жизненного цикла основных узлов магистральных нефтепроводов позволит обеспечить устойчивость реализации инновационной стратегии развития нефтетранспортирующих сетей.

Литература

1. Байков И.Р., Самородов Е.А., Ахмадуллин К.Р. Методы анализа надежности и эффективности систем добычи и транспорта углеводородного сырья. – М.: ООО «Недра-Бизнесцентр», 2003.
2. Загорулько Ю.А., Загорулько Г.Б., Кравченко А.Ю., Сидорова Е.А. Разработка системы поддержки принятия решений для нефтегазодобывающего предприятия // Труды 12-й национальной конференции по искусственному интеллекту с международным участием – КИИ-2010. – Москва: Физматлит, 2010. –Т.3. –С.137-145.

УДК 004.272.44, 004.4.414, 004.81

АНАЛИЗ ЯЗЫКА С ПОМОЩЬЮ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ОБЪЕКТНО-АТРИБУТНОЙ АРХИТЕКТУРЫ[1]

Салибекян Сергей Михайлович, ст. преподаватель, Московский институт электроники и математики (технический университет), Россия, Москва, salibek@yandex.ru
Панфилов Пётр Борисович, к.т.н., доцент, Московский институт электроники и математики (технический университет), Россия, Москва, panfilov@miem.edu.ru

Распознание текста – проблема в современной вычислительной технике весьма актуальная: компиляция, трансляция, поисковые запросы, смысловое распознание текста, автоматический перевод текста с одного языка на другой и т.д. В настоящее время для подобных целей используется несколько подходов: теория формальных языков (формальные грамматики, конечные автоматы) [1], нейронные сети, теория фреймов [2], концепция «Смысл-текст» [3]. В данной же статье описывается новый способ распознания языка – анализ с помощью объектно-атрибутного подхода (данный подход разработан в Московском институте электроники и математики).

Задача создания нового языка программирования и подходящего метода компиляции для него встала после разработки концепции объектно-атрибутной (ОА) архитектуры вычислительной системы с управлением потоком данных (dataflow) – старые парадигмы программирования для описания работы такой системы были неудобны: классическая императивная парадигма оказалась неприменимой, т.к. она несовместима с парадигмой dataflow; функциональные языки и акторные системы (в основном используемые в парадигме dataflow) имеют весьма скудные возможности абстракции данных и программы, объектно-ориентированная и агентная парадигмы весьма затруднительно реализовывать на распределенных вычислительных системах. В результате, мы пришли к выводу, что и язык программирования и компилятор, используемый для его распознания, также должны быть выполнены по принципам ОА-архитектуры (ОА-архитектура оказалась применимой не только к аппаратной, но и к программной части вычислительной системы) [4]. И, конечно, самым весомым доводом в пользу использования ОА-подхода для создания компилятора стало наше стремление к унификации разрабатываемой вычислительной системы: аппаратура, язык программирования и компилятор, работающие по одним и тем же принципам, делают вычислительную систему целостной и устраняют так называемый семантический разрыв (например, различие между организацией классического языка высокого уровня и машинным языком фон неймановского компьютера требует больших вычислительных затрат на компиляцию и делает скомпилированные программы неоптимальными).

В настоящее время в рамках НИР «Исследование и разработка архитектуры и среды программирования перспективной суперкомпьютерной системы на основе динамической

[1] Статья рекомендована к опубликованию в журнале "**Информационные технологии**"

модели вычислений с управлением потоком данных», выполняемой по федеральной целевой программе «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы» ведется разработка среды программирования и модели суперкомпьютера, функционирующих по правилам ОА-архитектуры. В рамках данного проекта осуществлено создание новой версии компилятора ОА-языка, обладающего следующими качествами: быстрый запуск среды программирования, распределенный режим работы компилятора, б**о**льшее разнообразие конструкций ОА-языка (по сравнению со старой версией), возможность добавления в ОА-язык конструкций языка Си для облегчения написания фрагментов программы, работающих в последовательном стиле.

В процессе выполнения НИР был отработан синтаксис ОА-языка программирования и выработаны основные приемы создания систем распознания языка. Однако ОА-концепция компиляции применима не только для работы с машинными языками высокого уровня (что было сделано в рамках НИР), но и для распознания естественного языка, где требуется высокая степень абстракции данных. Далее необходимо привести небольшой пример реализации ОА-системы для анализа языка.

