УДК 004.4’232+004.4’41

В.В. Лаптев, к.т.н., доцент, laptev@ilabsltd.com

Д.А. Грачев, аспирант, dagrachev@list.ru

Астраханский государственный технический университет, Россия, Астрахань

СЕМАНТИЧЕСКАЯ МОДЕЛЬ ПРОГРАММЫ И ЕЕ РЕАЛИЗАЦИЯ

Аннотация

В статье рассматривается статическая семантическая модель программы, разработанная и реализованная в редакторе кода интегрированной среды для обучения программированию. Семантическое дерево позволяет выполнять фазу анализа кода в редакторе и обеспечивает возможность разнообразного внешнего представления программы. Реализация выполнена на языке C# с широким использованием паттернов проектирования, что существенно снизило затраты на разработку.

Семантический редактор кода, учебный язык программирования, семантическая модель программы, статическое семантическое дерево, паттерны проектирования

Введение. В последние годы в вузы на ИТ-направления приходит довольно много абитуриентов, которых не обучали алгоритмизации и программированию в школе. Это весьма неприятное обстоятельство обусловлено тем, что стандарт среднего образования по информатике и ИКТ предусматривает изучение алгоритмизации и программирования только в профильных классах. В вузы же приходят много школьников, которых обучали информатике по стандарту общего среднего образования, в котором на алгоритмизацию отведено менее 10 часов. Поэтому в настоящее время в вузе требуется осуществлять обучение программированию с начального уровня.

При обучении программированию необходимо решить две важных проблемы: на каком языке и в какой среде проводить обучение.

Мы анализировали положение дел в сфере практического обучения программированию в российских вузах и в профильных классах средней школы (выпускники которых обычно поступают на программистские направления). Были выявлены следующие наиболее часто используемые для первоначального

обучения языки: Basic, Pascal, C/C++, C#, Java, школьный алгоритмический язык Ершол. Другие языки (например, Python) используются достаточно редко.

Язык программирования обычно существует в рамках некоторой среды программирования, поэтому были выявлены среды, используемые для начального обучения программированию:

- язык Basic: SmallBasic, Visual Studio 6, Visual Basic 7 Express;
- язык Pascal: Turbo Pascal, Free Pascal Lazarus, PascalABC.NET, Turbo Delphi, BlackBox Component Builder;
- язык C/C++: Visual Studio.NET, Visual C++ Express, Turbo C++, Dev-C++, wxDev-C++, Code::Blocks, CodeLite;
- язык C#: Visual Studio.NET, Visual C# Express, Turbo C#;
- язык Java: NetBeans, Eclipse, IntelliJ IDEA;
- язык Ершол: среда КуМир.

Такое разнообразие говорит о том, что в обучении программированию отсутствует общепринятый доминирующий язык и интегрированная среда – каждый преподаватель самостоятельно принимает решение, какой язык в какой среде использовать. Проблема состоит в том, что все используемые языки и среды (исключениями являются среда КуМир с языком Ершол [1,2,3] и среда PascalABC.NET с языком PascalABC [4]) являются промышленными системами для разработки программного обеспечения и не предназначены для обучения.

Во-первых, все промышленные языки имеют англоязычную лексику. Однако практика обучения показала, что при начальном обучении программированию русская лексика предпочтительнее английской. К.Ю. Поляков, обучавший школьников и в среде КуМир, и в среде Turbo Pascal, отмечает, что русские команды воспринимаются учениками намного легче английских [5]. Ф.В. Ткачев, обучавший школьников в среде BlackBox (школьная сборка – разработка Ф.В. Ткачева, в которой имеется возможность включить русскую лексику ключевых слов языка Component Pascal), тоже говорит о том, что «для начинающих программистов все-таки важно видеть понятные слова на родном языке» [6].

Во-вторых, промышленные языки велики по объему и весьма сложны, в связи с чем их изучение требует весьма значительных усилий. Например, современный стандарт языка C++ – это книга, объемом более 1300 страниц, из которых более 750 страниц занимает описание стандартной библиотеки. Промышленные языки обычно включают множество мелких деталей, относящихся к архитектуре ЭВМ (например, в C++ пять знаковых целых типов и пять беззнаковых, которые отличаются размером и диапазоном). В этом плане язык PascalABC, позиционируемый авторами как обучающий, практически не отличается от промышленных языков.

