Павлов Дмитрий Алексеевич

УДК 004.42

СОЗДАНИЕ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ

Аннотация

Статья описывает подходы к созданию языков, ориентированных на конкретную предметную область. Такие языки называются предметно-ориентированными языками (DSL), и их принято отличать от более известных языков общего назначения. Статья освещает преимущества использования DSL и даёт обзор современных технологий для их создания.

Ключевые слова: предметно-ориентированный язык, парсинг, компиляция.

Предметно-ориентированный язык программирования (англ. domain-specific language, DSL) – это язык, предназначенный для решения узкого круга задач; противоположностью ему является язык общего назначения. Примеры: C# – язык общего назначения, а язык регулярных выражений – это предметно-ориентированный язык. Предметно-ориентированные языки рождаются на свет по разным причинам:

• Появление новой технологической ниши. Возникновение языка TeX дало начало эпохе компьютерного набора математических текстов. С развитием баз данных неизбежно было появление унифицированного языка управления данными, которым стал SQL.

• Неуклюжесть языков общего назначения нередко заставляет разработчиков на этих языках придумывать свои маленькие, более выразительные языки. Это относится к различным скриптовым языкам в компьютерных играх, написанных на C++. Также у программ на C, C++, C#, Java практически повсеместно встречаются так называемые файлы конфигурации; нетрудно понять, что язык файла конфигурации, как бы прост он ни был, представляет собой DSL, который появился по той единственной причине, что код на основном языке программы не может выполняться без специального этапа компиляции, чего на машине пользователя никогда не происходит.[1]

• Исторические причины. Примером являются математические программы, такие как Mathematica и Maxima. Язык математики складывался веками, и появление компьютеров в нём мало что изменило.

Три вышестоящих пункта охватывают *серьёзные* причины к появлению предметно-ориентированных языков. Но идея, которую преследует данная статья, заключается в том, чтобы дать понять: с нынешними технологиями создавать DSL стало много про-

[1] Внушающим уважение исключением является известный в узких кругах оконный менеджер для X11 под названием dwm (http://ru.wikipedia.org/wiki/Dwm). Параметры конфигурации dwm записаны непосредственно в .h-файле, и изменения параметров вступают в силу только после пересборки и перезапуска dwm.

ще, чем раньше. Сейчас можно создавать DSL, преследуя *сиюминутные* цели. Примерно так, как мы при программировании создаём новый класс или библиотечный модуль, под любую мелкую (или крупную) задачу.

Итак, вы сочинили предметно-ориентированный язык, описали его грамматику, продумали семантику, а дальше что? Как получить транслятор, отладчик, где взять удобную среду для написания кода на этом DSL? Не самому же всё делать с нуля. Вот об этом и речь.

Прежде всего, надо определиться: DSL или EDSL? Второй вариант, который расшифровывается как «встроенный предметно-ориентированный язык» (embedded DSL, или internal DSL), сделает вашу жизнь значительно легче. Вы выбираете язык общего назначения, для которого все средства разработки уже сделаны до вас, и создаёте свой язык как надстройку над первым, оставаясь в рамках его грамматики. Фактически, вы делаете библиотеку. Само собой, если изначальный язык (хост-язык) недостаточно гибок в плане грамматических конструкций, то и от DSL особенного удобства ждать не приходится. Тем не менее, можно на Ruby[1], C#[2], Scala[3] создавать вполне удобные EDSL. Даже на C++ с шаблонами можно создавать совершенно потрясающие вещи[4].

Вообще говоря, если вы выбрали EDSL, то есть готовы жертвовать грамматикой языка ради готовых средств разработки, то почему бы не жертвовать до конца. Откажитесь от грамматики и задавайте программу сразу как экземпляр метамодели (что принято называть абстрактным синтаксическим деревом, хотя оно не абстрактное, да и к синтаксису отношения не имеет). Возьмите платформу, которая заточена под работу с такими программами. Речь, конечно, о семействе лиспов (**Common Lisp, Scheme, Clojure**). Никакого синтаксиса, кроме простейших S-выражений. Динамическая типизация. Система компилируемых макросов. Культура инкрементальной разработки. Лиспам нет равных в быстроте и удобстве создания EDSL. В знаменитой книге «Структура и интерпретация компьютерных программ» (SICP) мини-языки в рамках Scheme создаются практически под каждую задачу и даже без использования макросов.

