УДК 519.682.1

СИНТАКСИЧЕСКИ РАСШИРЯЕМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ КАК СРЕДСТВО РЕАЛИЗАЦИИ ЯЗЫКОВ ПРЕДМЕТНОЙ ОБЛАСТИ

Д. И. Соломатин

Воронежский государственный университет

Использование языков предметной области — устоявшаяся практика в разработке программного обеспечения. С появлением специальных инструментов, предназначенных для конструирования таких языков, можно говорить о возникновении нового подхода в программировании — языково-ориентированного программирования. Основная идея такого подхода, заключающегося в создании под каждую конкретную задачу своего языка вместо использования универсальных средств. В статье предлагается использовать в качестве таких инструментов расширяемые языки программирования, а также анализируются способы построения расширяемых языков на базе существующих.

ВВЕДЕНИЕ

Если проследить историю развития индустрии программирования, можно обнаружить интересные закономерности в плане предпочтения различных языков. Сложность программирования в машинных кодах и языках ассемблера быстро привела к созданию высокоуровневых языков программирования (Fortran, Algol, Cobol, C, Pascal и т.д.). Затем индустрия подхватила идеи объектно-ориентированной технологии (Smalltalk, C++, Java, C#). Главным качеством любого следующего уровня было возрастание уровня абстракции инструмента, предоставляемого разработчику, и уже из этого вытекает удобство использования, скорость разработки и т.д.

Стоит отметить, что наряду с описанными технологиями развивались и другие парадигмы программирования (логическое, функциональное), которые оказывались востребованными в ограниченном круге задач, а также энтузиастами, но мейнстримом их назвать никак нельзя, несмотря на то что они во многом опередили традиционные языки программирования. Динамические (скриптовые) языки также в целом следовали общему пути развития (от процедурного подхода к объектно-ориентированному), но оказывались более восприимчивыми к различным новым идеям (мультипарадигменные языки и т.д.).

Возрастание уровня абстракции предполагает переход от машинного представления программы (как делать) к представлению, которое ближе человеческому восприятию и в конечном итоге к моделированию предметной области. Это касается не только самих языков программирования, но и сопутствующих инструментов (например, UML), а также технологий разработки ПО (программная инженерия).

ОБЗОР СПОСОБОВ ПОСТРОЕНИЯ РАСШИРЯЕМЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ НА БАЗЕ СУЩЕСТВУЮЩИХ

В традиционных языках возможности абстрагирования достаточно жестко ограничены средствами, заложенными в каждом из них. В этом плане интересно отметить противоречия в развитии языков. Идеи включить в язык как можно больше возможностей породили сложность в изучении и понимании языков программирования, на фоне которой наличие этих возможностей стало спорным преимуществом (речь идет прежде всего о C++). Разработчики Java сильно упростили новый язык по сравнению с C++ (убрали препроцессор, шаблоны, перегрузку операций, приведение типов и т.д.) и при этом язык Java достаточно быстро стал популярным (естественно, совсем не из-за относительной простоты, но ограниченность в возможностях не стала препятствием к широкому распространению). Однако же оказалось, что все же средств языка недостаточно, что привело к его расширению (параметризация классов — аналог шаблонов и т.п.). Разработчики конкурирующего C# добавили в свой язык еще больше возможностей (ограниченный препро-

© Соломатин Д. И., 2008

цессор, ограниченные перегрузки операторов), но меньше, чем в C++. Из всего этого можно сделать вывод, что существует какое-то оптимальное для индустрии соотношение между возможностями языка и его сложностью. Можно также предположить, что это соотношение не постоянно, а меняется с течением времени (развитием наших представлений).

