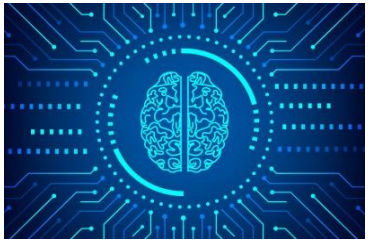
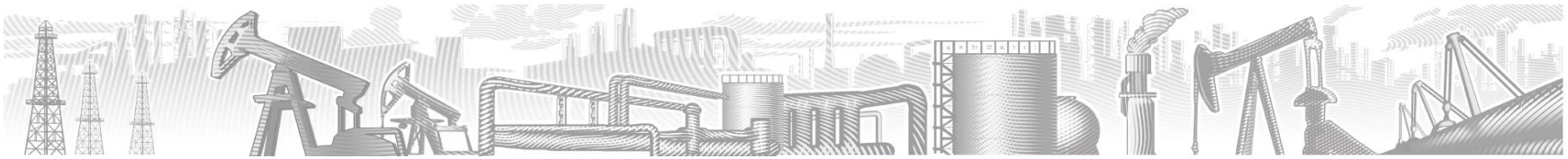
УФИМСКИЙ ГОСУДАРСТВЕННЫЙ
НЕФТЯНОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИНСТИТУТ НЕФТИ И ГАЗА (филиал в г. Октябрьском)



Кафедра «Информационные технологии, математика и естественные науки»

Курс ДОП
ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И НЕЙРОННЫЕ СЕТИ

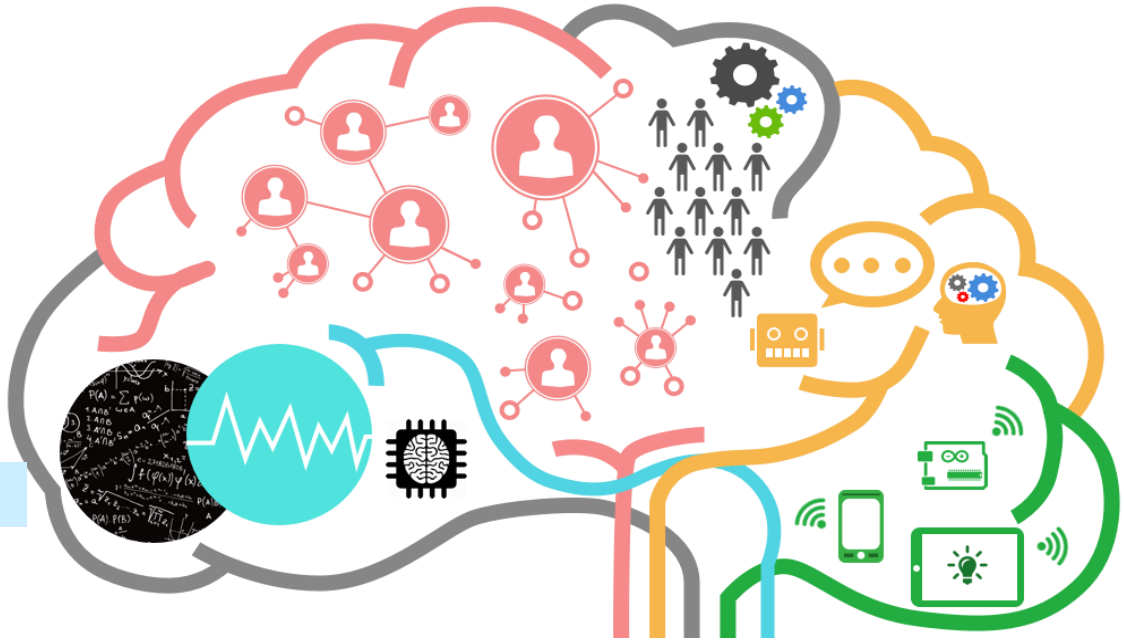
Разработал д.т.н. Тынчеров Камиль Талятович



РАЗДЕЛ 2. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ЭКСПЕРТНЫХ СИСТЕМАХ

Лекция 6. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ (продолжение)

Тема предыдущей лекции: Модели представления знаний (Логическая и ; продукционная модели представления знаний; модули, управляемые образцами; семантические модели).



Учебные вопросы лекции

5. Фреймы.
6. Объектно-ориентированный подход.
7. Практика использования моделей представления знаний в экспертных системах.

Представление знаний в экспертных системах : учебное пособие / сост. В. А. Морозова, В. И. Паутов. — Екатеринбург : Изд-во Урал. ун-та, 2017. — 120 с.
ISBN 978-5-7996-2037-0

Введение

Из материала предыдущей лекции мы узнали, что модели представления знаний – это одно из важнейших направлений исследований в области искусственного интеллекта. Без знаний искусственный интеллект не может существовать.

Модель представления знаний – это способ записи знаний, предназначенный для отображения текущего состояния объектов некоторой предметной области и отношений между ними, а также изменение объектов и отношений.

Модель представления знаний может быть универсальной, то есть применимой для большинства предметных областей, или специализированной, то есть разработанной для конкретной предметной области.

К основным универсальным моделям представления знаний относятся:
- логические модели;
- сетевые модели;
- продукционные модели;
- фреймовые модели.

