ОТК. Быстрое знакомство

версия 2.5

Зеленов Сергей
zelenov@ispras.ru

1 Введение

В данном документе рассматривается процесс разработки тестов с использованием инструмента OTK. В качестве примера рассмотрен процесс разработки тестов для модулей компиляторов, содержащих преобразования арифметических выражений.

Модуль компилятора, для которого разрабатываются тесты, будем называть *целевым модулем*. Язык программирования, обрабатываемый этим компилятором, будем называть *целевым языком*.

Документ состоит из следующих частей:

- Описание целевого модуля (см. раздел 2)
- Начало работы с проектом в инструменте OTK (см. раздел 3)
- Разработка модели (см. раздел 4)
- Разработка меппера (см. раздел 5)
- Разработка итератора (см. раздел 6)
  - Разработка итераторов модельных элементов (см. подраздел 6.1)
  - Задание в проекте итератора модельных структур (см. подраздел 6.2)
- Генерация тестов (см. раздел 7)
  - Конфигурирование генератора (см. подраздел 7.1)
  - Запуск генератора (см. подраздел 7.2)

2 Описание целевого модуля

Целевым модулем в рассматриваемом примере является модуль компилятора, который осуществляет некоторые преобразования арифметических выражений, содержащих суммирование целых чисел, значений целых переменных и подвыражений такого же вида.

3 Начало работы с проектом в инструменте OTK

Для начала работы над проектом, который будет содержать генератор тестов для целевого модуля, запустите инструмент OTK (см. документ "OTK. Установка и удаление").

Проект, описываемый в этом документе, входит в комплект примеров, поставляемых вместе с OTK, и расположен в подкаталоге `examples/expressions`, поэтому вы можете пропустить описание того, как создавать новый проект, а открыть уже существующий (для этого выберите пункт меню **File → Open Project**, войдите в упомянутый каталог

и откройте файл `Expressions.otk`) и приступить к разработке модели (см. раздел Разработка модели (см. раздел 4)).

Для создания нового проекта выберите пункт меню **File** → **New Project**, затем в появившемся окне **New project** (см. рисунок 1):

- введите идентификатор проекта (поле **Project identifier**),
- выберите базовый каталог проекта (поле **Project base directory**),
- введите имя файла настроек проекта (поле **Project file name**),
- выберите корневой java-пакет проекта (поле **Root package name**).

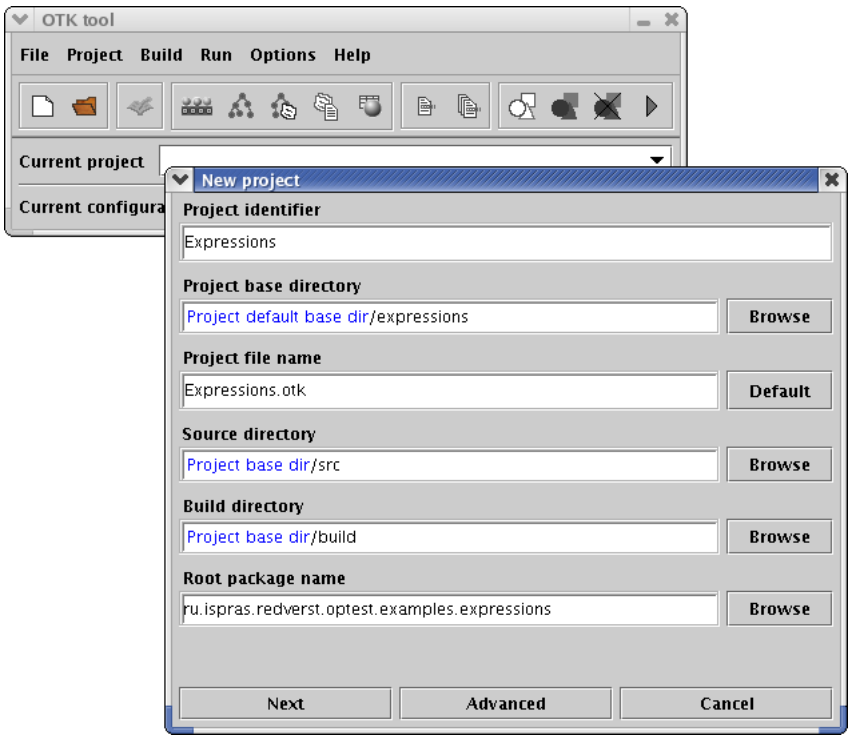

Рис. 1: Создание нового проекта.

