, база знаний которых насчитывает сотни правил, весьма желательным является использование какой-либо стратегии управления выводом, позволяющей минимизировать время поиска решения и тем самым повысить эффективность вывода. К числу таких стратегий относятся поиск в глубину, поиск в ширину, разбиение на подзадачи.

Методы слепого перебора, полного перебора или поиска в глубину являются исчерпывающими процедурами поиска путей к целевой вершине. В принципе эти методы обеспечивают решение задачи поиска пути, но часто эти методы невозможно использовать, поскольку при переборе придется раскрыть слишком много вершин. Прежде чем нужный путь будет найден. Для многих задач можно сформулировать правила, позволяющие уменьшить объем перебора. Все такие правила, используемые для ускорения поиска, зависят or специфической информации о задаче, представляемой в виде графа. Будем называть такую информацию эвристической информацией (помогающей найти решение) и называть использующие ее процедуры поиска эвристическими методами поиска. Нам необходима мера, которая позволяла бы оценивать "перспективность" вершин. Такие меры называют оценочными функциями. Оценочная функция должна обеспечивать возможность ранжирования вершин- кандидатов на раскрытие- с тем, чтобы выделить ту вершит', которая с наибольшей вероятностью находится на лучшем пути к цели.

Visual Basic позволяет разработчику создавать деревья и осуществлять поиск по ним по любому из вышеперечисленных алгоритмов.

Рассмотрим некоторые общие способы реализации деревьев на языке Visual Basic, Один из способов - создать отдельный класс для каждого типа узлов дерева. Для построения дерева порядка 3 можно определять структуры данных для узлов, которые имеют ноль, один, два или три дочерних узла. Этот подход был бы довольно неудобным. Кроме того, что нужно было бы управлять четырьмя различными классами, в классах потребовались бы какие-то флаги, которые бы указывали тип дочерних узлов. Алгоритмы, которые оперировали бы этими деревьями, должны были бы уметь работать со всеми различными типами деревьев.

В качестве простого решения можно определить один тип узлов, который содержит остаточное число указателей на потомков для представления всех нужных узлов. Назовем это методом *полных узлов,* так как некоторые узлы могут быть большего размера, чем необходимо па самом деле.

Для построения дерева порядка 3 с использованием метода полных узлов, требуется определить единственный класс, который содержит указатели на три дочерних узла.

Если порядки узлов в дереве сильно различаются, метод полных узлов приводит к напрасному расходованию большого количества памяти. Некоторые программы добавляют и удаляют узлы, изменяя порядок узлов в процессе выполнения. В этом случае метод полных узлов не будет работать. Такие динамически изменяющиеся деревья можно представить, поместив дочерние узлы в списки. Есть несколько подходов, которые можно использовать для создания списков дочерних узлов. Наиболее очевидный подход заключается в создании в классе узла открытого массива дочерних узлов, как показано в следующем коде. Тогда для оперирования дочерними узлами можно использовать методы работы со списками на основе массивов.

Второй подход состоит в том, чтобы сохранять ссылки на дочерние узлы в связных списках. Каждый узел содержит ссылку на первого потомка. Он также содержит ссылку на следующего потомка на том же уровне дерева. Эти связи образуют связный список узлов одного уровня, поэтому я называю этот метод представлением в виде связного списка узлов одного уровня.

Третий подход заключается в том, чтобы определять в классе узла открытую коллекцию, которая будет содержать дочерние узлы.

Представление *нумерацией связей* позволяет компактно представить деревья, графы и сети при помощи массива. Для представления дерева нумерацией связей, в массиве FirstLink записывается индекс для первых ветвей, выходящих из каждого узла. В другой массив, ToNode, заносятся узлы, к которым ведет ветвь.

Сигнальная метка в конце массива FirstLink указывает на точку сразу после последнего элемента массива ToNode. Это позволяет определить, какие ветви выходят из каждого узла.

Используя представление нумерацией связей, можно быстро найти связи, выходящие из определенного узла. С другой стороны, очень сложно изменять структуру данных, представленных в таком виде.