Встроенных в язык средств оказывается часто недостаточно для удобного решения какой-либо задачи и, если задача регулярная, изобретаются различные способы ее эффективного решения за пределами возможностей используемого инструмента, т.е. по сути производится расширение инструмента. И часто это расширение представляет собой то, что называют языком предметной области (Domain Specific Language — DSL), т.е. некий новый инструмент для решения повторяющихся задач, «неудобных» для базового инструмента. Многие авторы разделяют идеи, что методология разработки ПО, направленная на использования языков предметной области и, соответственно, их разработку соответственно задаче, — следующий шаг в индустрии разработки ПО и новая парадигма в программировании — языково-ориентированное программирование [4, 6, 7, 8, 9].

Типичный язык предметной области — это простой, как правило, декларативный язык для решения какой-то узкой задачи, для которой использование языка программирования общего назначения (General Purpose Language — GPL) затруднительно или неэффективно. Часто такие языки являются просто средством записи формализмов, принятых в данной области. Многие широко используемые в разработке ПО средства, если смотреть на них с этой точки зрения, представляют собой типичные DSL.

Пожалуй, самый яркий пример — язык запросов к реляционным базам данных. Концепции языка основаны на теории реляционных отношений, т.е. определенного структурного описания данных. В терминах SQL пользователь думает в категории «какие данные получить (изменить)», а не «как», т. е. уровень абстракции по сравнению с языком программирования общего назначения выше, что определяет эффективность применения данного DSL. Стоит отметить, что в рамках SQL пользователь часто даже не задумывается, как и где на самом деле хранятся данные и как они будут извлекаться, а работает строго в рамках предложенной абстракции и ее формального описания.

Следующий набор широко используемых DSL — языки командных процессоров так называемых «шеллах», в различных операционных системах. Примеры интерпретаторов таких языков — Unix-программы sh, csh, bash (соответствующие языки носят одноименные названия) и другие, в операционной системе Windows — командный процессор cmd.exe (ранее command.com). Данные языки используются для написания скриптов — последовательностей команд для автоматизации ручного ввода. Т.к. команды направлены на обработку данных, то такие языки поддерживают встроенные средства для этого: работа с файлами, перенаправление потоков ввода/вывода, работа с аргументами командной строки и окружением и т.д. Базовыми конструкциями таких скриптов служат вызовы других консольных программ операционной системы, каждая из которых выполняет какую-то свою специфическую функцию (например, поиск подстрок — утилита grep). Стоит отметить, что каждая такая программа по сути сама является интерпретатором специфического DSL, инструкции которого передаются через ключи командной строки, через стандартный ввод или через файл. Многие такие DSL, включая языки командных процессоров получили условное название «малых языков» Unix. По сравнению с рассмотренным SQL, языки командных процессоров гораздо ближе к языкам общего назначения, поддерживают базовые императивные конструкции, переменные, но, являются гораздо более простыми и имеют четкую область применения, где показали свою эффективность, что определило их популярность (широко используются, например, в системном администрировании).

Здесь интересно отметить, что некоторые языки общего назначения выросли из языков предметной области [10].

Еще один пример, где языки предметной области используются повсеместно — языки представления, к которым прежде всего относится HTML, а также языки, созданные на основе смешения HTML с языками программирования общего назначения (ASP, JSP, PHP и т.д.) или же языками, напоминающими языки командных процессоров (SSI). Часть рассматриваемых языков является наглядной демонстрацией одного часто встречающегося свойства

DSL — прозрачной интеграции с базовым языком приложения.

Специальные языки разметки используются также в различных системах компьютерной верстки, наиболее известной из которых, вероятно, является — TeX, а также его расширение — LaTeX. Возможностей текстового описания оказывается достаточным не только для верстки текста, но и для отображения сложных математических формул, этим объясняется высокая популярность TeX как редактора научных текстов. Здесь интересно обратить внимание на тот факт, что удачные языки предметных областей часто интегрируются между собой, так например, в язык разметки, используемый в популярном проекте Wiki в качестве тега встраивается язык описания математических формул в нотации TeX.