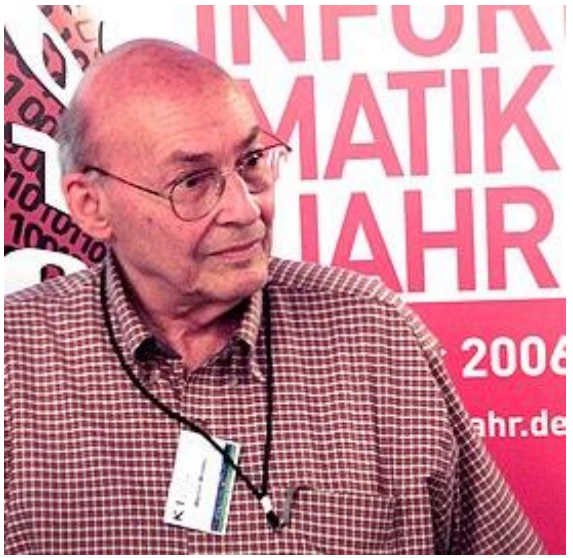
Знания хранятся в базе знаний в соответствии с моделью представления знаний.

5. Фреймы.

Определение фрейма

Стремление разработать представление, соединяющее в себе достоинства различных моделей, привело к возникновению так называемого фрейм-представления [36] (***фрейм*** — от англ. frame — «рама» или «рамка»). Фреймовая модель представления знаний основана на теории фреймов М. Минского, которая представляет собой систематизированную психологическую модель памяти человека и его сознания. С каждым фреймом ассоциируется разнообразная информация (в том числе и процедуры), например, информация о том, как пользоваться данным фреймом, каковы ожидаемые результаты выполнения фрейма, что делать, если ожидания не оправдались, и т. п.

На основе вышеизложенного можно сказать. Что фреймом называется структура данных для представления некоторого концептуального объекта. Фрейм можно представить в виде сети, состоящей из вершин и отношений (дуг).
«Верхние уровни» фрейма фиксированы и представляют сущности, всегда истинные в ситуации, описываемой данным фреймом.
«Нижние уровни» заканчиваются слотами (от англ. slot — поле), которые заполняются конкретной информацией при вызове фрейма. Можно провести аналогию между фреймами и описанием процедур в языках программирования [7; 8].



Ма́рвин Ли Ми́нски
американский учёный в области искусственного интеллекта
1927-2016

5. Фреймы.

Иерархическая структура фреймов

Фреймовая система как правило имеет иерархическую структуру, фреймы соединены друг с другом с помощью родовидовых связей. На верхнем уровне иерархии - фрейм, содержащий наиболее общую информацию, истинную для всех остальных фреймов. Фреймы обладают способностью наследовать значения характеристик своих родителей.

В общем случае, фрейм имеет следующую структуру:

Имя фрейма – для однозначной идентификации фрейма в системе – должно быть уникальным.

Слоты – описания, с помощью которых определяются основные структурные элементы этого понятия-фрейма. Слот имеет имя и значения слота. Слот может содержать несколько значений.

ИМЯ ФРЕЙМА
Имя 1-го слота: значение 1-го слота
Имя 2-го слота: значение 2-го слота
……
Имя N-го слота: значение N-го слота

СПИСОК СОТРУДНИКОВ
Фамилия: значение 1-го слота
Год рождения: значение 2-го слота
……
Специальность: значение N-го слота

5. Фреймы.

Фрейм соответствует описанию процедуры, а означенный фрейм (фрейм-пример) соответствует вызову процедуры.
Отличие фреймов от описаний процедур состоит в том, что фреймы могут вызываться не по имени, а по соответствию текущей ситуации той ситуации, которую описывает данный фрейм. Кроме того, фрейм, слоты и механизм их означивания описывают ситуацию в семантических (а не синтаксических) терминах и в метатерминах. С каждым слотом фрейма связаны описания условий, которые должны быть соблюдены, чтобы могло произойти означивание слота. В простейших случаях эти условия могут сводиться к указанию семантических категорий, которым должно удовлетворять значение слота. В более сложных случаях условия могут касаться отношений между значениями, выбираемыми для нескольких слотов [7, 8].
Итак, фрейм-пример может быть представлен в виде следующей конструкции:

$$f = [< r_1, v_1 >,< r_2, v_2 >, \ldots,< r_n, v_n >],$$

где f— имя фрейма; r_i — имя слота; v_i — значение слота. В качестве значений слотов могут выступать имена других фреймов, что обеспечивает связь между фреймами (так образуются сети фреймов) [7, 8].

Приставка мета- восходит к греческому μετά — «сверх». Обычно ее наличие свидетельствует о том, что речь идет о более высоком уровне понятия.

Система фреймов

Ту же запись можно представить в виде таблицы [24].

ИМЯ ФРЕЙМА			
Имя слота	Тип слота	Значение слота	Присоединенная процедура

В таблице дополнительный столбец предназначен для возможного присоединения к тому или иному слоту специальных процедур, что допускается в теории фреймов, и для указания типа слота [24].

Родственные фреймы связываются в ***систему фреймов***. Система содержит описание зависимостей (причинных, временных и т. п.) между входящими в нее фреймами. Для выражения указанных зависимостей фреймы, входящие в некоторую систему, имеют общее множество слотов.