Для имен каталогов исходных файлов проекта (поле **Source directory**) и результатов компиляции проекта (поле **Build directory**) инструмент предложит некоторые значения по умолчанию (это, соответственно, подкаталоги `src` и `build` базового каталога проекта), которые можно не изменять. Нажмите кнопку **Next**.

На втором шаге нужно указать имя модели (по умолчанию – **Model**), а также некоторую вспомогательную информацию. Здесь везде можно оставить значения, предлагаемые по умолчанию. Нажмите кнопку **Next**.

На третьем шаге также можно оставить все значения по умолчанию. Нажмите кнопку **Finish**.

В результате в базовом каталоге проекта будут автоматически созданы:

- файл настроек проекта (имеет расширение `.otk`),
- каталог исходных файлов проекта,
- каталог результатов компиляции проекта.

Кроме того, в каталоге исходных файлов проекта будет создан подкаталог для корневого java-пакета проекта, а в корневом java-пакете проекта будет создана заготовка файла для формального описания модели (файл с именем модели, введенном на втором шаге создания проекта, и с расширением `.tdl`).

4 Разработка модели

Технология тестирования UniTesK, поддерживаемая OTK, предполагает, что требования к структурам данных, обрабатываемых целевым модулем, записаны в четкой недвусмысленной форме. Такая форма представления требований называется *формальным описанием модели*.

В OTK модели формально описываются на специальном языке TDL (Tree Description Language).

У вас в проекте уже есть готовое формальное описание модели (файл `Model.tdl` в корневом java-пакете проекта), поэтому вы можете пропустить описание того, как разрабатывать модель, и приступить к разработке меппера (см. раздел Разработка меппера (см. раздел 5)).

Для того, чтобы создать формальное описание модели, необходимо проанализировать документацию на целевой модуль и построить *сводную диаграмму* модельных элементов.

В документации сказано: целевой модуль осуществляет некоторые преобразования арифметических выражений, содержащих суммирование целых чисел, значений целых переменных, а также подвыражений такого же вида.

Итак, каждое арифметическое выражение (далее будем говорить просто "выражение") это либо сложение, либо значение переменной, либо целое число (см. рисунок 2).

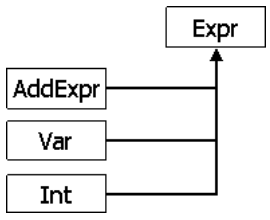

Рис. 2: Виды выражений.

Сложение содержит два операнда (левый и правый), каждый из которых является выражением (см. рисунок 3).

Кроме того, необходимо добавить головной модельный элемент, описывающий один тест (см. рисунок 4).

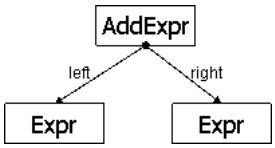

Рис. 3: Структура сложения.

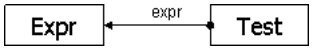

Рис. 4: Головной модельный элемент.

Итак, в результате получается следующая сводная диаграмма (см. рисунок 5):

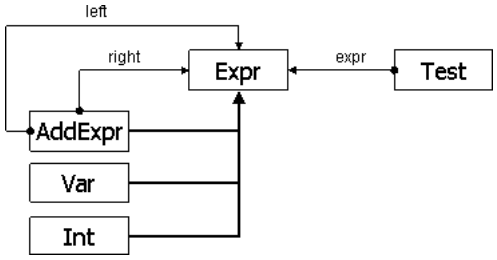

Рис. 5: Сводная диаграмма модели.