ОА-вычислительная система представляет собой набор функциональных устройств, обменивающихся информационными парами (ИП), представляющими собой совокупность нагрузки (данные или указатель на другую информационную конструкцию) и ярлыка (уникальный идентификатор нагрузки), т.е. ОА-система работает по принципу dataflow (управление вычислительным процессом с помощью потока данных). ИП могут быть двух видов:

1. Милликоманда – предназначена для управления ФУ-ом;
2. Информационная ИП – служит для описания признака какого-либо объекта.

ИП объединяются в капсулы [4], которые используются либо для описания признака какого-то объекта, либо хранят последовательность милликоманд (миллипрограмма). Форматы милликоманды и информационной ИП совпадают, что позволяет объединять данные и миллипрограмму в одной информационной конструкции.

Для распознания текста в ОА-системе в основном применяются следующие ФУ:

1. ФУ Find (поиск) предназначен для поиска заданных ИП в одной капсуле и запуска миллипрограмм, исходя из результата поиска: «удача» (Success), когда выполняется условие поиска, и «неудача» (Fail), когда условие поиска не выполняется. ФУ реализует следующие милликоманды (в скобках приведена мнемоника милликоманды):

- установить указатель на миллипрограмму, запускаемую при выполнении условия поиска (**SuccessProgSet**);
- установить указатель на миллипрограмму, запускаемую при невыполнении условия поиска (**FailProgSet**);
- установить указатель на капсулу, по которой будет осуществляться поиск (**Set**);
- поиск по капсуле одной информационной пары (**FindIc**), которая пришла в качестве нагрузки к милликоманде.

Получается, что миллипрограмма, на которую указывает SuccessProg, выполняет роль блока программы условного оператора then, а FailProg – блока else, в классической конструкции языка программирования высокого уровня if-then-else.

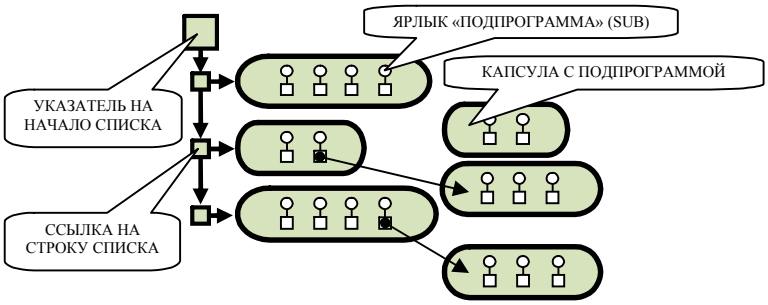

Рис. 1 – Информационная конструкция «Список»

Для реализации же множественного выбора (конструкция switch в языке Си или case в Pascal) предназначено ФУ «Список». В контексте данного ФУ находится ссылка на список капсул, где хранятся данные для осуществления поиска (рис. 1). В ОА-список кроме данных для поиска могут быть добавлены и миллипрограммы обработки данных и осуществления перенаправления информационного потока. ОА-список также может быть использован и в качестве таблицы переменных, где хранятся сведения обо всех мнемониках, что, например, встречаются в распознаваемой программе высокого уровня (переменные, константы, указатели и т.д.).

Источником же данных для ОА-системы распознания языка является ФУ лексического разбора, которое разбивает поступающий к нему текст на лексемы. В состав данного ФУ входит регистр, где хранится милликоманда, которая должна быть прикреплена к генерируемой лексеме. На начальной стадии разбора с помощью специальной милликоманды в регистр заносится милликоманда для первого ФУ, которое будет заниматься разбором лексемы в самом начале процесса компиляции. ФУ лексического разбора строится на основе конечного автомата, т.к. лексический анализ довольно прост и здесь не требуется особых «изысков». На выходе с ФУ лексического разбора язык будет представлен в виде потока милликоманд с ИП, описывающими лексемы: <Mk,<a,M>>, где Mk э N – милликоманда для ФУ, а э N – атрибут (индекс) нагрузки, M – нагрузка, представляющая собой указатель на ячейку памяти, где храниться информационная конструкция, передаваемая в качестве операнда, <…> – обозначение ИП, N – множество натуральных чисел.

В процессе распознания могут быть использованы не только вышеперечисленные типы ФУ, но и другие ФУ, которые могут выполнять различные вычислительные операции, операции ввода-вывода, хранение информации и т.д., что существенно повышает возможности ОА-системы по сравнению с классическим конечным автоматом. Теперь перейдем к описанию принципа функционирования ОА-системы в целом.