Представление зависимостей в явном виде позволяет предсказать переход от одного состояния *A* (выражаемого фреймом *A´*) к другому зависимому от него состоянию *B* выражаемому фреймом *B´*) и осуществить этот переход эффективно, т. е. не вычисляя заново значений всех параметров, характеризующих состояние В, а перечислив только изменившиеся (или новые) параметры [7].

5. Фреймы.

Различают ***фреймы-образцы***, хранящиеся в базе знаний, и ***фреймы-экземпляры***, которые создаются для отображения реальных фактических ситуаций на основе поступающих данных [24]. Модель фрейма является достаточно универсальной, поскольку существуют не только фреймы для обозначения объектов и понятий, но и ***фреймы-роли*** (отец, мать, начальник, пешеход), ***фреймы-сценарии*** (лекция, собрание), ***фреймы-ситуации*** (тревога, авария, рабочий режим устройства) и др. [24].

Феноменологическая сила фрейм-представления во многом основывается на включении в него предположений и ожиданий. Слотам фрейма могут быть заранее приписаны, по умолчанию, некоторые стандартные значения. Это позволяет анализировать с помощью фреймов ситуации, в которых отсутствует упоминание о ряде деталей. Стандартные значения, присвоенные по умолчанию, не жестко связаны со своими слотами и могут быть замещены более подходящими значениями, если они найдены в обрабатываемой ситуации. Использование концепции «умолчания» часто позволяет исключить необходимость в кванторных утверждениях. Системы фреймов, в свою очередь, обычно организуют в информационно-поисковую сеть. Эта сеть используется в случаях, когда предложенный фрейм не удастся привести в соответствие с данной ситуацией, т. е. когда слотам не могут быть присвоены значения, удовлетворяющие условиям, связанным с этими слотами. В подобных ситуациях сеть используется для того, чтобы предложить какой-либо другой фрейм [7; 8].

5. Фреймы.

Необходимо отметить, что фрейм-представление (так же, как декларативное и процедурное представление) является не конкретным языком для представления знаний, а некой концепцией, реализуемой по-разному в таких конкретных языках, как KRL (Knowledge Representation Language) [34], FRL (Frame Representation Language) [37], K-NET и т. п.
В частности, фреймы могут быть представлены с помощью иерархических семантических сетей, как это сделано в K-NET, где некоторое пространство в сети рассматривается как фрейм, а дуги, связывающие это пространство с остальной сетью, рассматриваются как слоты [7; 8].

Важнейшим свойством теории фреймов является заимствованное из теории семантических сетей наследование свойств. И во фреймах, и в семантических сетях наследование происходит по **АКО-связям**. Наследование свойств может быть частичным [24].
Основным преимуществом фреймов как модели представления знаний является то, что она отражает концептуальную основу организации памяти человека, а также ее гибкость и наглядность [38].
Специальные языки представления знаний в сетях фреймов (FRL, KRL и др.) позволяют эффективно строить промышленные ЭС. Широко известны такие фрейм-ориентированные ЭС, как ANALIST, МОДИС [39].-

Отношение между надмножеством и подмножеством (называется АКО — «A Kind Of», «разновидность»). (Пример: «собака является животным» = тип с именем собака является подтипом типа животные).

6. Объектно-ориентированный подход.

Объектно-ориентированная парадигма.

Наиболее развитым способом представления знаний в ЭС является ***объектно-ориентированная парадигма***. Этот подход является развитием фреймового представления. В его основе лежат понятия ***объект*** и **класс** [7, гл. 5]; [7, прил. 1]. В реальном мире, а точнее в интересующей разработчика предметной области, в качестве объектов могут рассматриваться конкретные ***предметы***, а также абстрактные или реальные ***сущности***.

Например, объектами могут быть покупатель, фирма, производящая определенные товары, банк, заказ на поставку. Объект обладает индивидуальностью и поведением, имеет атрибуты, значения которых определяют его состояние. Так, конкретный покупатель, делая заказ, может оказаться в состоянии, когда денег на его счете не хватает для оплаты, а его поведение в этом случае заключается в обращении в банк за кредитом [7].

