УДК 004.82

ПРИМЕНЕНИЕ МЕТОДОВ ИНЖЕНЕРИИ ЗНАНИЙ В КЛАССИЧЕСКОМ ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

А. А. Рафанович

Национальный институт здравоохранения, Бетезда, Мериленд, США

Поступила в редакцию 20.05.2013 г.

Аннотация. Статья посвящена описанию модели интеграции комбинированных методов инженерии знаний традиционных семантического, фреймового и продукционного подходов в классическое объектно-ориентированное программирование. Приведены фрагменты программной реализации экспертной системы автоматического решателя задач на основе рассматриваемой модели.
Ключевые слова: компьютерные технологии, инженерия знаний, искусственный интеллект, экспертные системы, объектно-ориентированное программирование.
Annotation. The paper is devoted to the approach to integration of traditional knowledge engineering methods like semantic networks, frames and production rules to the classical object-oriented programming. The article contains fragments of the program implementation of automatic task solver expert system based on described model.
Keywords: computer technologies, knowledge engineering, artificial intelligence, expert systems, object-oriented programming.

1. ВВЕДЕНИЕ

Одним из интересных направлений исследований в инженерии знаний (ИЗ) – областинаук об искусственном интеллекте (ИИ), экспертных системах (ЭС) и базах знаний (БЗ), являютсяисследования в области моделей организации и внутрисистемного представления данных и методов их обработки (Джексон, 2001). Данная область исследований выходит за рамки ИЗ и применяется также в исследованиях по базам данных и т.п. – однако в этой статье мы сделаем упор на модели и методы ИЗ в ИИ.

На сегодняшний момент можно выделить несколько широко известных подходов к представлению знаний. Это логическая, семантическая, фреймовая и продукционная модели (Гаврилова & Хорошевский, 2000). Не вдаваясь в описание каждой из них, отметим, что, в целом, разработка или выбор одной из существующих моделей при проектировании экспертной системы или просто любого программного продукта, обладающего чертами ИИ,имеет первостепенное значение. В данной статье мы хотели бы показать принципиальную возможность привнесения элементов ИЗвклассическое объектно-ориентированное программирование, ставшее на сегодня стандартом де-факто при разработке традиционногопрограммного обеспечения.

Оставляя за скобками логическую модель представления знаний, для которой были разработаны специализированные средства программирования, к примеру язык Пролог, сосредоточимся на оставшихся трех и покажем, что они на удивление хорошо вписывающихся в стандарты ООП (Грэхем, 2004) –попробовав объединить лучшие их черты для решения самых разнообразных задач.

2. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ПРЕДСТАВЛЕНИЮ ЗНАНИЙ

Развитие программирования, как логического, так и функционального, подчинено определенной логике стремления разработчиков программных систем к мимикрированию человеческого мышления и человеческих методов структурирования информации. С ростом возможностей аппаратного обеспечения ветви логического и функционального подходов к программированию будут все более сближаться – и, возможно, сольются.

Отметим, что важными элементами обоих вариантов являются возможность протоколи-

рованияимеющихся в системе знаний и хода решения поставленных задач по требованию пользователя. В этой связи мы не будем включать в рассмотрение стоящие особняком нейросетевые системы как принципиально отличный подход – в целом работающий по принципу черного ящика. По нашему мнению, применение нейронных сетей, в основном, целесообразно в случаях неформализуемости или ограниченности входных данных, как дополнение к классическим методам программирования.

Таким образом, стандартизация, сведение к единому базису подходов к представлению знаний и процессу принятия решений, улучшение технических характеристик программно-аппаратных систем, строящихся на основе указанных выше моделей и теоретических подходов, является одной из наиболее интересных и актуальных проблем в исследованиях по искусственному интеллекту и практическому программированию в целом.

Вынесем на рассмотрение объектно-ориентированный подход к представлению знаний, основанный на комбинации некоторых черт из трех отобранных моделей, и примеры его использования при решении логических задач. При разработке были использованы результаты, полученные в области объектно-ориентированного программирования и экспериментальных исследований в области объектно-ориентированных баз данных.

Сразу заметим, что семантическая и фреймовая модели почти полностью вписываются в стандарты ООП. Прямым аналогом фрейма в ООП можно считать класс, аналогом связей семантических сетей – принципы наследования. Продукционная же модель легко реализуема при помощи несложного интерфейса на основе рекурсии.

Тем не менее, описание полноценной семантической сети лишь при помощи механизма наследования классов ООП невозможно по причине того, что такая информация: во-первых, формируется не динамически, а жестко прописывается в программе на этапе разработки; во-вторых, множественное наследование в языках ООП часто имеет ряд ограничений; и, в-третьих, затруднен программный доступ к самой сети наследования.

