УДК 004.423

ОБЗОР СОВРЕМЕННЫХ СРЕДСТВ СОЗДАНИЯ И ПОДДЕРЖКИ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Д.С. Ботов

REVIEW OF MODERN DEVELOPMENT AND SUPPORT TOOLS FOR DOMAIN-SPECIFIC PROGRAMMING LANGUAGE

D.S. Botov

Приводится обзор современных средств создания предметно-ориентированных языков программирования и языкового инструментария для их поддержки.

Ключевые слова: предметно-ориентированный язык программирования, языковой инструментарий, языково-ориентированное программирование, метапрограммирование, расширяемое программирование.

In this article the review of modern development tools for domain-specific programming languages and language workbenches is given.

Keywords: DSL, domain-specific language, language workbench, language-oriented programming, metaprogramming, extensible programming.

Введение

В современной индустрии разработки программного обеспечения постепенно набирает силу и становится все более популярной новая парадигма, новый подход – языково-ориентированное программирование (ЯОП). ЯОП – это такой подход к программированию, который основывается на создании специальных языков программирования – предметно-ориентированных языков (domain-specific language, DSL) для решения задач в конкретной предметной области [1]. В ЯОП программист сначала создает один или несколько DSL для решения определенного набора задач, а затем применяет созданные DSL при разработке программной системы.

Различают два основных вида предметно-ориентированных языков: **внешние** (external DSL) и **внутренние** (internal DSL или embedded DSL). Внешние DSL имеют собственный синтаксис, отделенный от основного языка приложения. Внутренние DSL используют в своей основе язык программирования общего назначения, но отличаются тем, что используют конкретное подмножество возможностей этого языка в определенном стиле [2].

В статье приведен обзор современных средств (языки, платформы, среды разработки), позволяющих создавать как внешние, так и внутренние DSL.

Стоит отметить, что одной из важных проблем в создании и дальнейшем использовании DSL является наличие **языкового инструментария** (language workbench). Языковые инструментальные средства представляют собой специализированные интегрированные среды разработки (integrated development environment, IDE) для определения и создания DSL [2]. Именно сложность создания языковой инфраструктуры, необходимой для реализации DSL различного рода и комфортной работы с ними, является одной из причин малого применения DSL в промышленной разработке программного обеспечения (ПО), где важна высокая производительность разработчиков.

В данном обзоре особое внимание уделяется технологиям, позволяющим обеспечить поддержку разработки в стиле ЯОП средствами языкового инструментария.

1. Средства поддержки разработки внешних DSL

Процесс создания языка с собственным синтаксисом, т. е. внешнего DSL, можно представить в виде последовательности из трех шагов:

– определение семантической модели;

– определение синтаксической модели (абстрактный и конкретный синтаксис);

– определение правил трансформации (правила, по которым абстрактное представление транслируется в исполнимое).

Ботов Дмитрий Сергеевич – аспирант кафедры ЭВМ, Южно-Уральский государственный университет; dm@comp.susu.ac.ru

Botov Dmitry Sergeevich – Post-Graduate Student of Electronic Computer Department of South Ural State University; dm@comp.susu.ac.ru

Если для определения конкретного синтаксиса языка и задания правил трансформации путем построения транслятора языка есть готовые средства различного рода, начиная от связки программ lex + yacc, входящих в стандарт POSIX, для генерации лексического и синтаксического анализатора соответственно и заканчивая современными средствами автоматизации построения трансляторов языка, например ANTLR.

ANTLR [3] – генератор парсеров, позволяющий конструировать компиляторы, интерпретаторы, трансляторы с различных формальных языков по описанию LL(*)-грамматики на языке, близком к РБНФ. ANTLR предоставляет визуальную среду разработки, которая позволяет создавать и отлаживать грамматики с поддержкой подсветки синтаксиса, автодополнения, с визуальным отображением грамматик, отладчиком, инструментами для рефакторинга и т. д.

