4. СРЕДСТВА РАЗРАБОТКИ СИИ

Средства разработки ЭС условно делятся на четыре категории:

1) алгоритмические языки программирования;

2) языки программирования для ИИ;

3) интегрированные инструментальные среды (средства);

4) "оболочки" ЭС (shell-системы, "пустые" ЭС).

Такая классификация считается условной по двум причинам, во-первых, среди авторов не достигнуто согласие в терминологии, например, вместо категории "языки программирования для ИИ" используется "языки специального назначения", которая частично перекрывает вторую и третью категории данной классификации; во-вторых, существуют программные средства, которые трудно однозначно отнести к одной из перечисленных категорий.

4.1. Алгоритмические языки программирования

Алгоритмические языки программирования являются универсальным средством построения ЭС, используя их можно создать практически любую ЭС, но трудоемкость выполнения такой работы, а соответственно и сроки, будут велики. Разработка начинается прямо с языка программирования. Разработчик должен разработать язык представления знаний и средства его обработки, машину вывода, механизм контроля, среду разработки ЭС (сервисные программы), подсистему объяснения.

К этой категории средств разработки ЭС относятся языки C, Pascal, BASIC, FORTH.

Достоинства:

1) высокая гибкость, разрабатывается ЭС практически любой архитектуры и с любой моделью представления знаний;

2) высокая эффективность программной реализации;

3) высокий уровень адаптируемости к проблемной области, позволяет наилучшим образом учесть всю ее специфику;

4) относительно небольшая стоимость инструментария (компилятора языка и среды разработки программ);

5) хорошие возможности по подключению других программных систем и включения данной ЭС в другие комплексы в качестве подсистемы;

6) возможность накопления опыта разработки ЭС.

Недостатки:

1) разработка требует опытных программистов, имеющих опыт разработки ЭС;

2) очень высокая трудоемкость и большие сроки разработки.

4.2. Языки программирования для ИИ

К этой категории средств относят языки Lisp, Prolog, SmallTalk. Языки программирования для ИИ, ориентированы на обработку символьной информации. Они имеют язык представления знаний и включают элементы машины вывода.

Достоинства:

1) достаточная гибкость;

2) хорошая эффективность программной реализации (но, обычно, хуже, чем при использовании алгоритмических языков);

3) хороший уровень адаптируемости к проблемной области;

4) также небольшая стоимость инструментария;

5) обычно имеются возможности подключения других программных систем и включения данной ЭС в другие комплексы в качестве подсистемы.

Недостатки:

1) требует программистов с хорошей подготовкой;

2) довольно большие трудоемкость и сроки разработки;

3) низкая эффективность на ЭВМ стандартной архитектуры.

4.2.1. Краткая характеристика языка Лисп (LISP)

LISP - **LIS**t **P**rocessing (обработка списков). Язык Лисп был создан в 60-х годах американским ученым Дж. Маккарти и его учениками (Массачусетский технологический институт). На сегодняшний день

существует около 20 диалектов Лисп. Наиболее известными являются INTERLISP, FRANZLISP, QLISP, COMMONLISP.

На языке Лисп написаны:

экспертные системы (MYCIN, INTERNIST, KEE и др.);

системы естественно-языкового общения (MARGIE, SHRDLU и др.);

интеллектуальные операционные системы (FLEX).

Язык Лисп имеет очень простой синтаксис, поскольку возможны только две его конструкции: атом и список. Атом - это элементарная конструкция языка Лисп, характеризуемая своим именем и значением. Атомы состоят из букв и/или цифр. Атом языка Лисп – это то, что в алгоритмических языках является идентификатором или числовой константой. В некоторых Лисп-системах с атомом связывается также определенный список свойств. Примеры атомов: A, B, ВЫПУСК, АИ - 93 и т.д. Список - это конструкция Лисп, состоящая из множества атомов и подсписков. Это некоторый набор элементов. Элементы списка называются членами списка. Список заключается в круглые скобки, перед списком ставится апостроф (символ одинарной кавычки """), для того, чтобы можно различать список и функцию. Пример списка - перечень предметов, лежащих на рабочем столе:

'(КАРАНДАШ РУЧКА БУМАГА СКРЕПКИ)

Разделителями членов списка могут быть пробелы или запятые.