В-третьих, среды, поддерживающие тот или иной язык программирования, тоже не являются обучающими. Обучающая среда обязана включать набор упражнений и заданий, которые будет выполнять ученик. Из рассмотренных выше сред только PascalABC.NET включает в качестве отдельного компонента электронный задачник Programming Taskbook [7,8], содержащий более 1000 заданий. Заметим, что в настоящее время этот электронный задачник является единственным программным продуктом (кроме нашей системы), позволяющим хоть в какой-то степени автоматизировать работу преподавателя по подготовке вариантов заданий для программирования.

И наконец, в-четвертых, в промышленных средах редактор кода является обычным текстовым редактором. Хотя интегрированная среда обеспечивает некоторую поддержку языка программирования, но основные операции выполняются с символами, строками и блоками текста. Применение текстовых операций к коду программы приводит к регулярному нарушению синтаксиса языковых конструкций, что существенно повышает риск возникновения мелких синтаксических ошибок. Сообщения о синтаксических ошибках часто бывают неинформативными и часто сбивают начинающего с толку.

Редактор интегрированной среды сохраняет код программы в текстовом виде. Это приводит к тому, что код программы можно открыть, просмотреть и изменить вне интегрированной среды. При профессиональной разработке это

бывает полезно, но при обучении провоцирует обучаемых на нечестные способы выполнения заданий по программированию.

Многолетняя практика преподавания программирования одного из авторов статьи показывает, что именно «текстоориентированность» редактора кода приводит к тому, что обучаемый сосредоточен не на решаемой задаче, а на синтаксисе языка программирования. Между тем, исследования мыслительной деятельности профессиональных программистов показали [9], что программист при решении задач использует знания двух типов: семантические и синтаксические.

Семантические знания – это долговременные знания, приобретаемые в результате усвоения опыта решения многочисленных задач с использованием разнообразных средств разработки. Эти знания представляют собой обобщенные базовые понятия программирования и схемы алгоритмов, и не связаны ни с конкретным языком программирования, ни с конкретной системой разработки.

Синтаксические знания – краткосрочные, они приобретаются в процессе изучения конкретного языка программирования в конкретной среде разработки. Эти знания довольно легко забываются при смене средств программирования. Поэтому при начальном обучении важно учить именно программированию (что подчеркивает И.Р. Дединский [10]), а не языку – язык программирования является только инструментом реализации при решении задач.

Из всех рассмотренных языков и сред только язык Ершол в среде КуМир [1] создавался специально для начального обучения программированию. Язык обладает русскоязычной лексикой, и в среде реализован набор исполнителей (Робот, Черепашка и другие), предназначенных именно для начального обучения алгоритмизации. Однако язык устарел и не содержит объектно-ориентированных конструкций.

Таким образом, ввиду практического отсутствия как языков для обучения, так и поддерживающих сред, авторами было принято решение о разработке специального учебного языка программирования и поддерживающей этот язык

интегрированной среды обучения, в которой редактор кода не оперировал символами текста, а был ориентирован на язык программирования.

В работе [11] были сформулированы требования к подобной среде, в работе [12] описан учебный язык программирования Semantic Language и реализация его интерпретатора в рамках интегрированной среды [13,14] для обучения программированию. Интегрированная среда строится вокруг редактора кода, который не является обычным текстовым редактором, а оперирует конструкциями языка программирования и объектами программы. Для реализации подобного подхода авторами разработана семантическая модель программы. В статье описана формальная семантическая модель, внутреннее представление модели в системе и ее реализация на языке C#.

Формальная семантическая модель программы. Программа на учебном языке представляет собой либо единственный модуль, либо несколько модулей, один из которых является стартовым. Поэтому семантическая модель программы – это семантическая модель модуля. Модуль включает список импортируемых модулей, секцию определения и секцию инициализации. Секция инициализации содержит последовательность операторов, исполняемых при загрузке модуля в память. В секции определения задаются константы и переменные, определяемые типы, процедуры и функции. В общем случае все части модуля могут быть пустыми.

Для определения формальной семантической модели программы нужно определить формальную семантическую модель оператора. В общем виде формальная семантическая модель оператора представляет собой пару:

имя = < параметры, правила >

Имя – это уникальный идентификатор оператора. В учебном языке каждый оператор начинается ключевым словом, поэтому имя – это ключевое слово оператора. Параметры и правила определяют семантику оператора. Состав параметров зависит от оператора, набор правил представляет собой множество логических выражений и условий, которые должны быть истинными. Эти

выражения и условия проверяются редактором при вставке оператора в программу.