Последовательное обращение ко всем узлам называется обходом дерева. Обход в глубину начинается с прохода вглубь дерева до тех пор, пока алгоритм не достигнет листьев. При возврате из рекурсивного вызова подпрограммы, алгоритм перемещается по дереву в обратном направлении, просматривая пути, которые он пропустил при движении вниз.

Другой метод перебора узлов дерева - это обход в ширину. Этот метод обращается ко всем узлам на заданном уровне дерева, перед тем, как перейти к более глубоким уровням. Алгоритмы, которые проводят полный поиск по дереву, часто используют обход в ширину.

Детали реализации обхода зависят от того, как записано дерево. Для обхода дерева на основе коллекций дочерних узлов, программа должна использовать несколько другой алгоритм, чем для обхода дерева, записанного при помощи нумерации связей.

Особенно просто обходить полные деревья, записанные в массиве. Алгоритм обхода в ширину, который требует дополнительных усилий в других представлениях деревьев, для представлений на основе массива тривиален, так как узлы записаны в таком же порядке.

Язык логического программирования Visual Prolog. В октябре 1981 г. была широко распространена информация о японском проекте создания ЭВМ пятого поколения. Многих специалистов удивило, что в основу методологии разработки программных средств было положено логическое программирование. Целью проекта декларировалось создание систем обработки информации, базирующихся на знаниях. Тогда же появляется множество коммерческих реализаций Пролога практически для всех типов компьютеров. К наиболее известным можно отнести CProlog, Quintus Prolog. Silogic Knowledge Workbench. Prolog-2, Arity Prolog, Prolog-86, Turbo Prolog и др.

Наибольшую популярность в нашей стране получила система программирования Turbo Prolog - коммерческая реализация языка для IBM-совместимых ПК. Его первая версия была разработана датской компанией Prolog Development Center (PDC) в содружестве с фирмой Borland International в 1986 г. Система создавалась с серьезными отступлениями от неофициального стандарта языка, самым существенным из которых было введение строгой типизации данных, но это позволило значительно ускорить трансляцию и выполнение программ. Новый компилятор сразу же был по достоинству оценен программистами-практиками, хотя и вызвал критику в академических кругах.

В 1988 г. вышла значительно более мощная версия Turbo Prolog 2.0, включающая усовершенствованную интегрированную среду разработки программ. быстрый компилятор и средства низкоуровневого программирования. Кроме того, она предоставляла возможность работы с собственными внешними БД, dBase 111 и Reflex, интегрированным пакетом Lotus 1-2-3. графическим пакетом Paint Brush и другими приложениями. Фирма Borland распространяла эту версию до 1990 г., а затем компания PDC приобрела монопольное право на использование исходных текстов компилятора и дальнейшее продвижение системы программирования на рынок под названием PDC Prolog. В июне 1992 г. появилась версия 3.31 -эффективный универсальный инструмент профессиональных программистов, который вскоре стал одним из наиболее широко используемых. PDC Prolog

3.31 работал в среде MS DOS, OS/2, UNIX, XENIX, PharLap DOS Extender, MS Windows. Эта версия была хорошо совместима с традиционными языками программирования, в первую очередь с Си. В ней были расширены возможности создания приложений с интерфейсом GUI (Graphical User Interface), принятым в MS Windows и OS/2.

Хотя версия PDC Prolog 3.31 уже включала средства для написания программ, работающих под управлением графических операционных систем, процесс разработки подобных приложений все еще носил рутинный характер. Для того чтобы сделать более простыми, удобными и быстрыми процессы написания, тестирования и модификации приложений на языке PDC Prolog, специалисты Prolog Development Center создали систему программирования под названием Visual Prolog 4.0, выпущенную 7 января 1996 г.

При разработке приложений в среде Visual Prolog используется подход, получивший название «визуальное программирование», при котором внешний вид и поведение программ определяются с помощью специальных графических средств проектирования без традиционного программирования на алгоритмическом языке. В результате получили систему программирования, отличающуюся исключительной логичностью, простотой и эффективностью.