Попытаемся определить, какие требования необходимо наложить на модель знаний, если мы хотим предоставить программисту возможность представления и выборки абстрактных данных.

3. ОПРЕДЕЛЕНИЯ БАЗОВЫХ ЭЛЕМЕНТОВ МОДЕЛИ

Итак, под *типом данных* $t \in T$, где T – пространство типов данных, будем понимать пару, где:

- *n: наименование типа;*
- *P: множество прямых предков типа, где* $P = \{p_1 \ldots p_N\} \subseteq T$.

На множестве типов T рекурсивно вводятся операции сравнения:

$$\forall t_1, t_2 \in T t_1 = t_2 \leftrightarrow [\![n(t]\!]_1) = [\![n(t]\!]_2)$$
$$\forall t_1, t_2 \in T t_1 \geq t_2 \leftrightarrow t_1 = t_2 \vee \exists p_1 \in P(t_1) : p_1 \geq t_2$$
$$\forall t_1, t_2 \in T t_1 \geq t_2 \leftrightarrow t_2 \geq t_1$$
$$\forall t_1, t_2 \in T t_1 > t_2 \leftrightarrow \exists p_1 \in P(t_1) : p_1 \geq t_2$$
$$\forall t_1, t_2 \in T t_1 > t_2 \leftrightarrow t_2 < t_1.$$

В случае если $t_1 > t_2$, будем говорить, что t_1 является наследником от t_2, или включает в себя поведенческий аспект типа t_2. Понятно, что при указанных ограничениях на пространстве T запрещено циклическое наследование типа.

Под *атрибутом* a будем понимать пару $a =< n, t >$, где:

- *n: имя атрибута;*
- *t: тип атрибута, принадлежащий объединению пространства типов T с системными примитивами (обычно разные числовыетипы, строки, булевы значения и т.п.)*

Далее, следуя терминологии ООП,определим основной элемент данных модели как тройку $c =< n, t, A >$, и будем называть ее *классом*. Здесь:

- *n: имя класса;*
- *t: тип класса из пространства типов данных T;*
- $A = \{a_1 \ldots a_N\}$ *: множество атрибутов класса.*

Под *объектом*будем пониматькласс с означенными атрибутами. При этом атрибуты означиваются строго в соответствии со следующими правилами:

- *любой атрибут может быть означен специальным обьектом NULL, что означает отсутствие на текуший момент в системе данных по атрибуту;*
- если тип атрибута $t_1 \in T$, то атрибут может быть означен только объектом тождественного типа либо его наследником, т.е. объектом типа $t_2 \in T : t_2 \geq t_1$;

• если типом атрибута является *системный примитив*, то атрибут может быть означен только соответствующимисистемными данными;

4. МЕХАНИЗМ ВЫВОДА

Чтобы завершить определение модели представления знаний, введем понятие продукционного правила. *Продукционное правило* – это механизм вывода новых данных на основе анализа существующих. Обозначим все множество продукционных правил модели как P.

Набор всевозможных типов, объектови продукционных правил будем называть *Виртуальным Миром* $=< T, O, P >$, где T, O, P – соответствующие единицы модели.

Работа системы основывается на последовательном вызове продукционных правил, до тех пор пока:

• Одно из продукционных правил отработает и, изменив данные, сообщит об успехе выполнения. В этом случае, если:

– Все неозначенные переменные объектов данных означены, то решение найдено;

– В противном случае создаем копию виртуального мира и на ней снова рекурсивно начинаем исполнение продукционных правил;

– Исключение – если виртуальный мир совпадает с одним из базовых миров, это означает наличие цикла в рассуждениях системы. В этом случае возвращаемся на один шаг рекурсии вверх и начинаем исполнение продукционных правил для базового мира;

• Всепродукционные правила вернут отрицательный результат; В этом случае, если:

– Существует базовый виртуальный мир, то возвращаемся на один шаг рекурсии вверх и заново начинаем исполнение продукционных правил для базового мира;

– В противном случаев рамках изначально заданного виртуального мира задача не имеет решения.