DSL широко используются при построении пользовательских интерфейсов в традиционных системах программирования. Однако здесь используется графическая нотация и вследствие этого такие среды получили название визуальных сред программирования. С одной стороны это не языки программирования, а с другой для хранения созданного интерфейса часто используется текстовое представление в виде специального языка (например, DFM в среде Delphi) или же в XML (XAML в Windows Vista, GlageXML в среде GNOME).

Стоит особо остановиться на XML. XML (*eXtensible Markup Language*) — расширяемый язык разметки, предназначенный для описания различных структурированных данных; рекомендован консорциумом W3C. XML базируется, подобно HTML, на тегах. Расширяемым он назван из-за возможности добавления новых тегов, которые несут (в отличие от HTML) семантическую нагрузку. На основе XML создано множество языков различных предметных областей. По существу даже простой конфигурационный файл, записанный в XML можно считать специфическим языком программирования для конфигурирования данного конкретного приложения.

Ниже перечислены еще ряд областей применения, где активно используются DSL:

— конфигурационные файлы многих приложений представляют собой серьезные декларативные языки программирования (SendMail, CommuniGate, Squide, Apache и огромное множество других);

— системы сборки приложений (Make, Ant, Maven и т.д.);

— входные языки (грамматики) генераторов парсеров (Flex, Yacc, Bison, JavaCC, ANTLR и т.д.);

— различные языки работы с XML (XML Schema, XPath, XQuery, XSLT и т.д.);

— математические языки (MathCad, Matlab и т.д.);

— языки имитационного моделирования (GPSS и т.д.);

— различные диалоговые языки;

— языки САПР;

— и т.д.

Как показано выше, использование DSL — довольно старая и распространенная практика, возникает вопрос, почему такой подход называют новой парадигмой программирования [2, 4, 6, 8]. Принципиальная идея здесь, прежде всего, в использовании соответствующего *языкового инструментария* [6], т.е. удобных средств для создания языков предметной области и работы с ними. Образно говоря, идея состоит в том, чтобы создание таких языков вынести из области системного в раздел прикладного программирования. В дальнейшем такие языки могли бы создаваться как компоненты в объектно-ориентированном программировании и повторно использоваться другими разработчиками.

В качестве альтернативных языковому инструментарию средств можно рассматривать универсальное использование языков на основе XML, но такой подход на практике имеет довольно жесткие ограничения. XML разрабатывался как компромисс, удобный для обработки как компьютером (структурированность), так и человеком (текстовый формат). Легко заметить, что синтаксис любого языка программирования можно очень просто представить в XML, однако этого не происходит, т.к. при этом работа программиста с таким представлением программ будет сильно затруднена, и даже не из-за большего объема кода, а именно из-за сложности восприятия такого синтаксиса. Таким образом, можно сделать вывод, что языки предметной области на основе XML могут быть применимы только в ограниченных простых случаях, например, разного рода конфигурационных файлах.

В работе [6] предлагается классифицировать DSL на *внешние* и *внутренние*. Естественно, такая классификация достаточно условна.

Внешние DSL — те, которые написаны на отличном от основного приложения языке («малые языки», конфигурационные файлы XML и т. д.) Основное преимущество внешних DSL в том, что разработчик при проектировании таких языков ничем не ограничен, он может задать какой угодно синтаксис и семантику для описания понятий предметной области, нет никаких технических ограничений основного приложения. Но при этом придется разрабатывать транслятор такого языка с нуля. Другой недостаток — «символьный барьер», т.е. невозможность обратиться по имени из такого языка к объектам основного приложения, что порождает значительные накладные расходы в случае необходимости тесной интеграции основного приложения и DSL.

Внутренние DSL описываются и используются в коде основного приложения. То есть для использования внутренних DSL необходима возможность расширения синтаксиса и семантики основного языка приложения за счет самого языка или за счет инструментария среды разработки. Такие языковые средства присутствуют в некоторых функциональных языках с «минималистическим» синтаксисом, например Lisp, и, как правило, обеспечиваются замыканиями (closures) и развитой системой макросов. Таких возможностей нет в «языках фигурных скобок», которые широко используются в индустрии. Для преодоления этого ограничения предлагается использовать упомянутый выше языковой инструментарий.