Пример списка, членами которого являются тоже списки:

'((PASCAL C BASIC) (LISP PROLOG) (ART KEE) (EMYCIN EXSYS GURU))

Существенной особенностью языка Лисп является то, что здесь "данные" и "программы" (операторы) внешне ничем не отличаются друг от друга. Это дает возможность писать на Лиспе "программы", манипулирующие не только "данными", но и "программами". Именно данное свойство позволяет Лиспу стать средством программирования систем ИИ. Понятие "данные" и "программа" в Лиспе не используются, их заменяют такие понятия, как выражение и функция.

Лисп - функциональный язык. Все процедуры обработки информации оформляются в виде функций. Благодаря стандартному набору системных функций, Лисп может быть "расширен" за счет пользовательских функций. Функции в Лисп записываются в префиксной форме, т.е. сначала записываются операция, а затем ее операнды и вся конструкция заключается в круглые скобки:

(PLUS S1 S2...SN) - вычисляет сумму своих аргументов;

(PLUS 4 2) - результатом вычисления этой функции будет число 6;

(TIMES S1 S2) - вычисляет произведение своих аргументов;

(MAX S1 S2...SN) - возвращает наибольшее число из списка;

(MAX 5 9 2) - система вернет 9.

Встроенные функции обработки списков:

(CAR List) - возвращает первый элемент списка List;

(CAR '(A B C)) возвращает A;

(CDR List) - возвращает список List без первого элемента;

(CDR '(A B C)) - возвращает '(B C);

(CDR '((A B) (C D))) - возвращает '(C D);

(CONS <s-выражение> List) - s-выражение добавляется к списку в качестве первого элемента; s-выражение - это символьное выражение, оно состоит из атомов и списков;

(CONS 'K '(J K L)) - возвращает '(K J K L);

(CONS '(A B) '(J K L)) - возвращает '((A B) (J K L).

(APPEND List1 List2) - объединяет два списка в один;

(APPEND '(A B) '(J K L)) - возвращает '(A B J K L).

Предикаты или функции проверки условий:

(эти функции выполняют проверку условий и возвращают значение "ложь" (NIL) или "истина" (T), поэтому они называются предикатами)

(EQUAL S1 S2) - возвращает T, если значения аргументов равны, иначе NIL;

(EQUAL 3 2) - дает в результате NIL;

(EQUAL '(1 2 A) '(1 2 A)) - даст в результате T.

(GREATERP S1 S2) - возвращает T, если аргумент S1 больше S2, иначе NIL.

(ZEROP S1) - возвращает T, если аргумент 0, иначе NIL.

(NULL List) - предикат возвращает значение T, если список List пуст и NIL, если List не пустой список.

Значение переменной может быть присвоено с помощью следующего предложения:

(SETQ X Y).

В результате выполнения этого предложения переменной X будет присвоено значение Y. Например, (SETQ X 25) - переменной X будет присвоено значение 25; (SETQ X '(A B C)) - переменной X будет присвоено значение, представляющее собой список '(A B C).

Правила продукций в языке Лисп могут быть представлены с помощью условной функции. Условная функция (COND) позволяет выполнить некоторые действия. Например:

```
(COND ((GREATERP 4 Z) (SETQ P 3))).
```

Предложение можно прочитать так: "Если 4 больше Z, то присвоить P значение 3". В этом примере предложение COND состояло из двух частей: "Если" и "то", т.е.

```
(COND ((часть - Если)  (часть - ТО))).
```

В общем случае функция COND может включать несколько частей "Если - то".