В качестве примеров рассмотрим несколько формальных моделей простых и блочных операторов учебного языка [1]. Простыми операторами являются оператор импорта, операторы определения констант и переменных, операторы ввода и вывода, оператор присваивания и операторы вызова подпрограммы. К блочным операторам относятся оператор определения модуля, операторы определения подпрограмм, оператор цикла, условный оператор и оператор определения типа. Сначала опишем семантику оператора неформально, а затем приведем формальную модель.

Оператор определения переменной включает тип переменной и ее имя. Если тип переменной является простым (целый, вещественный, символ, булевский), то разрешается инициализировать переменную начальным значением. Инициализатор – это выражение, тип значения которого должен совпадать с типом переменной. Имя переменной не должно совпадать с ранее объявленными именами. Тип переменной должен быть либо определенным в языке, либо определенным в программе. Таким образом, формальную модель оператора объявления переменной можно представить следующим образом:

variable =
< param = < type, name, value >,
  rules = < ~(name ϵ Names), (type ϵ Types),
            if ~(value = null) then
              (type = integer)|(type = real)|(type = boolean)|(type = character)
          >
>

В этой модели определены следующие элементы:

- variable – ключевое слово оператора определения переменной в английской нотации;
- param – параметры формальной модели оператора;

– rules – правила и условия;

Параметрами являются:

– type – тип переменной;
– name – имя переменной;
– value – инициализирующее выражение переменной.

Модель включает два логических выражения:

– ~(name ϵ Names) – это выражение истинно при отсутствии имени определяемой переменной в множестве Names объявленных имен;
– (type ϵ Types) – это выражение истинно, если тип переменной присутствует в множестве Types встроенных или определенных типов программы;

В модели определено одно условие, при истинности которого проверяется истинность вложенных выражений и условий:

if ~(value = null) then

Это условие определяет, какие выражения должны быть истинны, если в определении переменной присутствует инициализирующее выражение. В данном случае выражение

(type = integer) | (type = real) | (type = boolean) | (type = character)

является истинным, если тип переменной является простым.

Блочные операторы отличаются тем, что включают тело – последовательность операторов учебного языка, которую обозначим как *P*:

$P = m_1\ m_2\ \ldots,\ m_k$,

где k – произвольное целое положительное число. Последовательность операторов является обязательным параметром блочного оператора. Однако для разных блочных операторов на состав операторов в теле накладываются семантические ограничения. Например, оператор определения типа разрешается задавать только в секции определений модуля, как и операторы определения процедур и функций. Поэтому в модели блочного оператора требуется для параметра *P* указывать ограничения.

Определим множества S, элементами которых являются модели операторов, которые разрешается задавать в теле того или иного блочного оператора:

- S_{exe} = { constant, variable, let, input, output, call }
- S_{dcl} = { constant, variable, type, procedure, function }
- S_{imp} = { import }

Множество S_{exe} включает операторы, которые разрешается задавать в теле операторов определения процедур, функций, методов, в секции инициализации модуля, в теле оператора цикла и условного оператора. Операторы объявления константы и переменной включены в это множество, поскольку с помощью них во время выполнения определяются локальные в блоке переменные и константы. С помощью множество S_{dcl} определяет состав операторов объявления, которые разрешается задавать в секции определения модуля. И множество S_{imp} определяет возможности конструирования последовательности операторов импорта. Тогда формальную модель оператора определения модуля можно задать таким образом:

module =

< param = < name, P_{imp}, P_{dcl}, P_{exe} >

rules = < ~(name, ModuleNames), ($P_{imp} \in S_{imp}$)&($P_{dcl} \in S_{dcl}$)&($P_{exe} \in S_{exe}$) >

>

Эта модель устанавливает, что параметрами оператора определения модуля является имя модуля и три последовательности операторов: последовательность операторов импорта, последовательность операторов объявления в секции определения, последовательность исполняемых операторов в секции инициализации. Правила истинны, если операторы последовательностей принадлежат соответствующим множествам и имя модуля не должно совпадать с именами системных и других входящих в программу модулей.

Из моделей блочных операторов самой простой является модель оператора цикла:

while =
< param = < exp, P >
  rules = < (type(exp) = boolean), ($P \in S_{exe}$) >
>

Параметрами, очевидно, являются булевское выражение – условие цикла, и тело оператора. Более сложную модель имеет условный оператор, поскольку для условного оператора требуется определить еще модели вложенных ветвей else_if и else:

else_if =
< param = < exp, P >
  rules = < (type(exp) = boolean), ($P \in S_{exe}$) >
>

else =
< param = < P >
  rules = <($P \in S_{exe}$) >
>

Определим множество операторов, которые могут входить в тело условного:

$S_{if} = S_{exe} \cup$ { else_if, else }

Тогда семантическая модель условного оператора определяется так:

if =
< param = < exp, P >
  rules = < (type(exp) = boolean), ($P \in S_{if}$), (last(P) = else) >
>

Третье семантическое правило устанавливает, что ветка else должна быть последней в теле.