Внешние DSL могут быть использованы как на этапе разработки, так и на этапе эксплуатации приложения, например, для добавления новой функциональности конечными пользователями. При этом часть кода приложения может быть написана разработчиком на внешнем DSL. Внутренние же DSL используются только на этапе разработки приложения в качестве еще одного инструмента разработчика.

Основные идеи языково-ориентированного программирования направлены на внутренние DSL. Ведущие поставщики средств разработки предлагают свои решения в данной области. Наиболее перспективны в этом плане продукты Microsoft DSL Tools [11] и JetBrains MPS [12]. Общность обоих продуктов заключается в отказе от обычного текстового представления программ в пользу создания специальных графических/псевдографических редакторов для DSL.

Microsoft DSL Tools состоит из мастера создания проекта, графического средства для описания модели предметной области, описания дизайнера нового языка в XML, набора генераторов кода и специальной среды для создания генераторов текста. В мастере создания проекта доступны следующие шаблоны решений: диаграммы последовательностей, классов, компонентов, а также минимальное описание нового языка. В результате работы мастера создаются два проекта: первый определяет сам язык и средства редактирования и обработки для этого языка; второй — как создаваемые языковые средства интегрируются с Visual Studio. Из XML-описания дизайнера создаваемого DSL генерируется код его реализации. Генераторы кода по модели предметной области и описанию дизайнера для этой области строят соответствующий код. Генератор текста по конкретной модели (DSL-программе) на этапе работы с созданным языковым расширением генерирует тексты (код на базовом языке программирования, документацию и т. п.), соответствующие данной конкретной модели. При разработке нового DSL можно добавить механизмы проверки корректности модели, которые будут применятся на этапе компиляции к DSL-программам, т.е. элементы статической верификации.

В MPS от JetBrains для создания нового DSL используется несколько языков [7]. Интересно заметить, что эти языки сами являются DSL для создания новых DSL. Первый из них — *язык структуры*, используется для задания абстрактного синтаксиса программы, т.е. типов элементов программы (типов узлов графа программы), их свойств и видов отношений между собой (мета-уровень). *Язык редактора* описывает, как программа на DSL будет выглядеть в текстовом представлении, т.е. задает конкретный синтаксис, но не только. Язык редактора построен в виде вложенных блоков, в которых редактироваться могут только некоторые блоки-ячейки, причем в ячейках может быть не только текст, но и любой элемент управления, удобный для редактирования ячеек данного типа, например, элемент управления для выбора цвета и т.п. Следующий язык, *язык трансформации*, предназначен для описания механизма перевода программы из исходного языка DSL в целевой, т.е., по сути, для компиляции DSL. Компиляция проходит в два этапа: трансформация исходно-

го языка в промежуточный целевой и трансляция промежуточного языка в финальный результат, который может быть различным, но, как правило, представляет текст на стандартном языке Java без расширений. Второй этап тривиален — прямое отображение, а первый этап осложняется тем, что исходный язык и промежуточный могут кардинально отличаться. Для облегчения выполнения таких преобразований предусмотрено еще несколько DSL: *язык запросов модели*, *язык образцов* и *язык шаблонов*.

Рассмотренные продукты представляют собой средства создания DSL для двух конкурирующих платформ программирования — .NET и Java. Появление этих продуктов от ведущих разработчиков является еще одним подтверждением востребованности языкового инструментария для разработчиков не только в академических целях, но и в индустрии программирования.