Для определения же семантической модели языка [2] (той части языка, описывающей семантику предметной области или определенный аспект системы, для конфигурирования которого разрабатывается внешний DSL) не существует специальных программных средств, каждая команда разработчиков решает проблему представления семантики DSL самостоятельно, обычно описывая метамодель языка на одном из языков программирования (как правило, языков программирования общего назначения). Кроме того, не существует и средств отображения семантики DSL на синтаксис. Задача такого отображения опять же решается конкретной группой разработчиков самостоятельно.

Из-за отсутствия языкового инструментария для поддержки семантической модели языка и поддержания отображения семантики на синтаксическую модель возрастают затраты на создание внешних DSL. Сами внешние DSL становятся замкнутыми языками для решения узких задач, и на практике практически невозможно и нерационально их повторное использование для решения задач в других предметных областях (смежных, аналогичных по характеру или по содержанию).

Среду разработки, которая поддерживала и облегчала бы написание сценариев на внешнем DSL, обычно разрабатывают либо с нуля, либо как плагин к уже существующей современной IDE. Практически все современные среды разработки (например, Eclipse IDE, Microsoft Visual Studio и др.) имеют гибкую плагинную архитектуру и позволяют добавлять поддержку новых языков программирования.

2. Средства поддержки разработки внутренних DSL, основанных на грамматиках языков программирования общего назначения

В простейшем случае при создании внутреннего DSL мы выбираем один из языков программирования общего назначения в качестве базового языка и разрабатываем на его основе библиотеку, своего рода надстройку над языком [4], которую затем используем в определенном стиле, как правило, для управления отдельными аспектами разрабатываемой программной системы.

Нужно понимать, что в отличие от внешнего DSL при разработке внутреннего DSL грамматика базового языка накладывает ограничения на выразительные возможности языка. Чем менее гибка грамматика базового языка, тем менее удобен и эффективен будет внутренний DSL. Таким образом, выразительные возможности базового языка должны соответствовать сфере и способу применения создаваемого на его основе внутреннего DSL.

При создании внутренних DSL чаще всего основываются на грамматике современных языков программирования общего назначения, предоставляющих гибкие возможности, которые позволяют создавать удобные DSL. Например, это такие языки как Ruby, Python, Scala, C#, F#, Haskell. Можно заметить, что в списке таких языков преимущественно языки мультипарадигмальные, как правило, наследующие выразительные возможности от нескольких, чаще всего неродственных языков. За счет такого сочетания разнообразных возможностей в грамматике языка мы получаем эффективный гибкий инструмент для создания внутренних DSL. Особенно стоит отметить эффективность использования выразительных возможностей парадигмы функционального программирования, хотя даже на нефункциональных языках со статической типизацией можно создавать удобные внутренние DSL, например, на C++ за счет механизма шаблонов (templates).

Выбирая современный язык программирования общего назначения как основу для создания внутреннего DSL, мы сразу получаем и готовый набор средств поддержки разработки – современные IDE, которые поддерживают базовый язык.

Таким образом, создавая внутренние DSL, мы жертвуем полной свободой определения грамматики, оставаясь в рамках грамматики базового языка, но при этом получаем возможность использовать современные интегрированные среды разработки.

3. Средства поддержки разработки DSL, основанных на языках и средствах программирования с настраиваемой грамматикой

Еще одним подходом для создания DSL (по сути внутренних DSL) является использование языков программирования с настраиваемым синтаксисом, т. е. языков, ориентированных на техники метапрограммирования. Такой подход называется «extensible programming»; он активно развивался в 1960-х годах, потом его развитие приостановилось, и интерес к этому подходу вновь возник только в XXI веке [4].

«Extensible programming» – это стиль программирования, ориентированный на использова-

ние механизмов расширения языков программирования, трансляторов и сред выполнения [5].