Таким образом, семантическая модель программы определяется формальной моделью оператора определения модуля, который включает последовательность операторов импорта, операторов объявления и последовательность исполняемых операторов в секции инициализации. Тела блочных операторов, входящих в эти последовательности, содержат новые последовательности операторов.

Реализация семантической модели программы. Внутренним представлением семантической модели программы с системе является многоуровневый список списков. Каждый оператор программы представлен в узлом списка, в котором сохраняется полная информация о семантике (в частности, каждый узел имеет внутреннее имя, соответствующее оператору). В каждом узле имеются ссылки на следующий и предыдущий. Комментарий в учебном языке Semantic Language тоже является оператором, поэтому представляется отдельным узлом списка. В узлах, соответствующих блочным операторам, имеется ссылка на тело, которое представляет собой дочерний список данного узла. Соответственно, первый оператор дочернего списка имеет ссылку на родительский узел.

Эту структуру строит редактор кода в соответствии с действиями программиста. При загрузке готового проекта строится такая же структура. Мы называем такое представление статическим *семантическим деревом* программы.

Первоначально (в момент включения в проект первого модуля, в котором еще нет ни одного оператора) дерево состоит из одного специального пустого узла, который имеет внутренне имя NullOperator. Этот же узел представляет в дереве пустые строки, которые вставляются в программу для повышения читабельности. Обычно новый оператор вставляется в программу на месте некоторой пустой строки. В дереве эта операция представляет собой простую замену узла NullOperator на конкретный узел-оператор. Если вставляемый оператор является блочным, то на месте тела также вставляется NullOperator.

Например, после вставки первого оператора **модуль**, дерево выглядит как показано на рис. 1 (показаны связи только вперед).

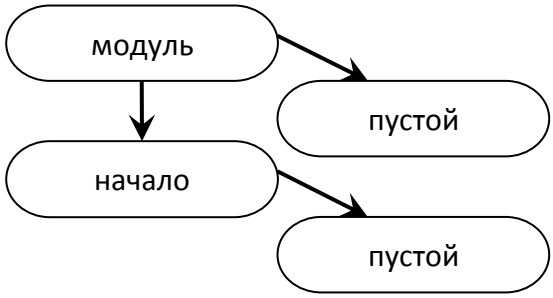

Рис. 1 – Семантическое дерево при вставке первого оператора **модуль**

Дочерняя вершина узла **модуль** – это тело секции определений, а дочерний узел вершины **начало** – это тело секции инициализации. Эти пустые узлы будут заменены на конкретные узлы-операторы при вставке. В качестве примера приведем простую программу, вычисляющую факториалы от 1 до N.

```
#Вычисление факториалов#
модуль Факториал
   подключить Математика;
      закр процедура (входной целое х) Факт
      переменная-целое К;
      переменная-целое тФакт;
      присвоить тФакт := 1;
      присвоить К := 1;
      пока К <= х повторять
         вывести тФакт;
         вывести '\n';
         присвоить К := К + 1;
         присвоить тФакт := тФакт * К;
      конец цикла;
   конец Факт;
начало
   вызвать Факт(15);
```

конец Факториал.

#Конец модуля вычисления факториалов#

В этой простой программе имеются в наличии все основные конструкции, которые могут быть представлены в семантическом дереве: комментарии, пустые строки, простые и блочные операторы. Дерево, которое построил редактор по этой программе, показано на рис. 2 (показаны связи только вперед).

Решение о явном представлении программы в виде семантического дерева привело к важным последствиям при реализации интегрированной среды Semantic IDE [13,14]. Во-первых, регулярная структура семантического дерева делает алгоритм обхода [12] тривиальным. Во-вторых, ошибки выявляются в редакторе, поэтому на вход интерпретатора поступает правильная программа. Интерпретатор при обходе дерева не выполняет ни лексического, ни синтаксического анализа – это существенным образом упростило реализацию, и повысило быстродействие. Вместо интерпретатора можно достаточно легко реализовать конвертер в семантически эквивалентный императивный язык программирования (например, в C или в Component Pascal) – для этого достаточно в алгоритме обхода вызов интерпретирующей функции заменить на вызов функции, формирующей текстовое представление оператора на языке программирования.