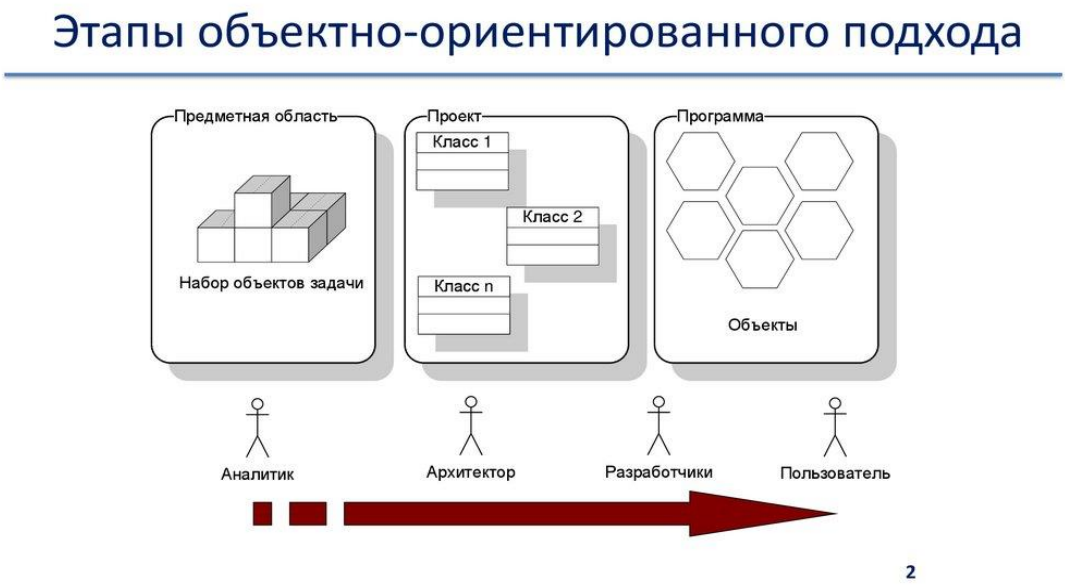
Каждый объект является представителем некоторого класса однотипных объектов. Класс определяет общие свойства для всех его объектов. К таким свойствам относятся состав и структура данных, описывающих атрибуты класса и соответствующих объектов, и совокупность методов — процедур, определяющих взаимодействие объектов этого класса с внешней средой. Например, описание класса «магазины» может включать такие атрибуты, определяющие состояние объектов, как название и адрес, которые индивидуальны для каждого объекта этого класса — конкретного магазина, штат сотрудников, размер текущего счета, а также методы: формирование заказов на поставку товаров, передача товара
со склада в торговую секцию и т. д. [7].

Свойства объектов и классов

Объекты и классы обладают характерными свойствами, которые активно используются при объектно-ориентированном подходе и во многом определяют его преимущества.

К этим ***свойствам*** относятся следующие [7]:

- инкапсуляция
- наследование
- полиморфизм



Свойства объектов и классов

- ***инкапсуляция*** — скрытие информации [7, гл. 5]. При объектно-ориентированном программировании имеется возможность запретить любой доступ к атрибутам объектов, доступ возможен только через его методы. Внутренняя структура объекта в этом случае скрыта от пользователя, объекты можно считать самостоятельными сущностями, отделенными от внешнего мира. Для того чтобы объект произвел некоторое действие, ему извне необходимо послать сообщение, которое инициирует выполнение нужного метода. Инкапсуляция позволяет изменять реализацию любого класса объектов без опасения, что это вызовет нежелательные побочные эффекты в программной системе. Тем самым упрощается процесс исправления ошибок и модификации программ;
- ***наследование*** — возможность создавать из классов новые классы по принципу «от общего к частному». Наследование позволяет новым классам при сохранении всех свойств классов-родителей (называемых в дальнейшем суперклассами) добавлять свои черты, отражающие их индивидуальность. С точки зрения программиста новый класс должен содержать только коды и данные для новых или изменяющихся методов. Сообщения, обработка которых не обеспечивается собственными методами класса, передаются суперклассу. Наследование позволяет создавать иерархии классов и является эффективным средством внесения изменений и дополнений в программные системы [7];
- ***полиморфизм*** — способность объектов выбирать метод на основе типов данных, принимаемых в сообщении. Каждый объект может реагировать по-своему на одно и то же сообщение. Полиморфизм позволяет упростить исходные тексты программ, обеспечивает их развитие за счет введения новых методов обработки [7].

Примеры объектно-ориентированного программирования на языке Python

Представьте сценарий, где вам нужно разработать болид Формулы-1 используя подход объектно-ориентированного программирования. Первое, что вам нужно сделать — это определить реальные объекты в настоящей гонке Формула-1. Какие аспекты в Формуле-1 обладают определенными характеристиками и могут выполнять ту или иную функцию?
Один из очевидных ответов на этот вопрос — гоночный болид. Условный болид может обладать такими характеристиками как:

- **мощность двигателя;**
- **марка;**
- **модель;**
- **производитель**, и т. д.

Соответственно, болид можно запустить, остановить, ускорить, и так далее. Гонщик может быть еще одним объектом в Формуле-1. Гонщик имеет национальность, возраст, пол, и так далее, кроме этого, он обладает таким функционалом, как управление болидом, рулевое управление, переключение передач.

Как и следует из названия, объектно-ориентированное программирование — это речь об объектах. Однако, перед тем как создать объект, нам нужно определить его класс.

Примеры объектно-ориентированного программирования на языке Python

Класс в объектно-ориентированном программировании выступает в роли чертежа для объекта. Класс можно рассматривать как карту дома. Вы можете понять, как выглядит дом, просто взглянув на его карту. Сам по себе класс не представляет ничего. К примеру, нельзя сказать что карта является домом, она только объясняет как настоящий дом должен выглядеть.
Отношение между классом и объектом можно представить более наглядно, взглянув на отношение между машиной и Audi. Да, Audi – это машина. Однако, нет такой вещи, как просто машина. Машина — это абстрактная концепция, которую также реализуют в Toyota, Honda, Ferrari, Волга и других компаниях.
Ключевое слово **class** используется для создания класса в **Python**. Название класса следует за ключом **class**, за которым следует двоеточие. Тело класса начинается с новой строки, с отступом на одну вкладку влево. Давайте рассмотрим, как мы можем создать самый простой класс в Python. Взглянем на следующий код:

```
1  # Создаем класс Car
2  class Car:
3
4      # создаем атрибуты класса
5      name = "c200"
6      make = "mercedez"
7      model = 2008
8
9      # создаем методы класса
10     def start(self):
11         print ("Заводим двигатель")
12
13     def stop(self):
14         print ("Отключаем двигатель")
```