В Visual Prolog входят различные элементы: прежде всего, интерактивная среда визуальной разработки (VDE - Visual Develop Environment), которая включает текстовый и различные графические редакторы, инструментальные средства генерации кода, конструирующие управляющую логику (Experts), а также являющийся расширением языка интерфейс визуального программирования (VPI - Visual Programming Interface), Пролог-компилятор, набор различных подключаемых файлов и библиотек, редактор связей, файлы, содержащие примеры и помощь.

Visual Prolog поддерживается различными ОС, в том числе MS-DOS PharLap-Extended DOS, всеми версиями Windows, 16- и 32-битовыми целевыми платформами OS/2, а также некоторыми другими системами, требующими графического пользовательского интерфейса.

В зависимости от выбранного интерфейса разработчику обеспечивается доступ к множеству генераторов кода (Code Expert), всевозможным ресурсным редакторам и особым дополнительным VPI-предикатам. определениям и библиотекам. Ресурсные редакторы применяются для создания, компоновки и редактирования окон, диалогов, меню, панелей инструментов, строк помощи, строковых таблиц, ярлыков, курсоров, битовых карт и оперативной помощи. Генераторы кода на основе подобных структур создают необходимый первичный Prolog-код. В результате появляется первичный код («скелет»), готовый для компиляции, редактирования связей и выполнения.

По желанию программиста генераторы кода могут отобразить любую часть первичного текста программы в окне редактора для просмотра и дополнения в соответствии с логикой приложения, а также для преобразования «скелета» в полноценное приложение. Этот процесс реализуется с помощью различных функций: редактирования, выбора, поиска, перемещения и вставки.

Прикладной программный интерфейс высокого уровня облегчает проектирование Пролог-приложений с утонченным видовым пользовательским решением, использующим графические возможности современных ОС и аппаратных средств отображения информации. Ресурсы и инструментальные средства, требующиеся для таких приложений (окна, меню, диалоги, органы управления, перья, кисти, курсоры мыши, графические курсоры, рисунки и т. п.), представляются в виде несложных Пролог-структур.

С помощью VPI можно создать мобильный исходный текст, а затем перекомпилировать его для работы как в 16-битовом режиме под управлением MS DOS или Windows, так и в 32-битовом режиме под управлением Windows NT, OS/2 PM и других ОС.

В декабре 1997 г. фирма PDC выпустила Visual Prolog 5.0, а с января 1999 г. приступила к распространению версии 5.1.

Пролог - это язык, который до сих пор находится в развитии. Область его применения постоянно расширяется, в него вносятся новые дополнительные функциональные возможности, призванные удовлетворить возрастающие потребности пользователей,

Visual Prolog является Prolog-системой со 100% оболочкой, выполненной в идеологии Visual, упрощающей разработку программ для систем Windows 3.x. Windows 95/98/2000, Windows NT. Среда разработки приложений системы Visual Prolog включает удобный текстовый редактор, различные редакторы ресурсов, средства разработки гипертекстовых Help-систем, оптимизирующий компилятор. Visual Prolog автоматизирует труд программиста по построению сложных процедур. С его помощью проектирование пользовательского интерфейса и связанных с ним окон, диалогов, меню, кнопок, строк состояния и т.п. производится в графической среде. С созданными объектами сразу же могут работать различные Кодовые Эксперты, генерирующие программные коды на языке Пролог.

Мощность языка Пролог и визуальной среды разработки программ делает простой и интуитивно понятной разработку экспертных систем, основанных на знаниях, систем поддержки принятия решений, планирующих программ, развитых систем управления базами данных и др., а также обеспечивает повышение скорости разработки приложений.

Интегрированная инструментальная среда GURU. В инструментальной среде построения ЭС GURU, разработанной фирмой Micro Data Base Systems, Inc., методы экспертных систем сочетаются с такими средствами обработки данных, как составление электронных ведомостей, управление базой данных и деловой графикой, и таким образом формируется уникальная среда для поддержки принятия решений и разработки прикладных интеллектуальных систем.