Для создания новых функций в языке Лисп применяется функция DEFUN. Например, предложение

```
(DEFUN ADD-DIF (X Y Z)
  (PLUS Z (DIFFERENCE X Y)))
```

определяет новую функцию с именем ADD-DIF. Элементы, идущие в скобках за именем функции, - ее аргументы. Таким образом, аргументами функции ADD-DIF будут переменные X, Y, Z. Аргументы используются при вычислении функции. Сами вычисления задаются после аргументов. В этом примере вычисления заданы выражением:

```
  (PLUS Z (DIFFERENCE X Y)),
```

Здесь прежде всего из X вычисляется Y, а затем полученное значение складывается со значением Z. Если вновь определенную функцию вызвать с аргументами 4, 5, 6: (ADD-DIF 5 4 6), то результатом ее работы будет число 7.

Лисп - это рекурсивный язык, т.е. обеспечивает возможность определения функций с помощью самих себя. Например, известно, что факториал можно определить двояко:

1) N! = 1 * 2 *...* N;
2) N! = N * (N - 1)!;

второй вариант на языке Лисп выглядит следующим образом:

```
(DEFUN FACT (N)
 (COND ((ZEROP N)  1)    /* если N = 0, то возвращает 1 */
   (T (TIMES N (FACT (DIFFERENCE N 1))))
)).
```

Если вызвать функцию (FACT 4) и осуществить трассировку, то получим

```
(FACT 4)
  4 * (FACT 3)
      3 * (FACT 2)
          2 * (FACT 1)
              1 * (FACT 0)
                  1
```

и окончание рекурсии.

Декларативные знания на языке Лисп можно представлять следующим образом. Например, некоторые знания о треугольнике и квадрате могут быть представлены:

```
  (треугольник
    (стороны A B C )
    (углы alfa betta gamma)
    (параметры площадь периметр)
    (тип обычный прямоугольный равнобедренный правильный));

  (квадрат
    (есть четырехугольник)
    (особенности
      (все-стороны равны)
      (все-углы прямые))).
```

Одним из основных **отличий языка Лисп** от традиционных языков программирования является запись в Лиспе в виде списков не только данных, но и программ (или функций). Например, список (PLUS 2 3) можно интерпретировать как действие, дающее в результате число 5, либо как список, состоящий из трех элементов.

4.2.2. Краткая характеристика языка Пролог (Prolog)

PROLOG - PROgramming in **LOG**ic (логическое программирование). Язык Пролог был разработан в 70-е годы. В Марсельском университете была разработана первая версия языка Пролог. В Великобритании работы по созданию языка Пролог проводились проф. Р.Ковальским и коллективом исследователей Лондонского имперского колледжа. В Эдинбургском университете была выполнена реализация Пролога на ЭВМ DEC - 10. В настоящее время Пролог - это широко известный язык с хорошей документацией, который был взят японцами как базовый язык для проекта ЭВМ пятого поколения. Крупные денежные средства были вложены в разработку Пролога в рамках программы "ESPRIT" Европейского экономического сообщества (ESPRIT = European Program for Research in Information Technology) и проекта "Alvey" в Великобритании ("Alvey" - 1982г. - отчет-программа группы, возглавляемой Джоном Элви из фирмы British Telecom - реакция Великобритании на Японский вызов - проект ЭВМ 5-го поколения), а также министерством обороны США и компанией IBM.

Пролог относится в большей степени к реляционным или **декларативным (описательным)**, а не к функциональным языкам. Функция - это вид процедуры, которая выбирает переданные ей аргументы, обрабатывает их и возвращает результат. Программа на реляционном языке состоит из операторов, описывающих отношения между представленными объектами. Пролог является языком логического программирования. **Логическая программа** - это множество аксиом и правил, задающих отношение между объектами. Вычислением логической программы является вывод следствий из программы. Математической основой Пролога являются исчисление предикатов, преимущественно первого порядка, метод резолюции Робинсона, теория рекурсивных функций.

Простейшим видом утверждения является факт. Факты используются для констатации того, что выполнено некоторое отношение между объектами. Факты представляются в форме предикатов:

<имя_предиката> (<аргумент_1>, ..., <аргумент_ К>).

