ИЗВЕСТИЯ
ПЕНЗЕНСКОГО ГОСУДАРСТВЕННОГО
ПЕДАГОГИЧЕСКОГО УНИВЕРСИТЕТА
имени В. Г. БЕЛИНСКОГО
ФИЗИКО-МАТЕМАТИЧЕСКИЕ И ТЕХНИЧЕСКИЕ НАУКИ
№ 13 (17) 2009

IZVESTIA
PENZENSKOGO GOSUDARSTVENNOGO
PEDAGOGICHESKOGO UNIVERSITETA
imeni V. G. BELINSKOGO
PHYSICAL, MATHEMATICAL AND TECHNICAL SCIENCES
№ 13 (17) 2009

УДК 004.424

РЕАЛИЗАЦИЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ СРЕДСТВАМИ ФРЕЙМВОРКА Xtext

© М. В. ЖУКОВ
Пензенский государственный педагогический университет имени В.Г. Белинского,
кафедра прикладной математики и информатики
e-mail: maxim.zh@gmail.com

Жуков М. В. – Реализация предметно-ориентированных языков средствами Фреймворка Xtext // Известия ПГПУ им. В. Г. Белинского. 2009. № 13 (17). С. 98–102. – *Предметно-ориентированные языки (DSL – Domain Specific Language) становятся все более популярной темой в сфере программной инженерии. Данная статья представляет собой руководство по созданию DSL с помощью фреймворка Xtext, который позволяет за короткое время создать не только синтаксический анализатор для разрабатываемого языка, но и удобный текстовый редактор, проверяющий синтаксические и семантические ограничения языка; компилятор или интерпретатор для DSL.*
Ключевые слова: предметно-ориентированный язык, синтаксический анализатор, интерпретатор, компилятор.

Zhukov M. V. – The realization of object-oriented languages by means of framework Xtext// Izv. Penz. gos. pedagog. univ. im.i V. G. Belinskogo. 2009. № 13 (17). P. 98–102. – *Domain Specific Languages are a hot topic nowadays. In this article we will show you how to create your own DSL with the Xtext framework. Xtext gives you opportunity to create in a brief space of time not only a parser for DSL, but: a comfortable text editor, checking syntactic and semantic constrains; a compiler and an interpreter for DSL.*
Keywords: domain specific language, parser, interpreter, compiler.

Предметно-ориентированный язык – это язык, ориентированный на решение частных, ограниченных рамками предметной области, задач в отличие от языков общего назначения (GPL – General Purpose Language), которые предназначены для решения широкого круга задач, возникающих перед программистом.

Несмотря на то, что множество статей было написано о том, как разрабатывался тот или иной DSL, однако очень мало литературы посвященной методологии их разработки. Наибольшего внимания заслуживает работа [2], содержащая описание следующих фаз:

- принятие решения о разработке DSL;
- анализ предметной области;
- разработка DSL;
- реализация DSL.

Одна из сложностей при создании DSL – его реализация. Этому вопросу и посвящена данная статья, в которой мы покажем, как легко реализовать свой собственный DSL с помощью Eclipse Modeling Project (EMP).

Среди DSL различают внутренние (internal) DSL, написанные на том же языке, в рамках которого они будут использоваться, и внешние (external) DSL, обладающие собственным синтаксисом и требующие написание синтаксического анализатора для их обработки [1]. В то время как реализация internal DSL не составляет особо большого труда, реализация external DSL не так тривиальна. Поэтому мы рассмотрим реализацию именно external DSL средствами фреймворка Xtext.

Мы начнем с определения грамматики нашего DSL в Xtext. Затем, используя средства Xtext, создадим синтаксический анализатор и текстовый редактор для нашего языка. После этого покажем, как можно сгенерировать код, используя язык Xpand, и в заключение рассмотрим, как улучшить редактор с помощью Xtend.

Создание нового Xtext проекта. Для создания Xtext проекта вам необходимо:

– запустить Eclipse 3.4 с установленным oAW 3.4 (openArchitectureWare);

– выбрать *Fil > New > Project > openArchitectureWare > XtextProject*;

– указать имя создаваемого языка (у нас это *dbmanipulator*);

– нажать *Finish*.