Для начала создания ОА-компилятора необходимо составить синтаксическую диаграмму для него. Затем для каждого узла этой диаграммы в ОА-системе создается свое ФУ: для тех вершин, из которых выходит только одна дуга, применяется ФУ «Поиск», для остальных – ФУ «Список». Также создается и ФУ лексического разбора, которое будет являться источником данных в ОА-системе: оно разбивает исходный текст на лексемы и выдает их для дальнейшего анализа на другие ФУ. В состав ФУ лексического анализа входит регистр, где хранится милликоманда, которая «прикрепляется» к генерируемой лексеме.

Для примера возьмем синтаксическую диаграмму простейшего оператора присваивания языка Си: [Var | * Var] = [Const | Var | *VarPointer], где Var – переменная, Const – константа, VarPointer – указатель на переменную, «|» – знак перечисления вариантов языковых конструкций. На рис. 2 представлена диаграмма синтаксического разбора, т.е. на данную стадию разбора приходят не отдельные символы, а информационные пары, обозначающие лексемы (лексический анализ проводит ФУ лексического разбора). К дугам графа приписываются лексемы, по которым ОА-система переходит из одного состояния в другое. Но, в отличие от синтаксической диаграммы распознающего автомата, к дугам диаграммы также приписываются и действия, которые должна выполнять ОА-система, реагируя на приходящую лексему (а не символы, выдаваемые на выходную ленту, как в классической теории автоматов-трансляторов). Такое решение дает программисту намного больше возможностей при создании распознающей системы, чем конечный автомат. Как уже говорилось, каждому узлу диаграммы соответствует свое ФУ – и обозначение (мнемоника) этого ФУ приписано к вершине графа. Основной алгоритм работы ОА-системы распознавания следующий: перед началом осуществляется настройка всех ФУ, и в том числе производиться и настройка ФУ лексического анализа, чтобы оно выдавало первую лексему на ФУ, отвечающее за начало анализа (в нашем случае Root). Далее ФУ, которому от ФУ лексического разбора поступила ИП, анализирует пришедшую лексему и в зависимости от результата анализа перенастраивает лексический анализатор таким образом, чтобы тот

«выбрасывал» очередную лексему для ФУ, ответственного за следующую стадию синтаксического анализа.

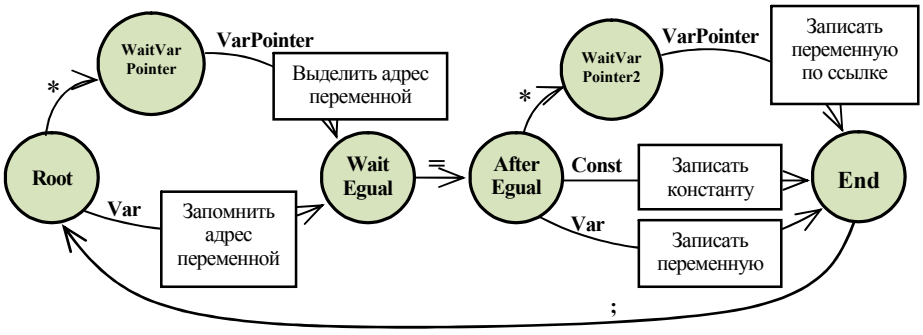

Рис. – 2 Синтаксическая диаграмма разбора оператора присваивания языка Си

Приведем фрагмент ОА-программы (Листинг 1), соответствующий вышеприведенной синтаксической диаграмме (синтаксис ОА-языка описан в [5]): здесь приводится описание инициализации первых трех ФУ ОА-системы.

```
     Root.Set=
     >{Separator="*" Lex.ReceiverMkSet=WaitVarPointer.FindIc}
     >{Var=nil Root.InObjPopMk=VarManager.Set
Lex.ReceiverMkSet=WaitEqual.FindIc}
     >{0=nil Console.Out="Wrong variable describeing" Lex.Stop}

     WaitVarPointer.Set={VarPointer=nil}
     WaitVarPointer.SuccessProgSet=
             {WaitVarPointer.InObjPointPop=VarManager.Read
Lex.ReceiverMkSet=WaitEqual.FindIc}
     WaitVarPointer.FailProgSet={Console.Out="Pointer not finded" Lex.Stop}

     WaitEqual.Set={Separator="="}
     WaitEqual.SuccessProgSet={Lex.ReceiverMkSet=AfterEgual.FindIc}
     WaitEqual.FailProgSet={Console.Out=" '=' not finded"   Lex.Stop}
     ……
     Lex.Lexing="x=10; y=*Uk" \*Запуск генератора лексем*\
```