Альтернативой подхода, используемого в Microsoft DSL Tools и JetBrains MPS, является добавление средств расширения непосредственно в базовый язык разработки приложений. И у такого подхода есть одно неоспоримое преимущество: расширение языка происходит в рамках самого языка, а не за счет среды программирования, что избавляет разработчика от привязки к IDE и имеет ряд других преимуществ, прежде всего, плоское текстовое представление всех частей программы. Попытки создания графических языков предпринимались неоднократно, однако широкого применения не нашли, за исключением узких областей. Это объясняется удобством текстового представления программ по сравнению с графическим, поэтому логично думать, что и для большинства DSL удобным окажется именно текстовое представление (как самих программ, так и метапрограмм, т.е. определения DSL).

Язык с расширяемым синтаксисом и семантикой (далее — *расширяемый язык*) может быть построен как совершенно новый язык программирования (например, Nemerle [14, 15]), так и как специфическое расширение существующего языка программирования. Причем последний вариант с позиции практического использования на данный момент более актуален, т.к. может дать значительные преимущества от применения языков предметной области без какого-либо существенного изменения в процессе разработки ПО, в средствах разработки и т.д.

Большинство функциональных языков являются расширяемыми в вышеназванном смысле, однако на практике используются редко, а принципы их расширения проблематично применить к императивным языкам. Поэтому задача расширения традиционных языков является актуальной. Ниже описываются возможные принципы реализации таких расширений.

Любой язык в текстовом представлении имеет как конкретный синтаксис, так и абстрактный. *Конкретный синтаксис* языка — это определенный набор правил, определяющих допустимые цепочки входных символов. Под *абстрактным синтаксисом* языка понимаются правила, по которым могут компоноваться логические структуры языка. Абстрактный синтаксис определяет, например, что условный оператор состоит из выражения проверки условия, а также зависимых then- и else-выражений, причем else-выражение может быть вырожденным. Как конкретно будет задаваться выражение, в стиле языка C, языка Pascal или же с помощью редактора блок-схем, для абстрактного синтаксиса языка значения не имеет. То есть абстрактный синтаксис описывает допустимые операции языка программирования. Таким образом для одного абстрактного синтаксиса языка может быть задано множество различных конкретных синтаксисов.

Принципиальное отличие абстрактного и конкретного синтаксисов в том, что абстрактный синтаксис представляет собой граф из абстрактных понятий языка (нетерминалов) и их связей, а конкретный синтаксис описывает плоскую структуру корректных выражений языка, необходимую для записи программ в плоском текстовом преставлении. Когда программист работает над программой, он мыслит категориями абстрактного синтаксиса программы, но запись вынужден вести в рамках какого-то конкретного синтаксиса.

Формальные методы обработки программ также, как правило, работают с представлением программы в виде структуры, соответствующей абстрактному синтаксису языка. Такая структура получила название *абстрактного синтаксического дерева* (AST — Abstract Syntax Tree). На самом деле данная структура часто не является деревом, а представляет собой граф, но во многих случаях рассматривать его удобно в виде дерева, отбросив часть связей. Перевод программы из текстового представления в AST называ-

ют *синтаксическим отображением* [1]. Обычно схема обработки программы выглядит следующим образом:

На рис. 1 шаг 1 — синтаксическое отображение. На шаге 2, в зависимости от задачи, может происходить как преобразование (перевод) AST-дерева в конечное представление, например, в бинарный код для целевой платформы при компиляции, так и выполнение кода на вычислителе, который понимает программы с абстрактным синтаксисом языка L (интерпретация).

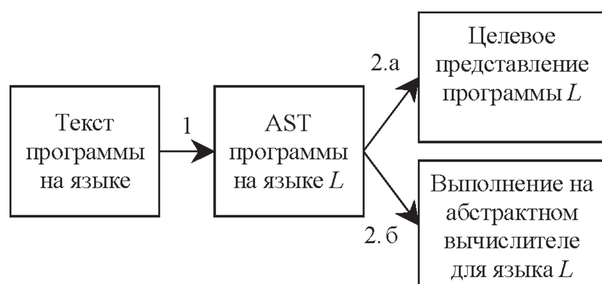

Рис. 1. Типичная схема обработки текста программы

Шаги 2.а и 2.б могут совмещаться в том смысле, что сначала из AST-дерева строится еще одно представление программы, более удобное для интерпретации (байт-код), а потом этот байт-код выполняется на виртуальной машине. Однако в этом случае выполнение происходит уже не программы на языке L, а байт-кода, который, как правило, имеет плоскую (более плоскую) структуру, так же как и текст программы.