Аргумент предиката называется термом. В отличие от классического исчисления предикатов первого порядка сам предикат в Прологе рассматривается как терм и может выступать в роли аргумента другого предиката. Факт "плюс (2, 3, 5)" означает, что имеет место отношение "2 плюс 3 равно 5". Отношение "плюс" может быть задано с помощью множества фактов, определяющих таблицу сложения. Вот начальный фрагмент таблицы (левый верхний угол):

плюс (0,0,0). плюс (0,1,1). плюс (0,2,2). плюс (0,3,3). ...
плюс (1,0,1). плюс (1,1,2). плюс (1,2,3). ...
плюс (2,0,2). плюс (2,1,3). плюс (2,2,4). ...
. . .
. . .

Конечное множество фактов образует программу. Это простейший вид логической программы. Множество фактов, с другой стороны, составляет описание ситуации. Следовательно, описание ситуации может рассматриваться как логическая программа.

Вопрос - это средство извлечения информации из логической программы. С помощью вопроса выясняется, выполнено ли некоторое отношение между объектами. Конец факта обозначается точкой, а конец вопроса - вопросительным знаком. Поиск ответа на вопрос состоит в том, чтобы определить, является ли вопрос логическим следствием программы. Логические следствия выводятся путем применения правил.

Простейшее правило вывода - совпадение: вопрос является логическим следствием тождественного с ним факта. Процедура поиска ответа на простой вопрос выполняется непосредственно. В программе ищется факт, предполагаемый в вопросе. Если факт, тождественный вопросу, найден, то ответ - "да" ("истина"). Ответ "нет" ("ложь") дается в том случае, когда факт, тождественный вопросу, не найден, поскольку данный факт не является логическим следствием программы. Например, плюс (1, 1, 2)? дает "да", а плюс (0, 2, 3)? - "нет".

В Прологе используются переменные. Использование переменных в логических программах отличается от использования переменных в традиционных языках программирования. В логических программах переменная обозначает неопределенный, но единственный объект, а не некоторую область памяти. Рассмотрим логическую программу,
содержащую следующие факты:

отец (Иван, Владимир). мужчина (Иван).
отец (Иван, Ирина). женщина (Мария).
отец (Иван, Сергей). мужчина (Владимир).
мать (Мария, Владимир). женщина (Ирина).
мать (Мария, Ирина). мужчина (Сергей).
мать (Мария, Сергей). женщина (Вера).

отец (Сергей, Надежда). Женщина (Надежда).
мать (Вера, Надежда).

Здесь отношение "мать" и "отец"имеют следующий формат:
отец (кто, чей); мать (кто, чья).

Предположим, мы хотим выяснить, чьим отцом является Сергей. Для этого можно задавать вопросы: отец (Сергей Иван)?, отец (Сергей, Мария)? и т.п. до тех пор пока не будет получен ответ "да" или список имен будет исчерпан. С помощью переменной реализуется более удобный способ поиска ответа. Вопрос формулируется в виде отец (Сергей, X)? Ответом на вопрос является X = Надежда. С помощью вопроса, содержащего переменную, выясняется, имеется ли такое значение переменной, при котором вопрос будет логическим следствием программы.

Следующий тип вопросов - это экзистенциальные вопросы (existence - существование), т.е. вопросы, в которых переменные связаны квантором существования. Например, вопрос отец (Сергей, X)? следует читать: "Существует ли такое X, что Сергей является отцом X?". Как видно, это вопросы, аргументами которых частично или полностью являются переменные.

Второе правило вывода - обобщение: вопрос является логическим следствием программы, т.е. получает ответ "да" и значение переменной, стоящей в списке аргументов, если в программе найдется факт, у которого совпадают с вопросом имя отношения и аргументы-константы в соответствующих позициях. Процедура поиска ответа на экзистенциальный вопрос при использовании программы, состоящей из фактов, сводится к поиску такого факта. Экзистенциальный вопрос в общем случае может иметь несколько решений. Из программы ясно, что Иван - отец троих детей. Следовательно, вопрос отец (Иван, X)? имеет решение { X = Владимир}, { X = Ирина}, { X = Сергей }. Другой пример плюс (X,Y,4)? Решение: { X = 0, Y = 4 }, { X = 1, Y = 3 }, { X = 2, Y = 2 }, { X = 3, Y = 1 }, {X = 4, Y = 0 }. В вопросе плюс (X, X, 4)? Требуется, чтобы два числа, дающие в сумме 4, совпадали. Имеется единственный ответ - { X = 2 }.