В результате будет создано три проекта:

– *dbmanipulator.dsl* – здесь будет объявлена грамматика для нашего DSL. После запуска Xtext ге-

нератора, сюда же будет добавлен синтаксический анализатор DSL;

– *dbmanipulator.dsl.editor* – будет содержать DSL редактор;

– *dbmanipulator.dsl.generator* – здесь будет содержаться описание процедуры трансформации DSL.

Определение грамматики языка. В качестве примера мы рассмотрим DSL, предназначенный для генерации скрипов создания MySQL транзакций. Для каждой транзакции необходимо будет указать список параметров и тело. Телом транзакции могут быть либо SQL-запросы, либо последовательный вызов других транзакций.

Специфицируем грамматику языка с помощью Xtext грамматики. Для этого откройте файл *dbmanipulator.dsl/src/dbmanipulator.xtxt* и наберите текст, представленный на рис. 1:

– правило *Program* (строки 1-5) говорит о том, что текст программы на нашем DSL должен начинаться с ключевого слова “*Begin*” и оканчиваться словом “*End*”. Между ними может располагаться [0:n] элементов записанных по правилу *Element*;

– правило *Action* (строки 12-15) указывает, что элемент *Action* должен начинаться с фразы “*Create_action*”, далее в угловых скобках указывается уникальное имя *Action*, в круглых скобках через запятую список аргументов (*Argument*), если он есть (знак ? – означает кратность [0:1]), и тело (*Body*) элемента *Action* в фигурных скобках;

– *Argument* (строки 17-18) состоит из имени аргумента и его типа, указанного через двоеточие. (Допустимы два типа аргументов: “*Integer*” и “*Varchar(...)*”, см. строки 9-10);

```
 1 Program:
 2     "Begin"
 3         (elements += Element)*
 4     "End";
 5
 6 Element:
 7     Action;
 8
 9 String BaseType:
10     "Integer" | "Varchar" "(" INT ")";
11
12 Action:
13     "Create_action" "<" name = ID ">"
14         ("with_arguments:" "(" (args += Argument)("," args += Argument)* ")" )?
15         "{" body = Body "}";
16
17 Argument:
18     name = ID ":" type = BaseType;
19
20 Body: ((actionCalls += ActionCall)+ | code = STRING);
21
22 ActionCall: action = [Action] "(" ((args += ID)("," args += ID)*)? ")"";";
```

Рис. 1. Описание грамматики DSL

– *Body* (строка 20) представляет собой либо композицию [1:n] элементов *ActionCall*, либо текст произвольного содержания;

– правило *ActionCall* определяет элемент *ActionCall* начинающийся с имени одного из уже объявленных элементов *Action* (ограничение задается с помощью конструкции *[Action]*) и списка идентификаторов.

Создание DSL-редактора. После того как была специфицирована грамматика языка, мы можем сразу же создать текстовый редактор для DSL: в контекстном меню Xtext редактора выберите пункт “*Generate Xtext Artifacts*”.

Использование DSL-редактора. Проект *dbmanipulator.dsl.editor* построен на Eclipse plug-in архитектуре. Самый простой способ запустить его – выбрать из контекстного меню *RunAs > Eclipse Application*. Для тестирования DSL-редактора, в появившемся Eclipse Application, создадим новый проект (*TestMyDSL*) *File > New > Other... > Xtext DSL Wizards > dbmanipulator Project*. Чтобы продемонстрировать возможности сгенерированного редактора наберите текст, приведенный на рис. 2. Вы увидите, что он предоставляет следующие возможности:

– цветовую разметку синтаксиса;

– автоматическое завершение фраз (используйте *CTRL–Space*);

– навигацию (используйте *F3*, когда курсор находится на идентификаторе);

– свертывание элементов;

– проверку синтаксиса и маркировку ошибок.