Семантику расширений языка программирования логично задавать средствами самого языка, посредством описания существующими конструкциями языка с заданной семантикой. Такое описание, по сути, представляет собой трансляцию нового языка $L+E$, где E — расширение, в базовый язык L. Обозначим новый язык $L+E$ через L^{+} или $L^{+(1)}$:

$$L \xrightarrow{E} L^{+(1)}. \quad (1)$$

В дальнейшем в язык может быть добавлено расширение $E_{(2)}$, которое будет использовать для задания семантики уже язык $L^{+(1)}$, обозначим вновь полученный язык за $L^{+(2)}$ и т.д.:

$$L^{+(n-1)} \xrightarrow{E_{(n)}} L^{+(n)} \quad (2)$$

или

$$L \xrightarrow{E_{(1,n)}} L^{+(n)}. \quad (3)$$

Так как необходимо задавать семантику расширений языка, которая определяется именно для элементов абстрактного синтаксиса языка, то трансляцию расширений языка в конструкции базового синтаксиса языка необходимо проводить также над абстрактным синтаксисом. В соответствии с этим положением, общая схема обработки программы на расширяемом языке представлена на рис. 2.

Как можно видеть, при обработке расширяемых программ добавляется еще один шаг, на рис. 2 — шаг 2. На этом шаге выполняется преобразование AST-дерева программы расширенного синтаксиса в AST-дерево базового синтаксиса; правила такого преобразования задаются в выражении (3). Будем называть такое преобразование *сужающим синтаксическим отображением*. Шаг 2 в общем случае выполняется итеративно, каждая итерация соответствует одному расширению языка.

По сути здесь речь идет уже о *метапрограммировании*. Под метапрограммированием подразумевается создание программ, которые создают другие программы в широком смысле. Применительно к предложенной технике расширения языка программирования, первые программы — это расширения (точнее код, который выполняет шаг 2), вторые — конструкции базового языка, в которые эти расширения транслируются.

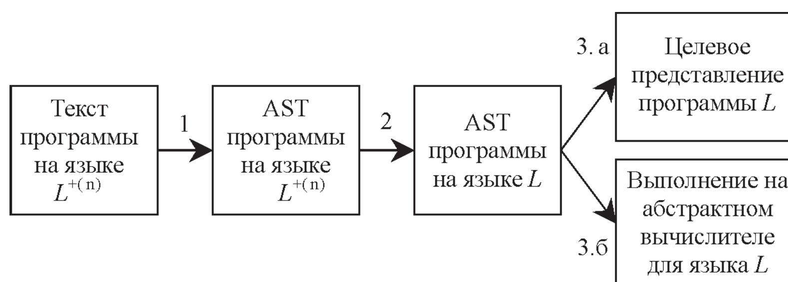

Рис. 2. Типичная схема обработки текста программы на расширяемом языке

Интересно отметить, что технологией сужающего синтаксического отображения пользуются компиляторы, когда синтаксические конструкции языка программирования не могут быть простым образом отображены на целевую платформу. P. Smacchia в своей работе [13] рассматривает пример применения такой техники для «захвата» локальных переменных при компиляции анонимных делегатов в языке C#.

Инструкции выполнения шага 2 (рис. 2) могут быть описаны как внешними средствами, например, внешними библиотеками (пусть даже написанными на том же самом языке программирования), так и средствами самого языка. В первом случае расширяемость — свойство, условно говоря, системы программирования. Во втором — свойство именно самого языка и такие языки программирования будем называть *внутренне расширяемыми* или *саморасширяемыми*. Необходимо отметить, что такое деление достаточно условно, т.к. речь идет прежде всего о тесноте интеграции самого языка и системы/подсистемы его расширения.