Примерами языков программирования, которые могут использоваться в качестве базовых языков для создания внутренних DSL в стиле «extensible programming», могут служить:

– **Forth** – один из первых конкатенативных языков программирования (появился в 1971 году), в котором программы записываются последовательностью лексем в виде постфиксной записи при использовании стековой нотации. Язык имеет простую грамматику и ориентирован на использование механизмов метарасширения синтаксиса. У Forth существует преемник – язык Factor [6]. Оба языка достаточно сложны в изучении и использовании.

– **Common Lisp** [7] – диалект языка Lisp, стандартизированный ANSI. Common Lisp включает в себя CLOS – систему Lisp-макросов, позволяющую вводить в язык новые синтаксические конструкции, использовать техники метапрограммирования и обобщенного программирования. Существуют текстовые среды разработки на Common Lisp (например, SLIME, Superior Lisp Interaction Mode for Emacs [8] – режим Emacs для разработки приложений на Common Lisp).

– **Nemerle** [9] – статически типизированный язык, сочетающий в себе возможности функционального и объектно-ориентированного программирования, для платформ .NET и Mono с макросами и расширяемым синтаксисом. Nemerle имеет бесплатную полноценную IDE, основанную на Visual Studio 2008 Shell, а также может интегрироваться с полноценной Visual Studio 2008, Visual Studio 2010. Однако на возможности расширения синтаксиса наложены существенные ограничения.

– **Racket** [10] – это язык и платформа программирования, являются реализацией и расширением языка Scheme – еще одного диалекта Lisp. Платформа поддерживает концепцию переключения между разными языками и позволяет создавать новые языки, используя при этом генератор парсеров в стиле yacc. Интегрированная среда разработки (DrRacket) и ее отладчик работают с этими языками. Более того, DrRacket также написан на языке Racket, что открывает возможности для модификации среды под язык и распространения ее в качестве IDE для создаваемого DSL.

– **Helvetia** [11] – инструментарий, интегрированная среда, написанная на Smalltalk, для произвольного расширения синтаксиса языка Smalltalk. Позволяет расширять и адаптировать среду разработки под расширение языка с сохранением отладки, с подсветкой синтаксиса, автодополнением. Это становится возможным благодаря однородности базового языка и среды. В качестве недостатка можно отметить, что Helvetia работает только на Pharo Smalltalk версии 1.1 и не портирована на современную версию Pharo 1.3.

4. Языки для поддержки разработки DSL без текстовых грамматик

Синтаксис всех популярных языков программирования общего назначения (в том числе приведенные в качестве примеров выше в п. 2 и 3) основывается на текстовых грамматиках. У таких грамматик есть один существенный недостаток: при попытке расширения грамматики она может стать неоднозначной, т. е. возможно несколько интерпретаций одной и той же строки исходного кода на таком расширенном языке [12]. Особенно остро данная проблема встает в случае, если мы пытаемся соединить несколько разных расширений грамматики одного и того же языка, которые в отдельности являются однозначными, но при их совмещении результирующая грамматика вполне может потерять однозначность и ее дальнейшее использование будет уже невозможно.

Проблема неоднозначности грамматик может быть решена путем отказа от использования текстовой грамматики как таковой и в этом случае программа будет задаваться как экземпляр некоторой синтаксической метамодели. Обычно метамодель программы представляется в виде абстрактного синтаксического дерева [4]. Создание нового DSL в таком подходе сводится к заданию метамодели DSL средствами базового языка. Ярким примером использования такого подхода к разработке DSL является семейство языков Lisp: Common Lisp [7], Scheme [13], Clojure [14] (Common Lisp уже упоминался выше, так как его можно использовать и в подходе «extensible programming»).

Lisp (от англ. LISt Processing language – «язык обработки списков») – семейство языков программирования, программы и данные в которых представляются системами линейных списков символов. Lisp является вторым в истории (после Фортрана) используемым по сей день высокоуровневым языком программирования [15]. Изначально Lisp создавался как средство моделирования различных аспектов искусственного интеллекта, но затем сфера применения языка расширилась.