Листинг 1 – ОА-программа синтаксического разбора

На листинге присутствуют следующие обозначения:

- Root.Set – милликоманда установки списка для ФУ Список (Root);
- ➢ - знак начала новой капсулы, входящей в ОА-список;
- Separator – атрибут символа-разделителя;
- nil – обозначает, что в качестве нагрузки ИП могут выступать любая константа или указатель;
- Var – атрибут переменной;
- FindIc – милликоманда поиска одной ИП в капсуле или списке;
- VarManager – ФУ, ответственное за работу с переменными;
- VarManager.Read – милликоманда чтения переменной из указанного в нагрузке адреса;
- Console.Out – милликоманда вывода сообщения на выводную консоль;
- Lex – обозначение ФУ лексического разбора;
- Lex.ReceiverMkSet – установка милликоманды, прикрепляемой к генерируемой лексеме;
- Lex.Lexing – запуск лексического разбора (в нагрузке символьная строка для разбора).
- Lex.Stop –милликоманда остановки ФУ лексического разбора.

Как видно из листинга 1, при выполнении условия поиска ФУ с помощью милликоманды Lex.ReceiverMkSet перенастраивает ФУ лексического разбора таким образом, чтобы оно выдавало очередную лексему на следующую стадию синтаксического разбора.

Т.е. ФУ работают асинхронно (параллельно), и вычисления активизируются с помощью милликоманд, которыми ФУ обмениваются между собой. Таким образом, ОА-распознающая лексическая система работает по принципу dataflow, который открывает огромные возможности:

1. распараллеливание вычислений;
2. реализация анализа на распределенной и гетерогенной (состоящий из вычислительных узлов различной архитектуры) вычислительной системе.

Напомним, что конечный автомат, активно используемый сейчас для анализа языка, по определению является последовательным, т.к. он в один момент времени должен находится в только одном состоянии q э Q, где Q – множество всех возможных состояний автомата (недетерминированный автомат в плане параллелизма особых преимуществ не дает, т.к. он также работает в последовательном режиме, обрабатывая один символ со входной ленты в один момент времени). Параллелизм в классической системе распознания языка может быть обеспечен лишь благодаря использованию нескольких уровней (проходов) компиляции (например, лексический и синтаксический уровни), где для каждого уровня применяется свой распознающий автомат. ОА же система состоит из совокупности параллельно работающих ФУ. А для более сложного распознания (далее синтаксического уровня языка) конечный автомат неприменим, т.к. он не может работать с глубокой абстракцией данных. ОА же архитектура подходит и для смыслового анализа языка, т.к. обладает следующими преимуществами, связанными с удобством абстракции данных:

1.Высокий уровень абстракции данных и программы [6].

2.Возможность совмещать в одной информационной структуре как данные, так и программу их обработки.

3.Возможность динамического синтеза и перестройки абстрактных данных. Это свойство самое важное для смыслового анализа текста: смыслы (объекты), заложенные в тексте, в подавляющем числе случаев не вписываются в рамки стандартных шаблонов (фреймов). И именно динамический синтез абстракций позволит по заложенным в ОА-систему правилам создавать описания объектов, заранее неизвестных программисту.

Наиболее близкой к предложенному способу смыслового распознания текста является концепция советского лингвиста Игоря Мельчука «Смысл-текст» [3], где синтез смысловых конструкций осуществляется не напрямую, а через несколько этапов (уровней) – от простого к сложному (рис. 3).