В примере выше мы создали класс под названием Car с тремя атрибутами: имя **name**, марка **make** и модель **model**. Наш класс также содержит два метода: **start**() и **stop**().

https://python-scripts.com/object-oriented-programming-in-python

Примеры объектно-ориентированного программирования на языке Python

Раннее мы поняли, что класс предоставляет чертеж **объекта**. Однако, чтобы на самом деле использовать объекты и методы класса, вам нужно создать объект из этого класса. Существует несколько методов и атрибутов класса, которые можно использовать вне объекта, мы рассмотрим их в следующем разделе.
Сейчас просто запомните, что по умолчанию, нам нужно создать объект класса перед тем, как мы сможем начать использовать его методы и атрибуты.
Объект также называется **экземпляром**. Тем не менее, процесс создания объекта класса называется инициализация. В Python, чтобы создать объект класса, нам просто нужно вписать название класса, с последующими открывающимися и закрывающимися скобками.
Давайте создадим объект класса Car, который мы создали в предыдущем примере.

```
1  # Создаем объект класса Car под названием car_a
2  car_a = Car()
3
4  # Создаем объект класса Car под названием car_b
5  car_b = Car()
```

В этом скрипте мы создали два объекта класса Car: **car_a** и car_b. Чтобы узнать тип созданных нами объектов, мы можем использовать метод type и передать ему названия наших объектов.
Выполните следующий код:

```
1  print(type(car_b))
```

Примеры объектно-ориентированного программирования на языке Python

В выдаче вы увидите:

```
<class '__main__.Car'>
```

Это говорит нам о том, что тип объекта car_b – класс Car.
На данный момент мы создали наш класс и соответствующие ему объекты. Теперь настало время получить доступ к атрибутам класса и вызвать метод класса при помощи объекта класса. Чтобы сделать это, вам нужно только вписать имя объекта, за которым следует оператор точка . и название атрибута или метода, к которому вы хотите получить доступ или вызов, соответственно. Давайте рассмотрим следующий пример:

```
car_b.start()
```

В этом скрипте мы вызываем метод start() через объект car_b. Выдача будет выглядеть следующим образом:

```
Заводим двигатель
```

Аналогично, вы можете получить доступ к атрибуту, пользуясь следующим синтаксисом:

```
print(car_b.model)
```

В выдаче вы увидите значение атрибута модели, как показано ниже:

```
2008
```

Вывод

Итак, объектно-ориентированный подход заключается в представлении системы в виде совокупности классов и объектов предметной среды. При этом иерархический характер сложной системы отражается в виде иерархии классов, а ее функционирование рассматривается как взаимодействие объектов, с которыми ассоциируются, например, продукционные правила. Ассоциирование продукционных правил ЭС с иерархией классов осуществляется за счет использования общих правил, в качестве префикса которых обычно используется ссылка на класс, к которому данное правило применимо. Указанный префикс, с точки зрения декларативного представления знаний, семантически подобен квантору все-общности в исчислении предикатов [7].

7. Практика использования моделей представления знаний в экспертных системах

Применение продукционных правил

Для того чтобы охарактеризовать представление знаний, используемое в некоторой системе, необходимо определить представление данных в рабочей памяти и представление знаний в базе знаний. В большинстве систем из соображений эффективности в базе знаний хранятся не только изолированные правила (как предписывает теория), но и некоторая связная модель, характеризующая проблемную среду в целом.

В связи с разным назначением правил и модели обычно для их задания используются различные представления. Ниже будут рассмотрены наиболее типичные подходы, используемые в экспертных системах при представлении данных, правил и моделей проблемной области [7].

При этом подходе данные в рабочей памяти представляются в виде изолированных троек: «объект — атрибут — значение».

С каждой тройкой связан коэффициент определенности. Иерархия объектов задается с помощью дерева контекстов. Это дерево может рассматриваться как фрейм, который обеспечивает механизм наследования. В дереве контекстов каждому объекту приписаны соответствующие ему атрибуты. Впервые этот подход был использован в ЭС MYCIN [8]. Ниже приведен способ представления продукционных правил в типичной ЭС [7]:

Применение продукционных правил

<правило>::= (ЕСЛИ <условие> ТО <действие> ИНАЧЕ <действие>)
<условие>::= (И {<предложение>}*)
<предложение>::= (ИЛИ {<предложения>}*) |
(<предикат><тройка>)
<тройка>::= (<объект><атрибут><значение>)
<действие>::= {<заключение>}* | <процедура>
<заключение>::= (<тройка><коэффициент определенности>)

Здесь при истинности условия выполняется действие, стоящее за указателем **ТО**, а при ложности — действие, стоящее за указателем **ИНАЧЕ**. Сущности, помеченные звездочкой, могут появиться в правиле один или более раз. Например, условие есть конъюнкция одного или более предложений, а предложение есть либо дизъюнкция одного или более предложений, либо предикат, примененный к тройке: «объект — атрибут — значение».