В-третьих, в виде семантического дерева может быть представлен любой документ, состоящий только из узлов-комментариев. Система позволяет включать в комментарий текст, рисунки, таблицы – все, что позволяет включать формат RTF. Поэтому справочный и методический обучающий материал можно создавать непосредственно в редакторе кода.

И наконец, явное отделение внутреннего представления программы от внешнего позволило реализовать идею сменяемого синтаксиса во внешнем представлении [13,14].

Фундаментом реализации семантического дерева является паттерн Наблюдатель (Observer) [15]. Как известно, в этом паттерне устанавливаются отношения между субъектом и наблюдателем. Субъект хранит у себя список

наблюдателей, которых оповещает при каких-либо изменениях в своем состоянии.

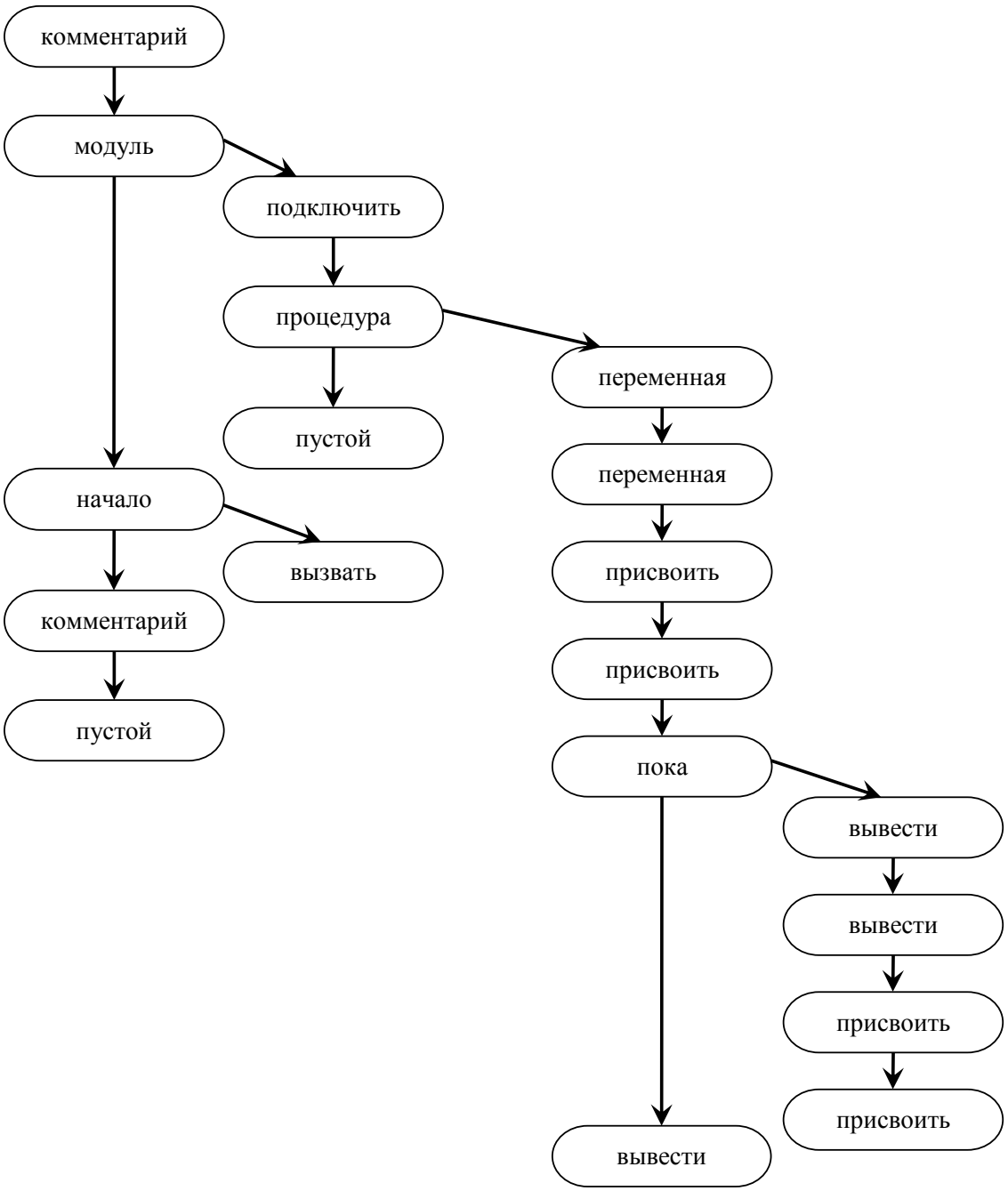

Рис. 2 – Семантическое дерево программы Факториал

В рамках данной реализации субъект наблюдения реализуется абстрактным классом SemanticSubject, который является корнем обширной иерархии классов, часть которой показана на рис. 3. Символом <<A>> помечены абстрактные классы.