Благодаря минималистичному собственному синтаксису языка, его динамической типизации, развитой системе компилируемых макросов, культуре инкрементальной разработки и другим особенностям языкам семейства Lisp, пожалуй, нет равных в быстроте и удобстве создания внутренних (встроенных) DSL. Однако при создании DSL на Lisp мы сталкиваемся с другой проблемой – сложностью создания языковой инфраструктуры, необходимой для реализации нового DSL и комфортной работы с ним.

5. Современные языковые инструментальные средства для поддержки разработки DSL без текстовых грамматик

Языковые инструментальные средства, по сути, представляют собой инструменты, которые не только помогают создать собственный DSL, но и

обеспечивают его поддержку в стиле современных интеллектуальных сред разработки, предоставляя возможности для построения современных IDE под создаваемые языки [2]. В результате программисты, которые будут писать сценарии на DSL, получат такую же инструментальную поддержку, как и программисты, разрабатывающие на языках программирования общего назначения (C/C++, Java, C# и т. д.). Среды разработки для DSL смогут предоставлять такие возможности, без которых немыслима современная промышленная разработка ПО, как например:

– автодополнение, генерация кода, средства рефакторинга;

– средства удобной отладки сценариев на DSL;

– средства управления версиями;

– средства модульного и интеграционного тестирования;

– средства обратной разработки (reverse engineering).

Существуют фреймворки для создания интеллектуальных редакторов, например, IntelliJ IDEA Language API [16], Xtext [17] (на базе платформы Eclipse) и другие, но ни один из этих фреймворков не поддерживает расширяемые языки на должном уровне. Даже если нам не нужна расширяемость, создание поддержки языка с использованием этих средств требует хороших знаний в области языков программирования и занимает очень много времени [12].

Частично проблему создания языковой инфраструктуры для DSL решала система Helvetia, кратко описанная выше. Разберем еще один из подходов для решения проблем при создании DSL, который предлагает компания JetBrains в своем продукте JetBrains MPS.

JetBrains MPS (MetaProgramming System) [18] – это система метапрограммирования, которая реализует парадигму языково-ориентированного программирования, является средой разработки языков и в то же время IDE для разрабатываемых языков.

Для того чтобы поддержать совместимость расширений языка друг с другом, MPS не работает с программами как с текстом. Вместо этого MPS хранит их как синтаксическое дерево, и редактирование происходит напрямую, без промежуточного использования текста. В результате вместо конкретного синтаксиса языка в MPS определяется абстрактный синтаксис (структура синтаксического дерева) для создаваемого DSL [12]. Способ описания структуры абстрактного синтаксического дерева чем-то напоминает XML Schema – язык описания структуры XML-документа.

Система JetBrains MPS позволяет вставлять код на языке программирования общего назначения внутрь относительно замкнутого DSL. Наибольшее число экспериментов проводится с расширением и использованием вставок на Java, так как сама MPS написана на этом языке программирования.

Поскольку в MPS не используется традиционное текстовое представление, то вместо обычного текстового редактора для работы с кодом на DSL предлагается использовать специальный проекционный редактор [12]. Для каждого узла синтаксического дерева, он создает проекцию – часть экрана, с которой может взаимодействовать пользователь. Такой редактор ведет себя достаточно близко к тестовому редактору, но чтобы освоить его в полной мере требуется время на обучение (по словам разработчиков из JetBrains на освоение проекционного редактора может потребоваться около 2 недель).

Такой подход позволяет существенно упростить создание IDE, поскольку постоянное наличие синтаксического дерева и созданного для работы с ним проекционного редактора позволяет легко реализовать подсветку ошибок, автоматическое дополнение, контекстные подсказки и т. п.