Система GURU легка в употреблении для новичков и в то же время является достаточно эффективной и гибкой системой для профессионалов - разработчиков.

В обычных «интегрированных» программных продуктах или несколько отдельных программ помещены в операционную среду, или несколько, второстепенных компонентов вкладываются в главный компонент (как, например, программа обработки электронных ведомостей или текстовый редактор). Трудности, с которыми сталкиваются при таких стилях «интеграции», хорошо известны. Первая трудность заключается в том, что пользователь вынужден переходить назад и вперед по отдельным программам и передавать данные между ними. Метод вложений заставляет пользователя выполнить всю обработку в пределах главного компонента, и в результате получаются относительно слабые вторичные компоненты.

Метод Интеграции, используемый в системе GURU, совершенно отличается от вышеупомянутых. Он основывается на принципе синергизма. При этом под «синергизмом» здесь понимается следующее. В системе GURU все средства всегда доступны. Многочисленные компоненты можно соединять по желанию в пределах одной операции, а это характеризует систему как гибкую и удобную в использовании. Например:

- в посылке любого правила ЭС можно делать прямые ссылки на поля базы данных, на ячейки электронных ведомостей, на статические переменные, на программные переменные и массивы;

- вывод любого правила ЭС может включать в себя операции управления базой данных, запросы на языке SQL (языке структурированных запросов), операции обработки электронных ведомостей, генерацию статистических данных, дистанционную передачу данных, выполнение процедур, генерацию деловой графики

- поскольку ЭС обосновывает задачу, она может брать консультации у других ЭС, выполнять процедурные модели, просматривать базы данных, составлять электронные ведомости или проводить статистический анализ;

- любую ячейку электронной ведомости можно определить в терминах поиска в базе данных или в терминах всей программы, или в терминах консультации с ЭС.

Взаимодействовать с системой можно любым из четырех различных способов: с помощью меню: на ограниченном естественном языке, в режиме команд или через специально разработанные интерфейсы. Каждый тип интерфейса системы GURU предназначен для удовлетворения потребностей и вкусов различных классов пользователей. Всеми четырьмя интерфейсами можно пользоваться во время одного и того же сеанса взаимодействия с системой GURU.

Как и в большинстве оболочек, в GURU используется продукционная модель представления знаний в виде совокупности «If-then» правил с обратной стратегией вывода в качестве основной имеется возможность моделирования нечетких и неточных рассуждений. Кроме посылок и заключения в правила можно включать команды, которые будут выполняться перед проверкой условия, а также пояснительный текст для генерации объяснений. Правила также включают необязательные параметры цены и приоритета, позволяющие управлять процессом выбора из совокупности, готовых, к выполнению правил очередного. С каждым правилом можно также связать число, определяющее, сколько раз это правило может выполняться в процессе консультации.

Правила, относящиеся к решению некоторой общей задачи, образуют базу знаний, или набор правил. В этот набор кроме правил включаются две специальные процедуры: инициализация и завершение, которые должны выполняться до и после выполнения правил. В набор правил также включаются описания переменных, участвующих в правилах, содержащие спецификации типа, точности и т. п.

По умолчанию в GURU принята стратегия обратного вывода, однако, можно использовать чисто прямой вывод, а также комбинировать его с обратным в рамках одного набора правил. Как стратегиями вывода, так и целевыми переменными можно управлять динамически в процессе консультации.

GURU обеспечивает мощные средства управления обработкой факторов уверенности, отражающих степень неточности и нечеткости выраженных в правилах эвристических знаний. Для предоставления такой нечеткости в GURU с каждым значением переменной может быть связан числовой коэффициент от 0 до 100. Система предоставляет разработчику выбор более чем из 30 различных формул, позволяющих управлять обработкой факторов уверенности во время вывода.