Мы подошли к месту, когда нельзя не упомянуть JetBrains **MPS**. Это весьма оригинальная вещь. В компании, где пишут среды разработки для мейнстримных языков программирования, зародилась идея создания IDE для создания IDE для создания DSL, по качеству аналогичных мейнстримным. С автодополнениями, рефакторингами, удобными отладчиками. В лиспе отсутствует культура автодополнений и рефакторинга, не развит статический анализ кода, непривычный разработчикам «скобочный» синтаксис. Лисп не годился. Пара вещей, однако, была заимствована из него: отказ от синтаксиса и макросы. Первое было даже не просто заимствовано, а доведено до абсолюта. В MPS нет ни парсера S-выражений, ни какого-либо другого парсера, потому что в MPS нет кода. Редактор MPS работает не с кодом, а непосредственно с экземпляром метамодели. В целях облегчения перехода разработчиков на данную технологию редактор *напоминает* текстовый, декорируя и отображая дерево так, что оно выглядит как код, но им не является. Инструментов для импорта кода MPS не предоставляет (за исключением кода на Java[5]). Что касается макросов для кодогенерации в MPS, то они там завязаны на строгую систему типов, что позволяет автоматически делать статическую проверку типов для создаваемого DSL.

Многие говорят, что MPS – это переизобретение Лиспа, и они неправы. Скорее, MPS – это скачок к визуальному метапрограммированию будущего, что не означает, однако, что этот скачок удачный. Время покажет. Возможно, молодое поколение будет

[1] http://www.khelll.com/blog/ruby/ruby-and-internal-dsls/
[2] http://fcs.codeplex.com/SourceControl/changeset/view/23139#320921
[3] http://debasishg.blogspot.com/2008/05/designing-internal-dsls-in-scala.html
[4] http://weegen.home.xs4all.nl/eelis/analogliterals.xhtml
[5] http://forum.jetbrains.net/thread/Meta-Programming-System-625

очаровано визуальной средой MPS и автодополнениями. С другой стороны, им может помешать изолированность MPS от мира текстовых программ, так как сейчас практически все языки текстовые. Их также может отпугнуть тотальная бюрократизация всего процесса программирования, что неудивительно, при том, что MPS основана на платформе Java. В общем, выбор делать им, а мы идём дальше, рассматривая тот случай, когда вам нужен именно текстовый DSL со своей уникальной грамматикой.

Итак, вы желаете реализовать настоящий DSL. С транслятором проблем не будет – связка lex+yacc, портированная, кажется, на все платформы в мире, уже лет 40 выполняет задачу автоматического построения парсеров. Для грамматик, которым не хватает yacc, есть другие инструменты. Но отладчик и IDE придётся писать самостоятельно. Если только не... найти хост-язык с настраиваемой грамматикой и взять существующую IDE для него, если она не сломается от введения новой грамматики. Фактически, получится EDSL, но без ограничений на синтаксис, которые накладывались бы нерасширяемым хост-языком. Этот подход называется «extensible programming»; он был довольно популярен в 1960-х, потом угас, но разгорелся с новой силой в XXI веке. Поэтому живых языков с настраиваемой грамматикой не очень много.

• **Forth** – старейший из таких языков. Отличается тем, что «своей» грамматики практически не имеет. Таков же его преемник **Factor**. Программирование на Forth, однако, может оказаться не таким простым даже для опытных программистов[1].

• **Common Lisp** отметился и здесь. Помимо defmacro, которые используются для создания «скобочных» EDSL, о которых было сказано выше, в CL есть так называемые макросы ридера (reader macros), которые позволяют изменить поведение ридера, то есть расширить синтаксис. Чтобы не конфликтовать со стандартным синтаксисом CL, reader macros обычно начинаются с «#», хотя могут начинаться и с любого другого символа. Например, #c(0, 1) – запись мнимой единицы одним из стандартных макросов CL. Известны примеры более интересных макросов: запуск команд оболочки прямо из CL[2], вызов C-функций[3], синтаксис для хеш-таблиц[4], расширенный синтаксис строк[5], и даже встраивание XML[6] непосредственно в лисповый код. Однако, если синтаксис вашего DSL конфликтует с синтаксисом S-выражений, то дело может оказаться труднее. В теории ничто не мешает изменить стандартный ридер на собственный, но рабочих примеров такой модификации не известно; кроме того, неизвестно, как на такой шаг отреагирует ваша IDE (как правило, это SLIME). Похоже, никто серьёзно не занимался вопросом создания языков с нестандартной грамматикой в CL.