Переменные также полезны и в фактах. Здесь переменные позволяют выразить совокупность многих фактов. Факт "умножить (0, X, 0)." объединяет все факты, утверждающие, что 0 умноженный на любое число, дает 0, т.е. переменные в фактах неявно связаны квадратом общности. Такие факты называются универсальными.

Третье правило вывода - конкретизация: из утверждения с квантором общности выводится факт при любой подстановке вместо переменной константы.

Правила продукций языка Пролога позволяют определить новые отношения в терминах существующих отношений (т.е. на основании существующих). Правила - это утверждения вида

$$A \longleftarrow B_1, B_2, ..., B_n,$$

где A, B_1, B_2, ..., B_n - предикаты. Смысл его таков: "A истинно, если истинно B_1 и истинно B_2 и ... и истинно B_n ". Например, правило, выражающее свойство "быть сыном":

сын (X,Y) <── отец (Y,X), мужчина (X). /* сын (кто, чей).*/

свойство "быть дочерью":

дочь (X,Y) <── отец (Y,X), женщина (X).

Правило для отношения "быть дедушкой":

дедушка (X,Z) <── отец (X,Y), отец (Y,Z). /*дедушка (кто, чей)*/

Логическая программа - это конечное множество фактов и/или правил.

4.3. Оболочки ЭС

Разработка первых экспертных систем требовала огромных усилий, что выливалось в продолжительные сроки разработки и достаточно высокую ее стоимость. Поэтому очень быстро разработчикам пришла в голову разумная идея удалить содержимое базы знаний и продавать полученный программный продукт или использовать его самим для быстрого построения новых систем. Поскольку база знаний такого продукта была пуста, то его назвали «пустой» экспертной системой. А раз все компоненты программы были в рабочем состоянии и их можно использовать, то также этот продукт стали называть и «оболочкой» экспертной системы.

Основой оболочки выступает архитектура - "готовый" механизм вывода и "пустая" база знаний. Приобретение знаний поддерживают сервисные программы - редактор базы знаний, средства отладки, трассировки, предназначенные для инженера знаний (инженер знаний = разработчик экспертной системы). Основная трудность в их использовании состоит в настройке на конкретную предметную область. Оболочка диктует разработчику язык описания фактов, формализм представления знаний и стратегию вывода. Естественно, что не все задачи укладываются в эту формальную рамку.

Достоинства:

1) очень большая скорость разработки;
2) обычно не требует программирования;

3) малое время изучения оболочки и тренировки.
Недостатки:
1) часто очень дорогие;
2) эффективна только для очень узкого круга задач;
3) слаборазвиваемая.
Примеры "оболочек": Emycin, Expert, Rule Master и т.д.

4.4. Интегрированные инструментальные среды

Основной недостаток оболочек был выявлен достаточно быстро, и он состоит в том, что весьма трудно подобрать подходящую оболочку для конкретной проблемной области (проблемная область = предметная область + совокупность задач, которые необходимо решать в данной предметной области). К тому же оболочки достаточно дорогие. Поэтому в одном программном продукте объединили (интегрировали) несколько вариантов представления знаний и механизмов вывода. Такой продукт также содержал все остальные компоненты экспертной системы. Его назвали интегрированной инструментальной средой.

Интегрированная инструментальная среда позволяет генерировать "оболочки" ЭС в некотором диапазоне, т.е. она дает возможность осуществлять выбор способа представления знаний, соответствующего механизма вывода, механизма контроля. Также она обычно, включает язык специального назначения (часто Лисп), для повышения гибкости системы и специальную поддержку отладки. Работа по разработке ЭС начинается с выбора наиболее подходящей для решаемых задач комбинации характеристик (способа представления знаний и механизма вывода).
Достоинства:
1) большая скорость разработки;
2) обычно не требует программирования;
3) существенно более гибкая, чем "оболочка";
4) есть возможность развития.
Недостатки:
1) достаточно дорогие;
2) готовые системы по показателям время работы и занимаемая память, обычно, неэффективны;
3) иногда требуются опытные программисты, знакомые с методами и моделями ИИ. Примеры: AGE, HEARSAY III, ART, KEE и др.