Полезными являются такие дополнительные средства управления логическим выводом, как установка степени «точности» вывода значения для некоторой переменной, изменение принятого по умолчанию порядка просмотра правил.

Эффективность машины логического вывода во многом зависит от того, как она осуществляет поиск в наборе правил, когда ищет правила, которые можно выполнять. В

отличие от традиционного программного обеспечения, использующего принципы искусственного интеллекта, система GURU предоставляет расширенные средства управления настройкой, в частности поддерживает до 50 различных стратегий поиска. Эффективность также зависит от количества и состава правил в наборе правил. Поскольку система GURU предоставляет разнообразные возможности создания наборов правил, то можно значительно сократить количество правил, необходимых для охвата всех знаний и опыта в конкретной проблемной области. Это приводит к ускорению процесса получения логических выводов, а также к упрощению управления этими правилами.

Интегрированная система GURU пытается превратить потенциальные преимущества ЭС в реальность, облегчить пользователю процесс создания ЭС, сделать его прямым, эффективным и естественным.

Интегрированная инструментальная среда G2 для создания интеллектуальных систем реального времени. В 1986 г. фирма Gensym вышла на рынок с инструментальным средством G2, версия 1.0. В настоящее время функционирует уже версия 5.2.

Основное предназначение программных продуктов фирмы Gensym (США) - помочь предприятиям сохранять и использовать знания и опыт наиболее квалифицированных сотрудников в интеллектуальных системах реального времени, повышающих качество продукции, надежность и безопасность производства и снижающих производственные издержки.

Классы задач, для которых предназначена G2 и подобные ей системы:

- мониторинг в реальном масштабе времени;
- системы управления верхнего уровня;
- системы обнаружения неисправностей;
- диагностика;
- составление расписаний;
- планирование;
- оптимизация;
- системы - советчики оператора;
- системы проектирования.

Инструментальный комплекс G2 является эволюционным шагом в развитии традиционных экспертных систем от статических предметных областей к динамическим.

Основные принципы, которые заложены в G2:

- проблемно/предметная ориентация;
- следование стандартам;
- независимость от вычислительной платформы;

- универсальные возможности, не зависимые от решаемой задачи;

- обеспечение технологической основы для прикладных систем;

- комфортная среда разработки;

- распределенная архитектура клиент-сервер;

- высокая производительность.

Основным достоинством оболочки экспертных систем G2 является возможность применять ее как интегрирующий компонент, позволяющий за счет открытости интерфейсов и поддержки широкого спектра вычислительных платформ объединить уже существующие, средства автоматизации в единую комплексную систему управления, охватывающую все аспекты производственной деятельности - от формирования портфеля заказов до управления технологическим процессом и отгрузки готовой продукции.

На основе базового средства G2 фирма Gensym разработала комплекс проблемно/предметно-ориентированных расширений для быстрой реализации сложных динамических систем на основе специализированных графических языков, включающих параметризуемые операторные блоки для представления элементов технологического процесса и типовых задач обработки информации. Набор инструментальных сред**,** сгруппированный по проблемной ориентации, охватывает все стадии производственного процесса и выглядит следующим образом:

- интеллектуальное управление производством - G2 Diagnostic Assistant (GDA), Statistical Process Control (SPC).

- оперативное планирование - G2 Scheduling Toolkit (GST), Dynamic Scheduling Packadge (DSP);

- разработка и моделирование производственных процессов - G2, ReThink.

- управление операциями и корпоративными сетями - Fault Expert.

G2 динамическая система в полном смысле этого слова. Это объектно-ориентированная интегрированная среда для разработки и сопровождения приложений реального времени, использующих базы знаний. G2 функционирует на большинстве существующих платформ: Solaris 1 and 2, Unix, OpenVMS, Windows NT / 2000 Professional / XP. База знаний G2 сохраняется в обычном ASCII-файле, который однозначно интерпретируется на любой из поддерживаемых платформ. Перенос приложения не требует его перекомпиляции и заключается в простом переписывание файлов. Функциональные возможности и внешний вид приложения не претерпевают при этом никаких изменений.