Использование семантических сетей

При этом подходе (см., например, ЭС PROSPECTOR [40]) данные в рабочей памяти и базе знаний представлены не в виде изолированных троек «объект — атрибут — значение», а в виде семантической сети. Отметим, что семантическая сеть может рассматриваться как обобщение представления в виде троек, допускающее представление любого *n*-арного отношения над произвольным числом объектов. Один из возможных способов представления правил в семантической сети [7]:

<правило>::= (ЕСЛИ <условие> ТО <мера И> <мера Л>
< действие>)
<условие>::= <утверждение>
<утверждение>::= <логическое утверждение> |
<описательное утверждение>
<логическое утверждение>::= (И {<утверждение>}*) |
(ИЛИ {<утверждение >}*) I
(НЕТ <утверждение>)
<действие>::= <описательное утверждение>

Использование семантических сетей

Правила связывают условие и действие, являющиеся утверждениями о проблемной области. Утверждения могут быть истинными и ложными. Утверждения делятся на логические и описательные. ***Логическое утверждение*** является булевской комбинацией других утверждений. Действие и терминальная компонента ***логического утверждения*** всегда являются описательными утверждениями, которые в общем случае представляют собой фрагмент семантической сети [7].

Для учета неопределенности во многих ЭС вводится коэффициент определенности утверждений.

Коэффициент определенности логического утверждения вычисляется по правилам логики размытых множеств [41].

Коэффициент определенности описательных утверждений либо указывается пользователем (для исходных данных), либо (для выведенных утверждений) вычисляется по правилу Байеса [42]. В случае неопределенности каждое правило содержит две меры, используемые для модификации коэффициента определенности утверждения, выведенного этим правилом. Первая из мер используется в том случае, если условие определено как ***истинное***, а вторая — если условие ***ложно***. Условие оценивается как истинное, если его коэффициент определенности лежит в диапазоне: (0;+1].

При изменении коэффициента определенности некоторого условия применяется ***правило***, и на основе процедуры Байеса подсчитывается коэффициент определенности действия (описательного утверждения) соответствующего правила [7].

Использование фреймов

При этом подходе (см., например, ЭС RLL [43]) в виде фреймов обычно представляются все виды данных и знаний, более сложных, чем список значений: объекты, правила, слоты фреймов, механизмы выводов и т. п. Фреймы организуются в сеть. Ниже приведен пример представления правила в виде фрейма [7]:

Тип	Правило
Описание	Сообщить пользователю об опасности для его здоровья, если имеется химическая токсичность
ЕСЛИ потенциально релевантен	(Химическая активность высока?)
ЕСЛИ истинно релевантен	(Близость химически активного места и пользователя?)
ТО сказать пользователю	«Не дышать, химически активные вещества!»
ТО добавить к агенде	(<соответствующие системные указания>)
Приоритет	Высокий
Среднее время выполнения:	0,1 с
Частота использования:	Рассматривалось 985 раз, использовалось 4 раза

Представление правила в виде фрейма

Представление правила в виде фрейма имеет следующие отличия [7]:

1) наличие не одного, а нескольких условий и действий для разных уровней анализа правила;
2) наличие большого количества описательных (невыполняемых) слотов, которые могут быть использованы метаправилами для управления.

Необходимо обратить внимание на то, что информация о правиле хранится в нескольких слотах. Это позволяет использовать различные интерпретаторы правил в зависимости от целей разработчика.

Например, один дешевый интерпретатор, предназначенный для быстрой обработки, может на основе анализа слота «ЕСЛИ потенциально релевантен» определять число подходящих правил и уже затем принимать решение в зависимости от полученного результата.

Второй, более традиционный интерпретатор оценивает все слоты «ЕСЛИ» всех правил и затем, если число подходящих правил больше одного, будет разрешать конфликт, выбирая единственное правило.

Третий интерпретатор, оценив сначала, сколько времени имеется на решение задачи, может устранить из рассмотрения все правила, имеющие высокое значение слота «Среднее время выполнения», проанализировать далее слоты «Если потенциально релевантен» и, если правил все еще много, устранить те из них, которые определяют новые фреймы, а для оставшихся правил оценить все слоты «ЕСЛИ» и выбрать единственное правило.

Использование управляемых образцами модулей

Механизм использования управляемых образцами модулей проиллюстрируем на примере инструментальной системы HEARSAY-III [44], предназначенной для проектирования ЭС и являющейся модификацией системы понимания речи HEARSAY-II [7].

Все рассмотренные ранее способы представления знаний использовали частный случай управляемых образцами модулей. Действительно, каждый модуль представлялся в виде продукционного правила. Сложность правил была весьма ограничена, что позволяло выразить их в виде, понятном эксперту.

Если же преобразования, выполняемые модулем, очень сложны, то для их представления приходится прибегать к процедурной форме. Стремление сохранить независимость модулей друг от друга привело к созданию в HEARSAY-III схемы, обеспечивающей взаимодействие модулей не непосредственно, а только через рабочую память, называемую «классная доска».