5. МЕХАНИЗМ КОНКРЕТИЗАЦИИ

Основным элементом механизма вывода представим понятиеспециального вида классов –*маски*. Маску можно определить как примитив доступа к данным системы на основе рекурсивного сравнения заданного шаблона и, последовательно, всех актуальных объектов данных. Объект данных считаетсясоответствующим маске, если выполняются все следующие условия:

1. Для любого атрибута маски найдется как минимум один означенный атрибутобъекта данных того же, либо наследуемого типа;

2. Если атрибут маски означен, его значение должно совпадать со значением соответствующегоатрибутаобъекта;

3. Если атрибут маски представляет собой объект, происходит его рекурсивное сравнение с соответствующим ему подобъектом основного объекта;

4. Все атрибуты объекта данных, которым не нашлось соответствия в маске игнорируются;

Кроме того, процесс наложения маски в рамках продукционного правила полагается обладающимследующим свойством итеративности: если при наложении маске соответствует несколько объектов данных то, в рамках одного виртуального мира, они полагаются соответствующими маске последовательно. Т.е. при каждом последующем вызове одного правила виртуального мира мы получаем новый результирующийобъект либо его отсутствие, если предыдущий возвращённый объект был последним которому подошла маска.

В качестве примера, схематично опишем решение пары задач разных классов в рамках единственной экспертной системы построенной на основе рассматриваемой модели данных: простой классической головоломки “волк, коза, капуста”.При описании будем придерживаться синтаксиса языка C++.

6. ПРИМЕР: КЛАССИЧЕСКАЯ ГОЛОВОЛОМКА

Для решения первой головоломки на основании базовых знаний об условиях задачи определим:

```
//Виртуальныймир
CVirtualWorld vw = new CVirtualWorld();

//Семантическая сеть типов данных
vw->add_type(newCVwType( “Сущность”) );
vw->add_type( new CVwType( “Место”, set( “Сущность” ) ) );
vw->add_type( new CVwType( “Предмет”, set( “Сущность”) ) );
vw->add_type( new CVwType( “Берег”, set( “Место” ) ) );
vw->add_type( new CVwType( “Берег”, set(“Предмет”) ) );
vw->add_type( new CVwType( “Животное”, set(“Предмет”) ) );
```

```
vw->add_type( new CVwType( “Овощь”, set(“Предмет”) ) );
vw->add_type( new CVwType( “Транспорт”, set(“Предмет”) ) );
vw->add_type( new CVwType( “Волк”, set(“Животное”) ) );
vw->add_type( new CVwType( “Коза”, set(“Животное”) ) );
vw->add_type( new CVwType( “Капуста”, set(“Овощь”) ) );
vw->add_type( new CVwType( “Лодка”, set( “Место” ) ) );
vw->add_type( new CVwType( “Лодка”, set(“Транспорт”) ) );

//Начальноесостояние:
CVwObject *bereg_r = vw->add_object( new CVwObject( “Берег” ) ); bereg_r->add_property(“Имя”, “Правый” );
CVwObject*bereg_l = vw->add_object( new CVwObject( “Берег” ) ); bereg_l->add_property( “Имя”, “Левый” );

CVwObject*volk = vw->add_object( new CVwObject( “Волк” ) ); bereg_l->add_property( “Содержит”, volk );
CVwObject*koza = vw->add_object( new CVwObject( “Коза” ) ); bereg_l->add_property( “Содержит”, koza );
CVwObject*kausta = vw->add_object( new CVwObject( “Капуста” ) ); bereg_l->add_property( “Содержит”, kapusta );
CVwObject*chelovek = vw->add_object( new CVwObject( “Человек” ) ); bereg_l->add_property( “Содержит”, chelovek );
CVwObject *lodka = vw->add_object( new CVwObject( “Лодка” ) ); bereg_l->add_property( “Содержит”, lodka );

Выборочно определимпродукционныеправила:
//Первое правило с подробными пояснениями
//Человек, находясь на берегу, может погрузить предмет в лодку
bool pogruzka_predmeta( CVirtualWorld *vw)
{
    CVwObjectMask *bereg_mask = new CVwObjectMask( “Берег” );

bereg_mask->add_property_type( “Содержит”, “Предмет” );
    bereg_mask->add_property_type( “Содержит”, “Человек” );
    bereg_mask->add_property_type( “Содержит”, “Лодка” );

//ищем берег с предметом, человеком и лодкой
//в соответствии с принципом итеративности наложения маски
//она последовательнонакладывается на объекты bereg_r и bereg_l
//при каждом последуюшем вызове продукционного правила porgruzka
//в итоге, с учетом совпадения атрибутов, переменная bereg
//означиваетсяобъектом bereg_l
    CVwObject *bereg = vw->apply_mask( bereg_mask );
    if( bereg )
    {
        // если есть, перемещаем атрибуты с берега на лодку
        CVwObject*lodka = bereg->get_property
( “Содержит”, “Лодка” );
CVwObject*predmet = bereg->get_property
( “Содержит”, “Предмет” );
CVwObject*chelovek = bereg->get_property
( “Содержит”, “Человек” );

        lodka->add_property( predmet ); bereg ->del_property( predmet );
        lodka->add_property( chelovek ); bereg ->del_property( chelovek );

//правило применено, возвращаемся в ядро системы,
//которое создаст нам копию виртуального мира и
//инициирует исполнение продукционных правил
//на созданной копии
returntrue;
    }

    //правило неприменено, итеративно повторить или
//перейти к слеуюшему правилу
    returnfalse;
}

//аналогично выгрузка
boolvygruzka(vw){...}
```

```
bool perevozka( CVirtualWorld *vw )
{
    CVwObjectMask *bereg_mask = new
CVwObjectMask( “Берег” );
    bereg_mask->add_property_type( “Со-
держит”, “Лодка” );

    CVwObject *bereg1 = vw->apply_mask(
bereg_mask );
    if( bereg1 )
    {
            //если в лодке есть человек,
  //она может переплыть на другой берег
            CVwObject*lodka = bereg1->
  get_property( “Содержит”, “Лодка” );
  CVwObject*chelovek = lodka->
  get_property( “Содержит”, “Человек” );

  CVwObject *bereg2 = vw->
  apply_mask(new CVwObjectMask( “Берег” )
);
  if( bereg2 )
  {
  bereg1->del_property( lodka );
  bereg1->del_property( chelovek );
  bereg2->add_property( lodka );
  bereg2->add_property( chelovek );

  return true;
            }
    }

    returnfalse;
}

Определяем терминальныесостояния:
//волксъедаеткозу
bool volk_est_kozu(vw)
{
    CVwObject*mesto = vw->
apply_mask (new CVwObjectMask(“Место”
) );

    if( mesto->get_property( “Содержит”,
“Волк” )&&
        mesto->get_property( “Содержит”,
“Коза” )&&
    ! mesto-> et_property( “Содержит”,
“Человек” ) )
    {
        return false;
    }
    return true;
}

//козасъедаеткапусту
bool koza_est_kapustu(vw){...}

И, наконец, успешное терминальное состо-
яние:
bool otvet( vw )
{
CVwObjectMask *bereg_mask = new CVwOb-
jectMask( “Берег” );
bereg_mask->add_property_type( “Имя”,
“Правый” );

CVwObject *bereg = vw->apply_mask(
bereg_mask );

    if( bereg->get_property( “Содержит”,
“Человек” ) &&
        bereg->get_property( “Содержит”,
“Волк” )&&
        bereg->get_property( “Содержит”,
“Коза” )&&
        bereg->get_property( “Содержит”,
“Капуста” )
    )
    {
        exit( true );
    }

    returnfalse;
}
```