G2 - среда для разработки и развертывания интеллектуальных динамических систем управления. Прикладные программы, написанные в среде G2, могут существенно повысить эффективность выполнения операций, благодаря следующим факторам

- непрерывному контролю над потенциальными проблемами прежде, чем они проявят неблагоприятное воздействие;

- принятие комплексных оперативных решений на основе информации, полученной посредством рассуждений и анализа данных, содержащихся в интеллектуальной модели процесса;

- диагностирование основных случаев возникновения проблем, критичных ко времени выполнения и выработки последовательности правильных действий;

- поддержание оптимальных рабочих условий;

- координирование действий и информации в выполнении сложных оперативных процессах.

Использование мощности объектно-ориентированного программирования. Объекты в G2 - это интуитивный способ представления материальных и абстрактных сущностей в прикладных программах. Мощность объектно-ориентированного подхода к разработке программ, позволяет быстро и легко:

- встроить модули и объекты из других прикладных программ;

- графически определить объекты, их свойства и действия;

- создать новые образцы объектов, имитируя существующие объекты.

Объекты или класс объектов определенные один раз могут использоваться многократно. Любой объект или группа объектов могут иметь несколько экземпляров. Каждый экземпляр наследует все свойства и поведение первоначального объекта (ов). Объекты, правила, и процедуры можно группировать в библиотеки, которые будут общими для всех прикладных программ. Для моделирования широкого разнообразия действий типа производственных процессов, сетевых топологий, информационных маршрутизации, или логических потоков объекты можно объединять графически на экране дисплея.

Представление знаний, правила, процедуры и модели. В G2 можно эффективно создавать и применять общие знания, создавая универсальные правила, процедуры, формулы и зависимости, которые являются применимыми для полных классов объектов. В результате сокращается время на разработку и увеличивается эффективность приложений.

Для представления знаний используется структурный естественный язык, что позволяет облегчить чтение, редактирование и поддержку баз знаний. Это облегчает использование и редактирование приложений пользователем непрограммистом. Для создания и редактирования баз знаний используется Редактор Баз Знаний.

Для представления знаний эксперта о проблемной области используются правила. Правила могут быть как общими, то есть относящимся ко всему классу, так и специфическим, относящимся к конкретным экземплярам класса. Правила возбуждаются автоматически в следующих случаях:

- появились новые данные (вывод от фактов к цели);

- требуется найти данные (вывод от цели к фактам) для автоматического вызова других правил, процедур, или формул;

- требуется определить значения переменных;

- каждые *n* секунд для оценивания правила в указанном интервале времени.

Подобно правилам, процедуры выполняются в реальном времени. Процедуры, правила, и модели выполняются одновременно согласно их приоритетам. Процедуры могут использоваться для эффективного представления поведения объектов. Для процедур можно определить состояния ожидания и выполнять обработку информации параллельно. В результате это позволяет формировать мощные прикладные системы реального времени проще и быстрее, чем с помощью традиционных инструментальных средств программирования.

Работа в реальном времени. Работа в реальном времени, операционные решения и реакции зачастую должны быть выполнены мгновенно. Прикладные программы в G2 могут одновременно выполнять рассуждения относительно многократно выполняемых действий в реальном масштабе времени, перерабатывая тысячи правил, выполняя процедуры и модели согласно их приоритетам. Для хранения хронологий данных и событий и для рассуждения относительно поведения через какое-то время используются переменные типа время.

G2 графика может моделировать знание, представляя объекты, связи и зависимости между объектами. Она может рассуждать в терминах связи, следуя сети связанных объектов для определения причин и результатов. Графическая связность объектов позволяет расширить прикладную программу используя графическое объединение аналогов. Графика включает встроенные диаграммы **(**графики**),** таблицы и рисунки и т.д.

G2 также работает с утилитами графического интерфейса Windows. Эти утилиты используют все преимущества объектно-ориентированных возможностей G2.