Генерация кода. Теперь, когда DSL разработан, нужно сделать так, чтобы компьютер смог понимать его. Для этого существует два подхода: написать компилятор (также называемый генератором), который трансформирует выражения DSL в другой язык, либо разработать интерпретатор. Xtext позволяет создавать как генератор (Xpand), так и интерпретатор (Ecore-model). Однако мы остановимся только на первом варианте – генерации кода.

```
1  Begin
2      Create_action <AddEmployee> with_arguments:
3          ( name:Varchar(45), age:Integer )
4      {
5          "INSERT INTO Employee(name, age) VALUES (name, age);"
6      }
7      Create_action <UpdateEmployee> with_arguments:
8          (oldName:Varchar(45), newName:Varchar(45), age:Integer)
9      {
10         "UPDATE Employee SET name=newName, age=age
11             WHERE name=oldName;"
12     }
13     Create_action <AddAndUpdateEmployee> with_arguments:
14         (name:Varchar(45), age:Integer, newName:Varchar(45))
15     {
16
17         AddEmployee(name   AddAndUpdateEmployee
18         UpdateEmployee(n   AddEmployee
19     }                      UpdateEmployee
20 End
```

Рис. 2. Редактор dbmanipulator

Рассмотрим, как с помощью языка Xpand написать шаблон, на основе которого генерируется SQL-скрипт создания транзакции для каждого элемента Action.

В workspace был сгенерирован проект *dbmanipulator.dsl.generator*, который содержит файл *Main.xpt* – Xpand-шаблон. Измените его так, как показано на рис. 3.

```
1 «IMPORT dbmanipulator»
2
3 «DEFINE main FOR Program»
4    «EXPAND code FOREACH elements.typeSelect(Action)»
5 «ENDDEFINE»
6
7 «DEFINE code FOR Action»
8 «FILE name + ".sql"-»
9 DELIMITER $$
10 DROP PROCEDURE IF EXISTS `dsl`.`«name»` $$
11 CREATE PROCEDURE `dsl`.`«name»`(«FOREACH args AS arg ITERATOR it-»
12 «arg.name» «arg.type»«IF !it.lastIteration», «ENDIF-»
13 «ENDFOREACH»)
14 BEGIN
15 «EXPAND body FOR this.body-»
16 END $$
17 DELIMITER ;
18 «ENDFILE-»
19 «ENDDEFINE»
20
21 «DEFINE body FOR Body-»
22 «IF code != null-»
23     «code»
24 «ELSE-»
25 «FOREACH actionCalls AS actionCall-»
26     CALL «actionCall.action.name»(«FOREACH actionCall.args AS arg ITERATOR it-»
27 «arg»«IF !it.lastIteration», «ENDIF-»
28 «ENDFOREACH-»);
29 «ENDFOREACH-»
30 «ENDIF-»
31 «ENDDEFINE»
```

Рис. 3. Xpand-шаблон для dbmanipulator DSL

Процедура *main* (строки 3-5) будет вызвана для элемента типа *dmmanipulator::Program*, который является корневым элементам нашего DSL. Внутри этой процедуры, вызывается другая процедура (*code*) (строка 4) для каждого элемента типа *Action*, находящегося внутри элемента *Program*.

Процедура *code* объявлена для элементов типа *Action*. В этой процедуре мы создаем файл (строка 8), имя которого совпадает с именем текущего элемента *Action* и расширением *sql*. Весь текст, расположенный между строками 8 и 18, будет записан в созданный файл. Xpand содержит элементы управления (*FOR, IF, ...*) и средства доступа к Ecore-модели. Смотрите документацию по openArchitectureWare [3] для более подробной информации.