4.5. Выбор средства разработки ЭС

Прежде всего необходимо определить категорию средств разработки. Выбор производится на основании анализа условий разработки и сопоставления достоинств и недостатков каждой из категорий. Работая на самом низком уровне, уровне языков программирования алгоритмического типа, можно создать практически любую ЭС. В то же время сроки выполнения такой работы будут велики. Работая же с "оболочками", т.е. системами, имеющими пустую базу знаний, разработчик ограничен строго определенной структурой базы знаний, машины логического вывода, полностью закрытой структурой подсистемы объяснения и готовым редактором базы знаний. Сроки разработки системы с использованием "оболочки" невелики, однако эффективность работы созданной системы зачастую мала, поскольку, несмотря на обилие различных "оболочек", очень сложно втиснуть исследуемую предметную область в систему ограничений, накладываемых любой из "оболочек".

Большую свободу допускают инструментальные средства проектирования ЭС. Они, как правило, позволяют изменить в определенных пределах структуру БЗ системы, настраивать ее на выбранную структуру редактора БЗ, комбинировать различные подходы к реализации машины логического вывода и разрабатывать отдельные части подсистемы объяснения, используя БЗ. Сроки создания ЭС с помощью инструментальных средств, больше чем с использованием "оболочек", поскольку требуется привлечение к работе, помимо инженера по знаниям, еще и эксперта, и программиста.

Языки программирования систем ИИ отличаются от алгоритмических языков наличием ярко выраженной ориентацией на логическую обработку знаний. В то же время они существенно экономят время разработки при условии правильного их выбора, наиболее полно соответствующего особенностям данной предметной области.

Опыт разработки ЭС обобщается в следующем правиле: "начинать разработку необходимо с использования "оболочки" или интегрированной инструментальной среды, чтобы быстро создать первый прототип ЭС для исследования и последующего уточнения концепции. После доведения прототипа ЭС до

работоспособного и удовлетворяющего поставленным требованиям осуществляется перевод его на язык программирования с целью повышения эффективности работы ЭС".

Имеется шесть основных вопросов, которые следует задать при выборе инструментального средства:

1. Удовлетворяет ли данное средство бригаду разработчиков по предоставляемым им возможностям?
2. Достаточна ли обеспечиваемая инструментальным средством поддержка для разработки системы в отведенные для этого сроки?
3. Надежно ли выбираемое средство?
4. Поддерживается и сопровождается ли инструментальное средство?
5. Обладает ли средство характеристиками, отвечающими особенностям поставленной задачи?
6. Обладает ли средство характеристиками, отвечающими особенностям ожидаемого применения системы?

Инструментальное средство, которое было наилучшим для разработки ЭС, может не быть наилучшим при создании окончательной версии системы. На стадии разработки ЭС должны быть в распоряжении богатые сервисные возможности (отладочные средства, редакторы и т.д.); инструментальное средство должно также давать достаточную вычислительную мощность, чтобы сократить сроки разработки; оно должно быть также достаточно гибким, чтобы можно было экспериментировать с различными типами и формами представления знаний, структурами управления. С другой стороны, окончательный вариант ЭС должен представлять хорошие возможности пользователю (общение с системой на естественном языке, графику), а также должен работать быстро и эффективно.

Контрольные вопросы и задания

1. Дайте сравнительную характеристику алгоритмическим языкам программирования и оболочкам ЭС как средствам разработки.
2. Назовите отличительные особенности языков программирования для ИИ.
3. Охарактеризуйте интегрированную инструментальную среду как средство разработки ЭС?
4. Каким образом описываются знания на языке Лисп?
5. Каким образом описываются знания на языке Пролог?
6. Что собой представляет логическая программа?
7. Дайте характеристику ситуациям, в которых целесообразно применять каждую из категорий программных средств разработки ЭС.