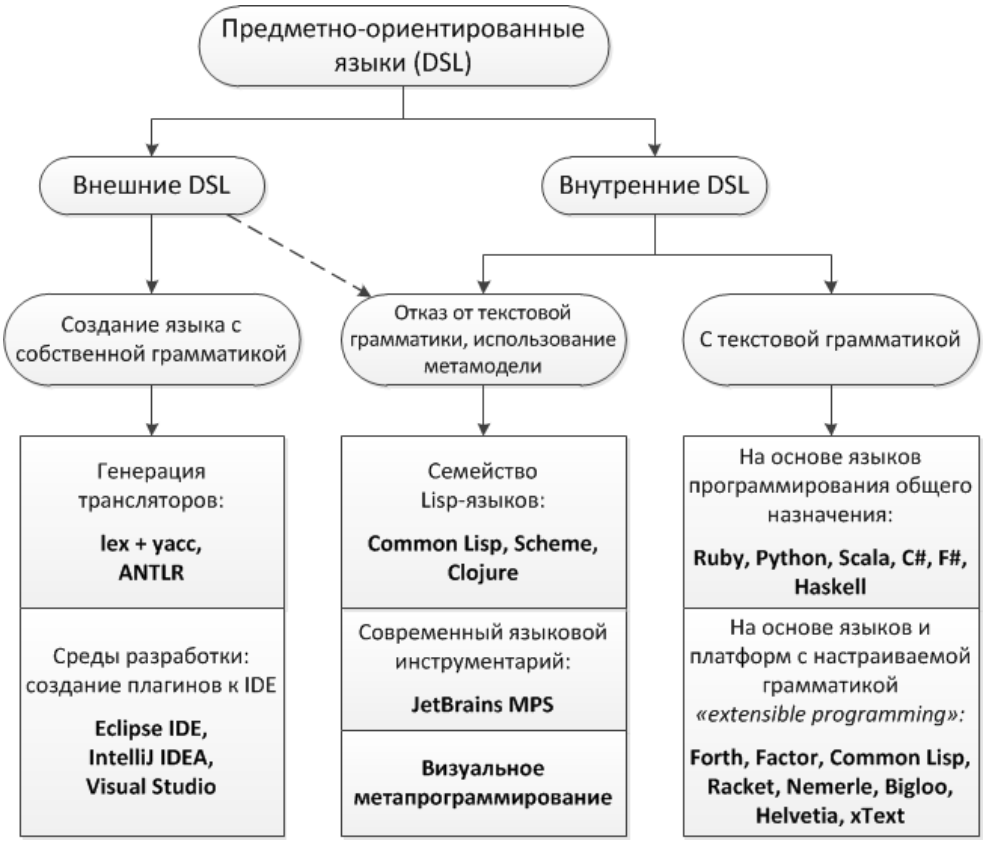
Среда разработки JetBrains MPS была создана на основе IntelliJ IDEA [16], в которой была реализована поддержка интеллектуального редактирования для многих языков. Реализация такой поддержки для нового DSL, создаваемого вне MPS, потребовала бы больших усилий. С MPS аналогичные возможности могут быть реализованы с минимальными затратами. Это возможно, поскольку для разработки языков используются специальные языки, которые конфигурируют существующую языковую инфраструктуру. Таким образом, MPS – это не просто редактор кода, а среда для создания полноценных IDE для DSL.

Компания JetBrains сама в свою очередь активно использует MPS в своих разработках. Так, например, с помощью MPS был создан багтрекер YouTrack и Realaxy ActionScript Editor – первая IDE, основанная на MPS.

Заключение

Подводя итоги обзора, можно отметить, что весьма невелико число языков и средств, которые позволяют не только создавать DSL, но и могут обеспечить эффективные средства и способы модернизации самих DSL и поддержки разработки на созданных языках. Пожалуй, только JetBrains MPS и частично Helvetia (но в отличие от JetBrains MPS эта система практически не развивается) представляют собой современные языковые инструментальные средства, покрывающие не только этапы создания, но и этапы эксплуатации и модернизации DSL, без чего невозможно представить их эффективное использование в индустрии разработке ПО сегодня.