• **Racket** – бывшая PLT Scheme – совсем другое дело. Платформа «из коробки» поддерживает концепцию переключения между разными языками и позволяет создавать новые языки, используя при этом генератор парсеров в стиле yacc. Есть примеры «нескобочных» языков, созданных на Racket, по мотивам Prolog[7] и Algol 60[8]. Среда (DrRacket) и её отладчик работают с этими языками. Более того, DrRacket написан на Racket же, что открывает возможности для модификации среды под язык и распространения её в качестве IDE для вашего DSL. Словом, Racket – очень перспективная платформа для создания DSL и поддерживается в прекрасном состоянии.

---

[1] http://www.yosefk.com/blog/my-history-with-forth-stack-machines.html
[2] https://github.com/jfm3/shellshock
[3] http://trac.clozure.com/ccl/wiki/OpenMclFfi
[4] http://frank.kank.net/essays/hash.html
[5] http://weitz.de/cl-interpol/
[6] http://www.cs.colorado.edu/%7Eralex/papers/PDF/X-expressions.pdf
[7] http://docs.racket-lang.org/datalog/Tutorial.html
[8] http://en.wikipedia.org/wiki/Racket_features#Algol

• **Nemerle** – статически типизированный язык для .NET с макросами и расширяемым синтаксисом. Возможности расширения, однако, ограничены. Есть проект Nemerle 2, в котором ограничения будут сняты и который обещает стать чем-то вроде MPS для .NET (но без отказа от грамматик). Пока, впрочем, дальше обещаний дело не зашло.

• **Bigloo** – ещё одна реализация Scheme с упором на быстродействие. Так же, как и Racket, обладает средствами генерации парсеров и поддерживает макросы, однако не имеет такого удобного механизма переключения языков и создания новых. Кроме того, не факт, что отладчик Bigloo и его IDE (основанное на Emacs) будут нормально работать с новыми языками, ибо, как и с CL, вряд ли кто-то специально занимался этим вопросом, а само собой ничего не делается.

• **Helvetia** – уникальная в своём роде разработка на базе языка Smalltalk, в которой используется то обстоятельство, что вся среда Smalltalk (включая парсер) поддаётся изменению в рантайме. Helvetia – инструмент для произвольного расширения синтаксиса Smalltalk с сохранением отладки. И даже с подсветкой и автодополнением! Что делает MPS не совсем уникальной средой. Примеры[1] включают SQL и что-то похожее на CSS. Единственная неприятность – единственный автор Helvetia защитил по ней PhD, ушёл работать в Google и забросил своё творение, лишь изредка возвращаясь и доделывая мелочи. Helvetia работает только на Pharo Smalltalk версии 1.1 и не портирована ни на современную версию Pharo (1.3), ни на Squeak. Кстати, автор в 2009 г. выбрал[2] в качестве хост-языка Smalltalk (а не Lisp) по причине «однородности» языка и среды. Среды разработки Smalltalk написаны на Smalltalk, а про Scheme или CL такого сказать было нельзя. DrRacket в то время был наполовину написан на C++, и его переписали на чистом Racket только в феврале 2011 г.

Как видите, в принципе есть инструменты на любой вкус и под любые требования. В наше время создавать DSL стало не только полезно, но и приятно.

[1] http://scg.unibe.ch/research/helvetia/examples
[2] http://scg.unibe.ch/archive/papers/Reng09bLanguageShootout.pdf

Abstract

The article gives a brief introduction to creation of languages that are designed for a particular application domain. Such languages are called domain-specific (DSL), and are treated differently from (more widespread) general-purpose languages. The article presents the advantages of using DSL-s in programming and gives a list of modern platforms for building them.

Keywords: domain-specific language, parsing, compilation.

Павлов Дмитрий Алексеевич,
преподаватель ФМЛ № 30,
dmitry.pavlov@gmail.com