В этой иерархии определены все классы-операторы, которые, как и операторы учебного языка, делятся на два вида: однострочные (простые) и многострочные (блочные). Помимо операторов в иерархию семантических объектов включены элементы операторов, которые сами имеют сложную структуру: выражения, список параметров процедур-функций-методов, не элементарные типы. Как правило, эти элементы операторов разрешено редактировать в составе оператора.

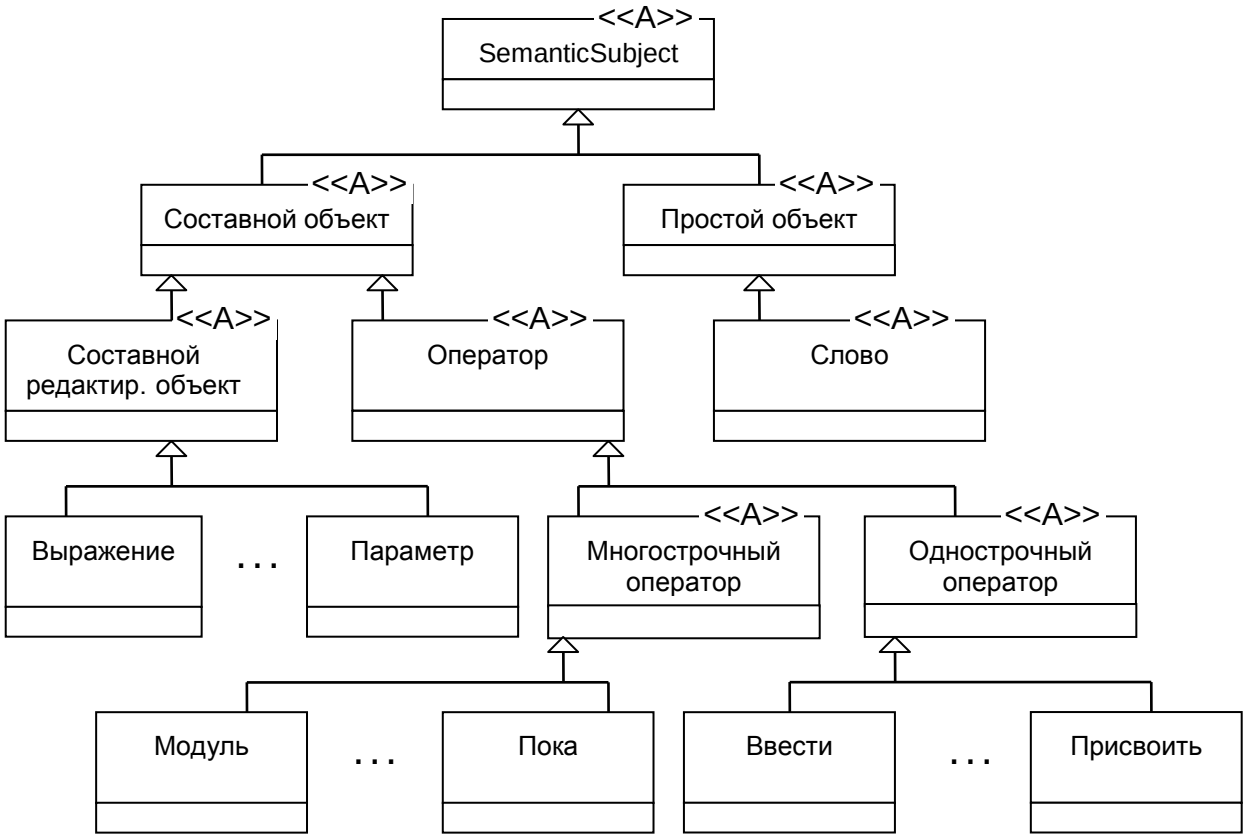

Рис.3 – Иерархия классов субъекта наблюдения

Семантическое дерево, элементами которого являются объекты-операторы, реализуется отдельным классом SemanticTree. В абстрактном классе SemanticOperator (см. на рис. 3 класс Оператор) определены все необходимые поля-ссылки.

Множество классов-наблюдателей, составляющих иерархию классов-блоков с базовым абстрактным классом Block, реализуют интерфейс ISemanticObserver. Все эти классы так или иначе связаны с отображением семантического дерева в окне редактора. Сам редактор, реализуемый классам

Editor, тоже является наблюдателем. Паттерн Наблюдатель используется еще в некоторых случаях, в частности, для наблюдения за курсором.