7. ЗАКЛЮЧЕНИЕ

Итак, мы обретаем возможность объединения разнообразных алгоритмов в рамках единой системы из многих правил, которая самостоятельно выбирает пути решения задач, алгоритмы которых заранее формально не определены, предоставляя пользователю полный, стандартизованный протокол решений.

Отметим следующий факт. Несмотря на то, что подход к решению задач с помощью представленной модели ИЗ привносит усложнения в программу, тем не менее, если перед системой ставятся разнообразные задачи, в особенности задачи у которых нет предопределенного алгоритма решения (головоломки, доказательства теорем, ЭС и другие типичные задачи ИИ),то программист, воспользовавшийся предложенным подходом, может рассчитывать на универ-

сальность работы с разнообразными входными данными, в частности в условиях их избытка/недостатка. В следующей статье автор предполагает опубликовать объектно-ориентированную реализацию эвристик для соответствующих алгоритмов, решающих более сложные задачи дискретной оптимизации, близкие к описываемым в (Melnikov, Radionov&Gumayunov, 2006).

СПИСОК ЛИТЕРАТУРЫ

1. *Гаврилова Т. А., Хорошевский В. Ф.* (2000). Базы знаний интеллектуальных систем. СПб.: Питер.

2. *Грэхем И.* (2004). Объектно-ориентированные методы. Принципы и практика. М.: Издательский дом «Вильямс».

3. *Джексон П.* (2001). Введение в экспертные системы. М.: Издательский дом «Вильямс».

4. *Melnikov B., Radionov A., Gumayunov V.* (2006). Some special heuristics for discrete optimization problems. – HYPERLINK "http://elibrary.ru/item.asp?id=15291944" Proceedings of 8th International Conference on Enterprise Information Systems, ICEIS 2006: Cyprus Computer Society. Paphos, 2006. P. 360–364.

Рафанович Алексей Александрович – научный сотрудник, Национальный институт здравоохранения, Бетезда, Мериленд, США. Phone: +1 (240) 751-5260 E-mail: araf.pub@gmail.com

Alexey Rafanovich – Stuff scientist, National Institutes of Health, Bethesda, MD, USA. Phone: +1 (240) 751-5260 E-mail: araf.pub@gmail.com