Динамическое моделирование и моделирование для анализа "ЧТО-ЕСЛИ". G2 позволяют динамически моделировать системы и процессы, используя объекты, правила, процедуры и формулы. Во время разработки модели используются объекты реального мира, что позволяет непрерывно проверять прикладные программы в течение их разработки. Модели могут использоваться на этапе эксплуатации как часть прикладной

программы G2 для сравнения фактических и модельных знаний.

Модели можно также использовать для проведения анализа и ответа на вопросы "ЧТО- ЕСЛИ", определения, например, лучших рабочих условий или лучших проектов. Модели можно также использовать для предсказания важных параметров действий в реальном времени, например, качество изделия или затрат.

G2 объединяет в себе как универсальные технологии построения современных информационных систем:

• стандарты открытых систем,

• архитектуру клиент/сервер,

• объектно-ориентированное программирование,

• использование ОС, обеспечивающих параллельное выполнение в реальном времени многих независимых процессов,

• специализированные методы (рассуждения, основанные на правилах, рассуждения, основанные на динамических моделях, или имитационное моделирование, процедурные рассуждения, активная объектная графика, структурированный естественный язык для представления базы знаний),

а также интегрирует технологии систем, основанных на знаниях с технологией традиционного программирования (с пакетами программ, с СУБД, с контроллерами и концентраторами данных и т.д.).

Все это позволяет с помощью данной оболочки создавать практически любые большие приложения значительно быстрее, чем с использованием традиционных методов программирования, и снизить трудозатраты на сопровождение готовых приложений и их перенос на другие платформы.

Язык CLIPS. Название языка CLIPS - аббревиатура от *C Language Integrated Production System.*

Язык был разработан в Центре космических исследований NASA (NASA's Johnson Space Center) в середине **1980-х** годов и во многом сходен с языками, созданными на базе LISP, в частности OPS5 и ART. Использование С в качестве языка реализации объясняется тем, что компилятор LISP не поддерживается частью распространенных платформ, а также сложностью интеграции LISP-кода в приложения, которые используют отличный от LISP язык программирования. Хотя в то время на рынке уже появились программные средства для задач искусственного интеллекта, разработанные на языке С, специалисты из NASA решили создать такой продукт самостоятельно. Разработанная ими система в настоящее

время доступна во всем мире, и нужно сказать, что по своим возможностям она не уступает множеству гораздо более дорогих коммерческих продуктов.

Первая версия представляла собой, по сути, интерпретатор порождающих правил. Процедурный язык и объектно-ориентированное расширение CLIPS Object-Oriented Language (COOL) были включены в этот программный продукт только в 1990-х годах. Существующая в настоящее время версия может эксплуатироваться на платформах UNIX, DOS, Windows и Macintosh. Она является хорошо документированным *общедоступным* программным продуктом и доступна по сети FTP с множества университетских сайтов. Исходный код программного пакета CLIPS распространяется совершенно свободно и его можно установить на любой платформе, поддерживающей стандартный компилятор языка С. Однако я бы рекомендовал пользоваться официальной версией для определенной платформы, поскольку такие версии оснащены пользовательским интерфейсом, включающим меню команд и встроенный редактор.

Правила и функции в CLIPS. CLIPS включает:

• язык представления порождающих правил;

• язык описания процедур.

Основными компонентами *языка описания правил* являются:

- *база фактов (fact base)*;

- *база правил (rule base).*

На них возлагаются следующие функции:

- база фактов *представляет* исходное состояние проблемы*;*
- база правил *содержит операторы, которые преобразуют состояние проблемы, приводя его к решению.*

Машина логического вывода CLIPS сопоставляет эти факты и правила и выясняет, какие из правил можно активизировать. Это выполняется циклически, причем каждый цикл состоит из трех шагов:

(1) *сопоставление* фактов и правил;

(2) *выбор* правила, подлежащего активизации;

(3) *выполнение* действий, предписанных правилом.

Такой трехшаговый циклический процесс иногда называют *"циклом распознавание-действие".*