Существует класс расширений языка, заслуживающий отдельного внимания. Речь идет о расширениях, сужающее синтаксическое отображение для которых может быть выполнено локальным преобразованием AST-поддеревьев добавленных конструкций без каких-либо изменений во всем остальном дереве программы. Такие расширения языка будем называть *локальными расширениями*.

Типичный пример такого расширения можно представить на следующем пример. Предположим, что в языке L есть цикл WHILE со следующим абстрактным синтаксисом:

└ <цикл_WHILE>
    └ <A : условие_цикла>
    └ <B : тело_цикла>

Тогда в языке $L^{+(1)}$ семантика расширения, цикла FOR (в стиле языка C), с абстрактным синтаксисом:

└ <цикл_FOR>
    └ <A : инициализация_цикла>
    └ <B : условие_цикла>
    └ <C : действие_в_конце_каждой_итерации>
└ <D : тело_цикла>

может быть выражена посредством уже существующего в языке L цикла WHILE и оператора последовательности операций:

└ <последовательность_операций>
    └ <A>
        └ <цикл_WHILE>
        └ <B>
        └ <последовательность_операций>
            └ <D>
            └ <C>

Отметим интересное, свойство локальных расширений языка — они легко могут быть выражены не только в абстрактном, но и в конкретном синтаксисе языка. Ниже показано, как рассмотренный пример с циклом WHILE может быть перезаписан в рамках конкретного синтаксиса языка с C-подобным синтаксисом.

Если цикл WHILE записывается:

«while» «(» A «)» B

а цикл FOR в виде

«for» «(» A* «;» B* «;» C* «)» D*

то последний может быть перезаписан в следующем виде:

«{» A* «;» «while» «(» B* «)» «{» D* «;» C* «}» «}»

Заметим, сужающее синтаксическое отображение в случае, если допустимы только локальные расширения, может быть явно не выделено в отдельный шаг, особенно для интерпретируемых языков (для которых выполняется шаг 3.б, см. рис. 2).

Как будет показано далее, класс локальных расширений языков программирования довольно широк с точки зрения практического использования, а неоспоримым их преимуществом является простота обработки, что видно из рассмотренных примеров.

Расширение языка задается, как правило, как на уровне конкретного синтаксиса, так и абстрактного. Но возможны ситуации, когда расширение задано лишь для конкретного синтаксиса. Например, можно добавить в конкретный синтаксис языка синоним какой-либо синтаксической конструкции, следовательно, в абстрактном синтаксисе языка обе эти конструкции будут представлены одной структурой. Но формально можно также добавить новую структуру и в абстрактный синтаксис языка, а уже на шаге сужающего синтаксического отображения заменять эту новую структуру на базовую.

С другой стороны, расширение языка может менять только семантику некоторых существующих конструкций языка без изменения его синтаксиса, как абстрактного, так и конкретного. Такая ситуация характерна, например, для

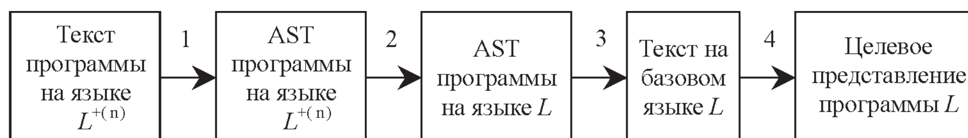

Рис. 3. Схема обработки текста программы на расширяемом языке при использовании существующего компилятора языка L

аспектно-ориентированного программирования. Техническая реализация таких расширений может быть выполнена аналогично описанному подходу с изменением AST-дерева программы при добавлении новых конструкций языка.