В соответствии с этой диаграммой разрабатывается формальное описание модели на языке TDL. Откройте файл `Model.tdl` в корневом пакете проекта. Для этого можно использовать пункт меню **File → Open File**; для настройки команды запуска внешнего текстового редактора используйте пункт меню **Options → File Editor**.

В этом файле находится автоматически сгенерированная заготовка описания модели:

```
[ translate.language = "java";
  visitor.name="TestVisitor";
]
tree ru.ispras.redverst.optest.examples.expressions.Model;

header
{
import ru.ispras.redverst.optest.OtkNode;
}
```

Здесь объявляется имя модели (помечено ключевым словом `tree`), а также указываются некоторые вспомогательные декларации (текст в квадратных скобках, а также блок, помеченный ключевым словом `header`).

После блока `header` нужно вставить описание модельных элементов:

```
node Test : <OtkNode>
{
    child Expr expr;
}

abstract node Expr : <OtkNode>
{}

node AddExpr : Expr
{
    child Expr left;
    child Expr right;
}

node Var : Expr
{}

node Int : Expr
{}
```

Каждое описание модельного элемента начинается ключевым словом `node`, перед которым в случае обобщенного модельного элемента может стоять модификатор `abstract`.

Имя обобщенного модельного элемента, конкретизацией которого является данный модельный элемент, указывается после двоеточия. Если модельный элемент не является ничьей конкретизацией, то после двоеточия нужно написать '<OtkNode>'.

Ссылки, ведущие на другие модельные элементы, описываются в виде полей, помеченных ключевым словом `child`.

Убедитесь, что разработанная модель успешно компилируется. Для этого выберите пункт меню **Build → Rebuild All**.

5 Разработка меппера

У вас в проекте есть формальное описание модельных элементов. Генератор в процессе работы будет строить из этих модельных элементов различные структуры. Для того, чтобы генератор строил тесты для компилятора, т.е. генерировал тексты программ на целевом языке, например, на языке программирования C, нужно предоставить генератору информацию о том, как модельные элементы преобразуются в текст на целевом языке. Для этой цели используется специальный компонент генератора, называемый *мепперо́м*.

Меппер оформляется в виде специального java-класса.

У вас в проекте уже есть готовый меппер `DefaultMapper`, расположенный в пакете `ru.ispras.redverst.optest.examples.expressions.mapper`, поэтому вы можете

пропустить описание того, как создавать новый меппер, и приступить к разработке итератора (см. раздел Разработка итератора (см. раздел 6)).

Для создания нового меппера выберите пункт меню **File** → **New File** → **New Mapper**, затем в появившемся окне **New mapper** нажмите кнопку **Add all non abstract nodes** для выбора всех неабстрактных модельных элементов (см. рисунок 6). После этого нажмите кнопку **Next**.

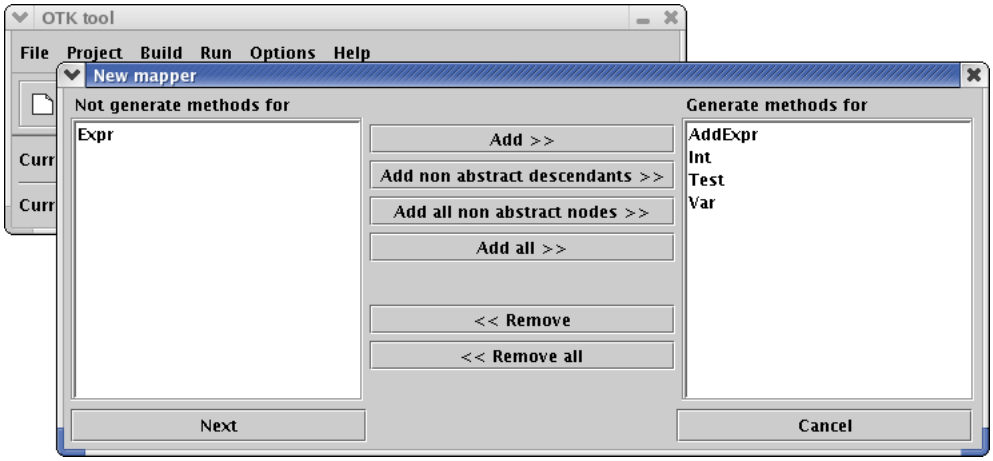

Рис. 6: Выбор модельных элементов для меппирования.