Модули в HEARSAY-III называются источниками знаний (ИЗ). Каждый источник знания состоит из программы-условия, которая определяет, применим ли ИЗ к текущему состоянию классной доски, и программы-действия, производящей результаты [7].

Использование управляемых образцами модулей

Классная доска разделена на несколько уровней, на каждом из которых обрабатываются данные определенного вида. Так, в системе HEARSAY-II выделены следующие уровни: предложение, словосочетание, слово, слог, фонема и т. д. Поиск решения рассматривается системой как итеративный процесс, состоящий из выдвижения гипотез и проверки их правдоподобности. Текущее состояние решения представляется в виде гипотез на классной доске. Гипотеза представляет собой интерпретацию некоторой части устного высказывания на определенном уровне. Гипотезы различных уровней объединены в направленный граф (сеть), что позволяет описывать гипотезы высшего уровня через гипотезы более низкого уровня [7].

Итак, в HEARSAY-III рабочая память представляется в виде сети, а знания о проблемной среде — в виде модулей, вызываемых по образцу. Использование программ, вызываемых по образцу, является шагом в направлении к процедурному представлению с попыткой сохранить независимость источников знания.

Использование управляемых образцами модулей

Подобный подход (в отличие от использования продукций и сетей) позволяет решать значительно более сложные задачи, но уменьшает возможности по объяснению и приобретению новых знаний. Использование программ, вызываемых по образцу, требует разработки для каждой предметной области своего специфического решателя, осуществляющего планирование процесса решения и использование знаний. Применение правил в виде продукций, фреймов, сетей позволяет создавать системы, ориентированные на определенный класс задач, сохранив способности к объяснению и приобретению знаний. Однако малая мощность подобных правил приводит к резкому снижению эффективности при решении сложных задач. Так, например, экспериментальная попытка представить часть HEARSAY-II в виде продукционных правил привела к замедлению работы примерно в 1000 раз.

Общим для всех рассмотренных подходов является использование образцов при вызове модуля или правила [7].

Смешанные представления (объекты и правила)

Как правило, в экспертных системах используется не одно, а несколько представлений. Исполняемые утверждения представляются либо в виде продукционных правил, либо в виде модулей (процедур), вызываемых по образцу. Для представления модели предметной области используются объектный подход или сетевые модели (семантические сети и фреймы) [7].

Главное преимущество использования объектно-ориентированного программирования при разработке систем обработки данных заключается в поддержке методов, облегчающих повторное использование кода. Однако, как отмечают многие исследователи, эффект от внедрения объектно-ориентированной технологии программирования начинает проявляться лишь через 5–8 лет. Это обусловлено необходимостью накопления опыта разработок и формирования устойчивой и достаточно гибкой иерархии классов. Очевидно, что подобные издержки неприемлемы для инструментальных средств инженерии знаний, где одним из определяющих требований является необходимость создания «быстрого прототипа». Поэтому объектно-ориентированный инструментарий для создания систем, основанных на знаниях, должен включать и библиотеку стандартных, но достаточно легко модифицируемых объектов [7].

Вывод

Применение объектно-ориентированного подхода в системах инженерии знаний выводит на первый план другую его особенность, а именно возможность естественной декомпозиции задачи на совокупность подзадач, представляемых достаточно автономными агентами, работающими со знаниями.

На сегодняшний день это единственная практическая возможность работы в условиях экспоненциального роста сложности (количества взаимосвязей), характерного для систем, использующих знания. Так, практически все инструментальные средства для создания динамических ЭС поддерживают объектно-ориентированный подход к проектированию систем, объединенный с правилами [7, гл. 9].

Выводы

1. Фреймом называется структура данных для представления некоторого концептуального объекта. Фрейм можно представить в виде сети, состоящей из вершин и отношений (дуг).
2. Фреймовая система как правило имеет иерархическую структуру, фреймы соединены друг с другом с помощью родо-видовых связей. На верхнем уровне иерархии - фрейм, содержащий наиболее общую информацию, истинную для всех остальных фреймов. Фреймы обладают способностью наследовать значения характеристик своих родителей.
3. Важнейшим свойством теории фреймов является заимствованное из теории семантических сетей наследование свойств. И во фреймах, и в семантических сетях наследование происходит по АКО-связям. Наследование свойств может быть частичным.
4. Наиболее развитым способом представления знаний в ЭС является объектно-ориентированная парадигма. Этот подход является развитием фреймового представления. В его основе лежат понятия объект и класс .
5. Применение объектно-ориентированного подхода в системах инженерии знаний выводит на первый план другую его особенность, а именно возможность естественной декомпозиции задачи на совокупность подзадач, представляемых достаточно автономными агентами, работающими со знаниями.