Использование языкового инструментария требует детального изучения и наработки опыта применения таких инструментов при создании DSL. Продукт JetBrains MPS, рассматриваемый в данном обзоре, имеет достаточно высокий порог



Подходы к созданию внешних и внутренних DSL, языки и инструментарий создания и поддержки DSL

вхождения, что затрудняет его широкое распространение и применение. Кроме того еще одним недостатком современных языковых инструментариев является отсутствие стандартов и возможности переноса разрабатываемых DSL между различными средами. Так, начав создание DSL в JetBrains MPS, разработчик языка становится заложником этого инструментария, так как в нем отсутствуют возможности экспорта создаваемого DSL. Это обусловлено отсутствием каких-либо общепринятых форматов для представления DSL и существенным различием в подходах к созданию DSL [2, 4].

Стоит обратить особое внимание на то, что ни один из инструментов для создания DSL не позволяет в полной мере представить семантическую модель языка и настроить отображение семантики языка на синтаксис. В основном все средства сосредоточены на представлении текстовой грамматики DSL либо на представлении метамодели, под которой обычно понимается абстрактное синтаксическое дерево. Однако М. Фаулер в [2] справедливо отмечает, что семантические модели DSL, как правило, отличаются от абстрактного синтаксического дерева. Если синтаксическое дерево соответствует структуре сценариев DSL – форме (как например, в JetBrains MPS), то семантическая модель языка в свою очередь основывается на том, как будет обрабатываться информация сценария – это смысл, содержание.

Семантическая модель должна отражать суть, специфику предметной области. И именно наличие семантической модели является одним из важных отличий работы с DSL от работы с языками программирования общего назначения. Можно предположить, что отсутствие эффективных средств представления семантики DSL и отображения семантики на синтаксис осложняет широкое применение DSL в разработке ПО.

Разнообразие подходов к созданию DSL наглядно представлено на рисунке. Как обобщение данного обзора на схеме изображены подходы и средства создания и поддержки DSL, которые были рассмотрены выше.

На схеме также упомянут подход визуального метапрограммирования, рассмотрение которого выходит за рамки данной статьи, так как он существенно отличается по своим принципам и инструментарию от других подходов и требует отдельного детального изучения и рассмотрения.

Литература

1. Language-oriented programming. – http://en.wikipedia.org/wiki/Language_oriented_programming

2. Фаулер, М. Предметно-ориентированные языки программирования: пер. с англ. / М. Фаулер. – М.: ООО «И.Д. Вильямс», 2011. – 576 с.

3. ANTLR, ANother Tool for Language Recognition. – http://www.antlr.org

4. Как создавать DSL. – http://shmat-razum.blogspot.ru/2011/09/dsl.html

5. Extensible programming. – http://en.wikipedia.org/wiki/Extensible_programming

6. Factor programming language. – http://factorcode.org/

7. Common Lisp. – http://common-lisp.net/

8. SLIME, The Superior Lisp Interaction Mode for Emacs. – http://common-lisp.net/project/slime/

9. Nemerle, programming language. – http://nemerle.org/

10. Racket, programming language. – http://racket-lang.org/

11. Helvetia. – http://scg.unibe.ch/research/helvetia

12. Соломатов, К. Как система JetBrains MPS позволяет достичь более широкого использования DSL-й (языков специфичных для предметной области) / К. Соломатов. – http://habrahabr.ru/post/66094/

13. Scheme, dialect of Lisp. – http://schemers.org/

14. Clojure, dialect of Lisp. – http://clojure.org

15. LISP, LISt Processing language. – http://ru.wikipedia.org/wiki/Lisp

16. IntelliJ IDEA, code-centric IDE. – http://www.jetbrains.com/idea/

17. Xtext, framework for development of programming languages and domain specific languages. – http://www.eclipse.org/Xtext/

18. JetBrains MPS, MetaProgramming System. – http://www.jetbrains.com/mps/

Поступила в редакцию 27 ноября 2012 г.