Помимо паттерна Наблюдатель в реализации широко применяется паттерн Команда [15] (Command). В Semantic IDE определен классический базовый абстрактный класс Command с методами Execute() и Undo(). Посредством паттерна Компоновщик (Composite) реализован классический класс MacroCommand, определяющий составную команду. Такой подход позволил определить наборы базовых команд для различных объектов модели, а уже из них компоновать более сложные составные команды. Например, базовые команды определены для выражения, списка параметров, модуля, семантического дерева, слова и т.п. В системе реализован класс CommandHistory для хранения истории выполненных команд.

Команды IDE составляют отдельную иерархию с базовым абстрактным классом IDECommand, но тоже скомпонованы по «тематическим» группам. Например, все команды для работы с проектом собраны в одну группу. Для хранения команд IDE реализован классический стек команд с методами Undo() и Redo().

Заключение. Описанный подход с широким использованием паттернов позволил существенным образом снизить затраты времени и сил при модификации и развитии системы. Построение семантического дерева программы редактором позволяет полностью отказаться от фазы анализа (как лексического, так и синтаксического) при реализации конвертеров учебного языка в промышленные языки – это также существенным образом упрощает разработку.

Литература

1. Система программирования Кумир – http://www.niisi.ru/kumir/index.htm
2. Кушниренко А.Г., Лебедев Г.В. Программирование для математиков: Учеб. Пособие для вузов – М.: Наука, Гл. ред. физ.-мат. лит., 1988. – 384 с.

3. А.Г. Кушниренко, В.Г. Лебедев. 12 лекций о том, для чего нужен школьный курс информатики и как его преподавать. Методическое пособие. – М.: Лаборатория Базовых Знаний, 2000. – 464 с.
4. Система программирования PascalABC.NET – http://pascalabc.net/
5. Поляков К.Ю. КуМир и школьная информатика. – http://kpolyakov.blogspot.ru/2011/04/blog-post_5678.html
6. Куркина Л.Г., Ткачев Ф.В., Цвелая И.А. Русифицированные мини-исполнители во вводных курсах программирования. – http://www.inr.ac.ru/~info21/texts/bytic-xx-2009.htm
7. Абрамян М.Э. Электронный задачник Programming Taskbook: опыт разработки и применения / II Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование», Москва, 18–21 декабря 2006 г. Сборник докладов научно-практической конференции. — М.: МАКС пресс, 2006. — С. 194–199.
8. Абрамян М.Э. Новые возможности задачника Programming Taskbook: обработка динамических структур и конструктор учебных заданий / IV Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование», Москва, 14–15 декабря 2009 г. Сборник трудов. – М.: ИНТУИТ.РУ, 2009. – с. 282–289.
9. Шнейдерман Б. Психология программирования. Человеческий фактор в вычислительных и информационных системах. – М.: Радио и связь, 1984.
10. Дединский И.Р. Аналитический подход к довузовскому преподаванию программирования. – http://storage.ded32.net.ru/Lib/Doc/AnalyticApproach2010.pdf
11. Лаптев В.В. Требования к современной обучающей среде по программированию // Объектные системы-2010 (Зимняя сессия): материалы II Международной научно-практической конференции. Россия, Ростов-на-

Дону, 10-12 ноября 2010 г. / Под общ. Ред. П.П. Олейника. – Ростов-на-Дону, 2010. – 134 с., с. 104-110.

12. Грачёв А.Д., Лаптев В.В. Разработка учебного языка программирования и интерпретатора для обучающей среды // Объектные системы-2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – 110 с., с. 92-101.
13. Грачёв А.Д., Лаптев В.В. Интегрированная среда для обучения программированию // Объектные системы – 2013: Материалы VII Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2013 г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГТУ (НПИ), 2013. – 118 с., с.17-24
14. Грачёв А.Д., Лаптев В.В. Семантическая интегрированная среда для обучения программированию // Педагогическая информатика. – 2013, №2. – с. 71-81.
15. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2013. – 368 с.

V.V. Laptev, candidate of engineering, associate professor, laptev@ilabsltd.com

D.A. Grachev, post graduate student, dagrachev@list.ru

Astrakhan state technical university, Russia, Astrakhan

SEMANTIC MODEL OF THE PROGRAM AND ITS REALIZATION

Summary

In article the static semantic model of the program developed and realized in the code editor of the integrated environment for training in programming is considered. The semantic tree allows to carry out a phase of the analysis of a code in the editor and provides possibility of various external submission of the program. Realization is executed in the C# language with wide use of design patterns that significantly lowered costs of development.