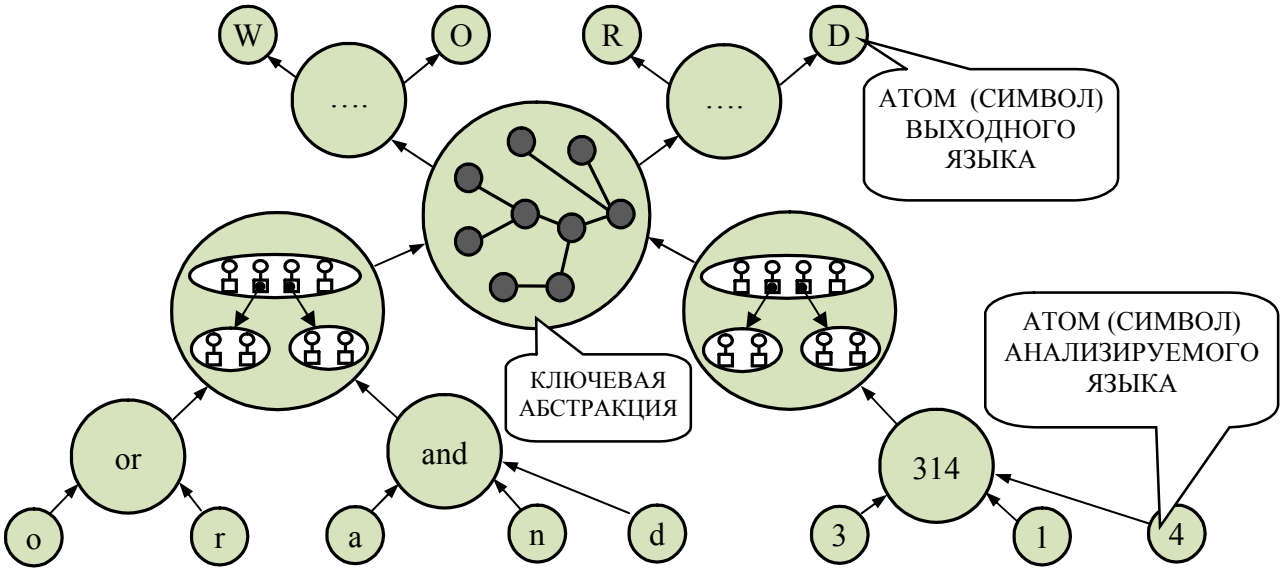

Рис. 3 – ОА-конус абстракций

Мельчук предлагает пять уровней: семантический (уровень смысла), глубинно-морфологический, поверхностно-морфологический, глубинно-синтаксический, поверхностно-синтаксический, фонологический, фонетический (уровень текста) – хотя и

делает оговорку, что число уровней может быть и другим (на усмотрение разработчика модели). Смысл же, заложенный в тексте, Мельчук представляет в виде графа, вершинами которого являются так называемые семы – элементарные (неразложимые) смысловые единицы. В виде графов представляются и отдельные элементы на глубинно-морфологическом и поверхностно-морфологическом уровнях. Однако концепция «Смысл-текст» не получила широкого распространения ввиду своей сложности и, самое главное, негибкости (так, семы в концепции Мельчука представляли собой несколько видов предикатов с фиксированным числом аргументов), что существенно затрудняло синтез абстракций.

В предлагаемой концепции смыслового анализа текста синтез смысловых абстракций также осуществляется от простого к сложному (рис. 3): процесс синтеза начинается со смысловых атомов (элементарных смыслов), в случае анализа текста смысловыми атомами являются символы. Далее осуществляется лексический анализ: выделение из последовательности символов лексем – данной работой в ОА-системе занимается ФУ лексического разбора. Как правило, синтезированные на данном этапе абстракции представляют собой отдельные ИП, в которых атрибутом является идентификатор типа лексемы, а в нагрузке хранится сама лексема (символ-разделитель, число, строка и т.д.). Далее лексемы поступают на этап синтаксического анализа, где происходит анализ взаимосвязи лексем. На этом этапе формируемые абстракции довольно сложны и представляют собой информационные капсулы или ОА-информационные деревья (смысловой граф) [4, 6]. Абстракции эти передаются для дальнейшего анализа на следующий уровень анализа, где ФУ синтезируют из них еще более сложные абстракции. Основная нагрузка по синтезу абстракций на уровнях от синтаксического и выше ложится на ФУ «Поиск» и «Список», которые производят анализ поступивших абстракций и вызов соответствующих миллипрограмм для их обработки. Процесс анализа текста представляет собой так называемый конус абстракций: в его основании располагаются смысловые атомы (символы текста), в вершине – ключевая абстракция, описывающая смысл всего распознаваемого текста. Если же необходимо осуществить перевод с одного языка на другой, то прямой конус абстракций дополняется конусом обратным, который осуществляет синтез текста на другом языке из ключевой абстракции (рис. 3).

ОА-подход, обладающий большой гибкостью при построении смысловых конструкций, позволит в будущем осуществлять не только анализ машинных языков высокого уровня, относящихся к классу контекстно-независимых по иерархии Хомского, но и живых языков, относящихся к контексто-зависимым и свободным. Так, в ОА-модель можно, например, добавлять признаки стилистической окраски текста, ассоциативные связи и т.д; т.е. ОА-ситема будет представлять собой мощную базу знаний для анализа текста.