Чтобы посмотреть созданный шаблон в действии, необходимо запустить генератор кода. Для этого, в контекстном меню файла *TestMyDSL.oaw* (проект *TestMyDSL*) нажмите *Run as > oAW Workflow*. В результате будут сгенерировано три файла: *AddEmployee.sql*, *UpdateEmployee.sql* и *AddAndUpdateEmployee.sql*. Текст файла *AddAndUpdateEmployee.sql* приведен на рис. 4.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `dsl`.`AddAndUpdateEmployee` $$
CREATE PROCEDURE `dsl`.`AddAndUpdateEmployee`
  (name Varchar(45), age Integer, newName Varchar(45))
BEGIN
  CALL AddEmployee(name, age);
  CALL UpdateEmployee(name, newName, age);
END $$
DELIMITER ;
```

Рис. 4. AddAndUpdateEmployee.sql

Добавление ограничений предметной области. Хотя созданный DSL редактор и позволяет пользователю избежать синтаксически неверных конструкций, пользователь по-прежнему может нарушать ограничения, накладываемые семантикой предметной области. Для добавления семантических ограничений используется язык Check.

Давайте создадим ограничение, гарантирующее, что элементы типа *ActionCall* будут содержать такое же количество аргументов, что и связанный с ними элемент типа *Action*.

Чтобы сделать это, откройте файл *dbmanipulator.dsl/src/dbmanipulator/Checks.chk* и добавьте туда текст, приведенный на рис. 5 в секции *Check.chk*. *ActionCall* указывает тип элемента, для которого предназначена проверка. Далее специфицируется сообщение, которое будет выводиться, если возвращаемое методом *checkParametersCount* значение будет *false*. Реализация метода *checkParametersCount* написана на языке Xtend и находится в файле *dbmanipulator.dsl/src/dbmanipulator/Extensions.ext* (см. секцию *Extension.ext* рис. 5). Теперь, если число аргументов элемента типа *ActionCall* не будет соответствовать требуемому, то появится сообщение представленное в секции *model.dsl* (рис. 5).

Усовершенствуем наш редактор так, чтобы при вводе пользователем параметров элемента типа *ActionCall* автоматически предлагался список допустимых имен параметров. Для этого в файл *dbmanipulator.dsl.editor/src/dbmanipulator/ContentAssist.ext* добавьте код на языке Xtend приведенный на рис. 6 в секции *ContentAssist.ext*. Результат представлен в секции *model.dsl*.

Checks.chk
```
context ActionCall ERROR "Missing arugments count. Should be "
        + action.args.size:
   this.checkParametersCount();
```

+

Extensions.ext
```
Boolean checkParametersCount(ActionCall actionCall):
   actionCall.args.size == actionCall.action.args.size;
```

=

*model.dsl
```
13     Create_action <AddAndUpdateEmployee> with_arguments:
14         (name:Varchar(45), age:Integer, newName:Varchar(45))
15     {
16         AddEmployee( age);
17         Update[Missing arugments count. Should be 2] age);
18     }
```

Рис. 5. Check

Что дальше? Данная статья представляет лишь краткое рассмотрение возможностей, предоставляемых фреймворком Xtext при разработке и реализации DSL. Для получения более полной информации обратитесь к документации по openArchitectureWare [3].

*ContentAssist.ext

```
List[Proposal] completeActionCall_args(emf::EObject ctx, String prefix) :
    let actionCall = (ActionCall)ctx:
        let action = (Action)actionCall.eContainer.eContainer:
            action.args.select(e | e.type == positionType(actionCall))
            .collect(e| newProposal(e.name));

String positionType(ActionCall actionCall):
    actionCall.action.args.get(actionCall.args.size).type;
```

=

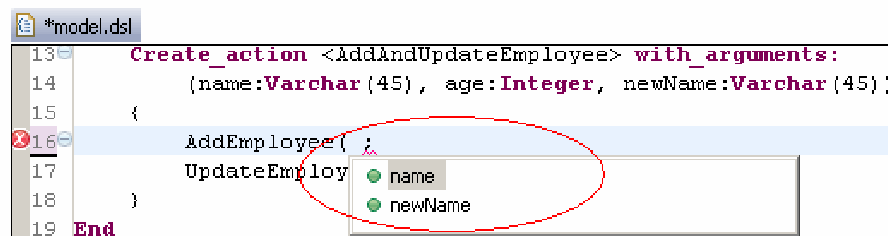

Рис. 6. Улучшение редактора

СПИСОК ЛИТЕРАТУРЫ

1. Fowler M. Domain Specific Languages. http://martinfowler.com/bliki/DomainSpecificLanguage.html
2. Mernik M., Heering J., Sloane A. When and how to develop domain-specific languages. Software Engineering Report SEN-E0517, 2005. 48p.
3. Документация по openArchitectureWare http://www.eclipse.org/gmt/doc/ .