Semantic editor of a code, educational programming language, semantic model of the program, static semantic tree, design patterns

Literatura

1. Sistema programmirovanija Kumir – http://www.niisi.ru/kumir/index.htm
2. Kushnirenko A.G., Lebedev G.V. Programmirovanie dlja matematikov: Ucheb. Posobie dlja vuzov – M.: Nauka, Gl. red. fiz.-mat. lit., 1988. – 384 s.
3. A.G. Kushnirenko, V.G. Lebedev. 12 lekcij o tom, dlja chego nuzhen shkol'nyj kurs informatiki i kak ego prepodavat'. Metodicheskoe posobie. – M.: Laboratorija Bazovyh Znanij, 2000. – 464 s.
4. Sistema programmirovanija PascalABC.NET – http://pascalabc.net/
5. Poljakov K.Ju. KuMir i shkol'naja informatika. – http://kpolyakov.blogspot.ru/2011/04/blog-post_5678.html
6. Kurkina L.G., Tkachev F.V., Cvelaja I.A. Rusificirovannye mini-ispolniteli vo vvodnyh kursah programmirovanija. – http://www.inr.ac.ru/~info21/texts/bytic-xx-2009.htm
7. Abramjan M.Je. Jelektronnyj zadachnik Programming Taskbook: opyt razrabotki i primenenija / II Mezhdunarodnaja nauchno-prakticheskaja konferencija «Sovremennye informacionnye tehnologii i IT-obrazovanie», Moskva, 18–21 dekabrja 2006 g. Sbornik dokladov nauchno-prakticheskoj konferencii. — M.: MAKS press, 2006. — S. 194–199.

8. Abramjan M.Je. Novye vozmozhnosti zadachnika Programming Taskbook: obrabotka dinamicheskih struktur i konstruktor uchebnyh zadanij / IV Mezhdunarodnaja nauchno-prakticheskaja konferencija «Sovremennye informacionnye tehnologii i IT-obrazovanie», Moskva, 14–15 dekabrja 2009 g. Sbornik trudov. – M.: INTUIT.RU, 2009. – c. 282–289.
9. Shnejderman B. Psihologija programmirovanija. Chelovecheskij faktor v vychislitel'nyh i informacionnyh sistemah. – M.: Radio i svjaz', 1984.
10. Dedinskij I.R. Analiticheskij podhod k dovuzovskomu prepodavaniju programmirovanija. – http://storage.ded32.net.ru/Lib/Doc/AnalyticApproach2010.pdf
11. Laptev V.V. Trebovanija k sovremennoj obuchajushhej srede po programmirovaniju // Ob#ektnye sistemy-2010 (Zimnjaja sessija): materialy II Mezhdunarodnoj nauchno-prakticheskoj konferencii. Rossija, Rostov-na-Donu, 10-12 nojabrja 2010 g. / Pod obshh. Red. P.P. Olejnika. – Rostov-na-Donu, 2010. – 134 s., s. 104-110.
12. Grachyov A.D., Laptev V.V. Razrabotka uchebnogo jazyka programmirovanija i interpretatora dlja obuchajushhej sredy // Ob#ektnye sistemy-2012: materialy VI Mezhdunarodnoj nauchno-prakticheskoj konferencii (Rostov-na-Donu, 10-12 maja 2012g.) / Pod obshh. red. P.P. Olejnika. – Rostov-na-Donu: ShI JuRGTU (NPI), 2012. – 110 s., s. 92-101.
13. Grachyov A.D., Laptev V.V. Integrirovannaja sreda dlja obuchenija programmirovaniju // Ob#ektnye sistemy – 2013: Materialy VII Mezhdunarodnoj nauchno-prakticheskoj konferencii (Rostov-na-Donu, 10-12 maja 2013 g.) / Pod obshh. red. P.P. Olejnika. – Rostov-na-Donu: ShI (f) JuRGTU (NPI), 2013. – 118 c., s.17-24.
14. Grachyov A.D., Laptev V.V. Semanticheskaja integrirovannaja sreda dlja obuchenija programmirovaniju // Pedagogicheskaja informatika. – 2013, №2. – s. 71-81.

15. Gamma, Je. Priemy ob#ektno-orientirovannogo proektirovanija. Patterny proektirovanija / Je. Gamma, R. Helm, R. Dzhonson, Dzh. Vlissides. – SPb.: Piter, 2013. – 368 s.