Литература

1. Марселлус Д. Программирование экспертных систем на Турбо-Прологе : пер. с англ. / предисл. С. В. Трубицина. М. : Финансы и статистика, 1994. 256 с. : ил.
2. Рот М. Интеллектуальный автомат : компьютер в качестве эксперта : пер. с нем. М. : Энергоатомиздат, 1991. 80 с. : ил.
3. Экспертные системы на персональных компьютерах : материалы семинара. М. : МДНТП, 1990. 140 с.
4. Экспертные системы : принципы работы и примеры / А. Брукинг [и др.]; под ред. Р. Форсайта; пер. с англ. С. И. Рудаковой; под ред. В. Л. Стераника. М. : Радио и связь, 1987. 220 с. : ил.
5. Экспертные системы : сборник / ред. Б. М. Васильев. М. : Знание, 1990. 47 с. : ил.
6. Убейко В. М., Убейко В. В. Экспертные системы в СССР. Обзор информ. // Маш. пр-во. Сер. Автоматизированные системы проектирования и управления, вып. 5 (СНИИТЭМР). М., 1991. 67 с.
7. Статические и динамические экспертные системы : учеб. пособие для вузов / Э. В. Попов, И. Б. Фоминых, Е. Б. Кисель, М. Д. Шапот. М. : Финансы и статистика, 1996. 320 с. : ил.
8. Попов Э. В. Экспертные системы. Решение неформализованных задач в диалоге с ЭВМ. М. : Наука, 1987. 288 с. : ил.
9. Попов Э. В. Общение с ЭВМ на естественном языке. М. : Наука, 1982. 360 с.
10. Мендельсон Э. Введение в математическую логику. М. : Наука, 1971. 320 с.
11. Попов Э. В., Фридман Г. Р. Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. М. : Наука, 1976. 455 с.
12. Мальцев А. А. Алгоритмы и рекурсивные функции. М. : Наука, 1965. 391 с.
13. Маслов С. Ю. Обратный метод установления выводимости в классическом исчислении предикатов // Доклады академии наук СССР. 1964. Т. 159, № 1. С. 17–20.
14. Ефимов Е. И. Решатели интеллектуальных задач. М. : Наука, 1982. 320 с.
15. Финн В. К. Индуктивные модели // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. А. С. 58–76.
16. Поспелов Д. А. Элементы аксиоматики временных отношений // Вопросы кибернетики. Ситуационное управление. Теория и практика. М. : Научный совет по комплексной проблеме кибернетики АН СССР, 1975. С. 15–21.
17. Поспелов Д. А. Логико-лингвистические модели в системах управления. М. : Энергоатомиздат, 1981. 231 с.
18. Поспелов Д. А. Псевдофизические логики // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. А. С. 48–57.

Литература

19. Поспелов Д. А. О «человеческих» рассуждениях в интеллектуальных системах // Вопросы кибернетики. Логика рассуждений и ее моделирование. М. : Научный совет по комплексной проблеме «Кибернетика», 1983. С. 5–37.
20. Поспелов Д. А. Логические модели представления знаний. Вводные замечания // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. А. С. 33–35.
21. Крипке С. А. Теорема полноты в модальной логике // Модальная логика. Фейс Р. М. : Наука, 1974. С. 223–246.
22. Крипке С. А. Семантический анализ модальной логики. Ч. I. Нормальные модальные исчисления высказываний // Модальная логика. Фейс Р. М. : Наука, 1974. С. 254–303.
23. Крипке С. А. Семантический анализ модальной логики. Ч. II. Нормальные модальные исчисления высказываний // Модальная логика. Фейс Р. М. : Наука, 1974. С. 304–323.
24. Гаврилова Т. А., Червинская К. Р. Извлечение и структурирование знаний для экспертных систем. М. : Радио и связь, 1992. 200 с. : ил.
25. The OPS_5 user's manual. Technical Rept. CMU-CS-81. Pittsburgh : Carnegie — Mellon University, 1981.
26. Walker C. T., Miller K. R. Expert system 1987 : An Assessment of Technology and Applications. Madison, 1988.
27. Кирсанов Б. С., Попов Э. В. Отечественные оболочки экспертных систем для больших ЭВМ // Искусственный интеллект : справочник. М. : Радио и связь, 1990. Т. 1. С. 369–388.
Т. 1. С. 369–388.
28. Хорошевский В. Ф. Программный инструментарий представления знаний в экспертных системах // Экспертные системы : состояние и перспективы; под ред. Д. А. Поспелова. М. : Наука, 1989. С. 38–47.
29. Ковригин О. В., Перфильев К. Г. Гибридные средства представления знаний в системе СПИЭС // Всесоюзная конференция по искусственному интеллекту : тез. докл. Переславль-Залесский, 1988. Т. 2. С. 490–494.
30. Соловьев С. Ю., Соловьева Г. М. Вопросы организации баз знаний в системе ФИАКР // Экспертные системы : состояние и перспективы / под ред. Д. А. Поспелова. М. : Наука, 1989. С. 47–54.
31. Цейтин Г. С. Программирование на ассоциативных сетях // ЭВМ в проектировании и производстве. Л. : Машиностроение, 1985. Вып. 2. С. 16–48.
32. Сапатый П. С. Язык ВОЛНА-0 как основа навигационных структур для баз знаний на основе семантических сетей // Изв. АН СССР. Техническая кибернетика. 1986. № 5. С. 198–211.
33. Уварова Т. Г. Операционная семантика волновых языков и метод ее описания // Изв. АН СССР. Техническая кибернетика. 1987. № 2. С. 128–142.
34. Уотермен Д. Руководство по экспертным системам : пер. с англ. М. : Мир, 1989. 388 с.

На следующих лекциях будут изучены

Спасибо за внимание!