В заключение следует сказать о том, какая работа была проделана в области распознания текста с применением ОА-архитектуры:

- разработана концепция разбора языка с помощью ОА-системы;
- отработаны основные типы ФУ, используемые для языкового анализа;
- создана среда программирования для написания ОА-программ [6], которая может быть использована в том числе и для анализа текста;
- с помощью ОА-среды программирования создан ОА-компилятор, который осуществляет перевод программы, написанной на специальной ОА-языке в ОА-образ (совокупность информационных капсул и миллипрограмм, описывающих алгоритм решения какой-либо задачи).

Литература

1. Ахо, Ульман Теория Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. – М.: Мир, 1978
2. Минский М. Фреймы для представления знаний: Пер. с англ.- М.: Энергия, 1979.
3. Мельчук И.А. Опыт теории лингвистических моделей «СМЫСЛ <–>ТЕКСТ» – М.: Школа «Языки русской литературы», 1999.

4. Салибекян С.М., Панфилов П.Б. ОА-архитектура – новый подход к созданию объектных систем // Объектные системы – 2011: материалы III Международной научно-практической конференции (Ростов-на-Дону 10-12 мая 2011 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону, 2011. – С. 73-79 URL: http://objectsystems.ru/files/Object_Systems_2011_Proceedings.pdf
5. Салибекян С.М., Панфилов П.Б. Перспективная суперкомпьютерная система на основе объектно-атрибутной модели вычислений с управлением потоком данных / Международная конференция «Развитие суперкомпьютерных и грид-технологий в России» в рамках «Второго Московский Суперкомпьютерного форума» Россия, Москва, ВВЦ 26–27 октября 2011 года URL: http://www.hpc-platform.ru/tiki-download_file.php?fileId=82
6. Салибекян С.М., Панфилов П.Б. Объектно-атрибутный подход к построению интеллектуальных систем // Нейрокомпьютеры: разработки и применение. 2

УДК 004.4

ИЕРАРХИЯ КЛАССОВ МЕТАМОДЕЛИ ОБЪЕКТНОЙ СИСТЕМЫ

Олейник Павел Петрович, к.т.н., Системный архитектор программного обеспечения, ОАО «Астон», Россия, Ростов-на-Дону, xsl@list.ru

В настоящее время наиболее часто используемыми являются объектно-ориентированные языки программирования, популярность которых обусловлена наличием ряда хорошо известных преимуществ [1].

В данной статье подробно рассматривается иерархия классов, используемая для представления метамодели объектной системы, которая была апробирована и успешно применяется в корпоративном решении, ключевой особенностью которого является предоставление возможности добавления новых классов, описывающих сущности предметной области, без перекомпиляции приложения. Данная задача возникла из-за необходимости самостоятельного расширения функционала (путём добавления как новых атрибутов существующих классов, так и добавлением производных классов) опытными пользователями системы. Для решения задачи потребовалось разработать ряд структур, которые бы позволили сохранять информацию об имеющихся классах предметной области и о наличии в них атрибутов. Т.е. необходим механизм представления метамодели объектной системы для создаваемой среды разработки приложений.

Решение любой задачи начинается с формирования критериев оптимальности (КО), которым должна соответствовать полученная реализация. Для рассматриваемой задачи были сформулированы следующие критерии:

1. Разработать единую иерархию, позволяющую представить ключевые элементы объектно-ориентированной парадигмы, такие как классы, атрибуты, наследование (одиночное и множественное), ассоциации и т.п.
2. Предусмотреть возможность предоставления базовых (системных) классов, в которых реализован наиболее общий функционал
3. Разработать иерархию атомарных литеральных типов, которые представляют наиболее распространенные типы данных современных объектно-ориентированных языков программирования.

Рассмотрим требования каждого критерия более подробно. Реализация требования КО1 позволит унифицировано обрабатывать все элементы метамодели (которые представляются в виде экземпляров выделенных классов) с помощью различных структур данных, например с помощью итераторов при формировании цикла. При этом пользователю предоставляется возможность применять хорошо зарекомендовавшие себя объектно-ориентированные подходы, как например, наследование и организация ассоциаций.

КО2 требует наличия системных (встроенных) классов. В общем случае пользователи не могут модифицировать поведение этих классов, но имеют возможность наследоваться от