Рассмотрим ситуацию, когда расширения добавляются к какому-то существующему компилируемому языку. Здесь возможны два варианта технической реализации. Первый вариант — разработать заново компилятор для расширяемого языка. Второй — написать препроцессор, который будет переводить текст программы на языке с расширениями в семантически эквивалентный текст программы на базовом языке без расширений, который затем компилируется существующим компилятором (или интерпретируется существующим интерпретатором). Очевидно, что второй вариант технически гораздо проще в реализации, а конечный результат, целевое представление программы, будет одинаковым, поэтому именно этот вариант применяется на практике. На рис. 3 приводится схема обработки текста программы при таком решении:

Преобразование, обратное синтаксическому отображению (шаг 3 на рис. 3), назовем *обратным синтаксическим отображением*.

Шаги 1, 2 и 3 (рис. 3) выполняются препроцессором, шаг 4 выполняется компилятором существующего языка (работа компилятора соответствует схеме, представленной на рис. 1, но это уже выходит за рамки рассматриваемой задачи).

После работы препроцессора желательно по возможности сохранить первоначальный (до шага 1) код программы без изменений в тех его частях, где расширения не используются, для возможности последующей частичной (в местах без расширений) отладки по коду с расширениями (при отладке, как правило, используется символьная отладочная информация и информация соответствия инструкций и номеров строк, кодирующих эти инструкции в тексте программы).

ЗАКЛЮЧЕНИЕ

Рассмотренный подход к расширению существующих языков программирования можно использовать для реализации языков предметной области, символьно интегрированных в базовый язык. Примеры таких решений существуют, однако выполнены они индивидуально для каждой задачи. Наличие же универсального инструмента для построения расширений существующих языков может многократно упростить реализацию таких решений, поэтому создание такого инструмента является актуальной задачей.

СПИСОК ЛИТЕРАТУРЫ

1. *Ахо А.* Теория синтаксического анализа, перевода и компиляции : в 2 т. / А. Ахо, Д. Ульман. — М. : Мир, 1978 г.

2. *Дмитриев С.* Языково-ориентированное программирование: следующая парадигма / С. Дмитриев // RSDN Magazine, № 5, 2005.

3. *В. С. Гуров* Текстовый язык автоматного программирования // тезисы докл. Междунар. науч. конф., посвященной памяти профессора А. М. Богомолова Компьютерные науки и информационные технологии. — Саратов : Изд-во Сарат. ун-та, 2007.

4. *Кириллов Д.* Ориентация на язык / Д. Кириллов // Компьютерра, — 2006. — № 10.

5. *Федоров А.* Software Factories — быстрый способ создания шаблонов приложений / А. Федоров// КомпьютерПресс, — 2007. — № 2.

6. *Martin Fowler.* Language Workbenches: The Killer-App for Domain Specific Languages? 2005. http://www.martinfowler.com/articles/language Workbench.html

7. *Dmitriev S.* Language Oriented Programming: The Next Programming Paradigm // on Board, № 2, 2005.

8. *Ward M.P.* Language-Oriented Programming // Software concepts & Tools. — v.15, n.4. — 1994, P. 147—161.

9. *Van Deursen, P. Klint* (1997). Little languages: little maintenance? Proceedings of the First ACM-SIGPLAN Workshop on Domain-Specific Languages (DSL '97), Report, University of Illinois at Urbana-Champaign, 109—127.

10. *Kurtev I., Jouault F., Bezivin J., Valduriez P.* Model-based DSL Frameworks. OOPSLA 2006 Companion Proceedings. 2006.

11. *Domain-Specific Language Tools.* MSDN: http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx.

12. Meta Programming System. http://www.jetbrains.com/mps/

13. Patrick Smacchia пер. Руслан Козлов // RSDN Magazine, — № 1. — 2006.

14. Nemarle Home Page: http://nemerle.org/

15. *K. Skalski, M. Moskal, and P. Olszta.* Metaprogramming in Nemerle, 2004.http://nemerle.org/metaprogramming.pdf