На втором шаге для нашего меппера ничего задавать не надо. Нажмите кнопку **Next**.

На третьем шаге нужно ввести имя пакета и имя класса для меппера (см. рисунок 7). После этого нажмите кнопку **Finish**.

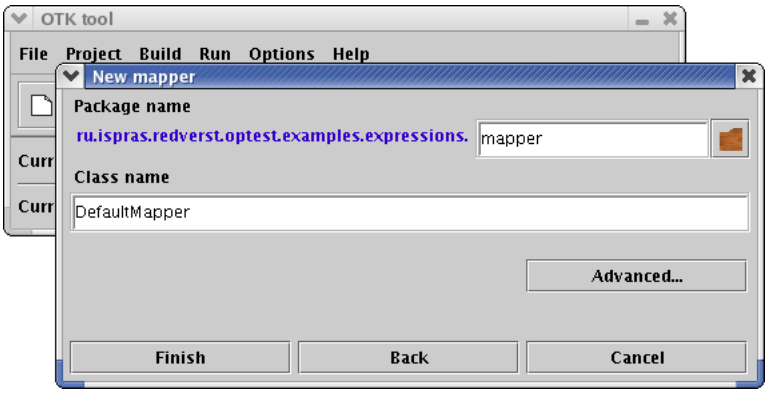

Рис. 7: Указание имени пакета и имени класса меппера.

В результате будет сгенерирован файл с заготовкой java-класса меппера. Если отбросить комментарии и использовать java-конструкцию `import` для возможности употреблять короткие имена классов, то эта заготовка выглядит так:

```
package ru.ispras.redverst.optest.examples.expressions.mapper;

import ru.ispras.redverst.optest.examples.expressions.Model;

public class DefaultMapper
    extends ru.ispras.redverst.optest.examples.expressions.EmptyMapper
{
    public void visitAddExpr( Model.AddExpr node ) {}
    public void visitInt    ( Model.Int     node ) {}
    public void visitTest   ( Model.Test    node ) {}
    public void visitVar    ( Model.Var     node ) {}
}
```

Этот класс наследуется от вспомогательного класса `EmptyMapper`, который генерируется автоматически. Для каждого модельного элемента, выбранного в ходе создания меппера, имеется метод, имя которого образуется путем приписывания приставки '`visit`' к идентификатору элемента. Каждый такой метод отвечает за генерацию текста для соответствующего модельного элемента.

Рассмотрим метод меппера для элемента **AddExpr**, моделирующего выражение-сложение:

```
public void visitAddExpr( Model.AddExpr node )
{
    txt( "( ${left} + ${right} )" );
}
```

Здесь для генерации текста используется библиотечный метод `txt`, предоставляемый родительским классом `EmptyMapper`. Этот метод принимает на вход один параметр – строку с генерируемым текстом. В этой строке могут содержаться ссылки на текст, получающийся в результате обработки меппером полей данного модельного элемента. Такие ссылки оформляются в виде специальной конструкции '`${...}`', где в фигурных скобках указывается имя поля.

Рассмотрим методы меппера для остальных модельных элементов:

```
public void visitInt( Model.Int node )
{
    txt( "3" );
}

public void visitTest( Model.Test node )
{
    txt( "int main() {" ); nl();
    incIndent();
    txt( "int n = 2;" ); nl();
    txt( "int res = ${expr};" ); nl();
    txt( "printf( \"%d\\n\", res );" ); nl();
    popIndent();
    txt( "}" ); nl();
}

public void visitVar( Model.Var node )
{
    txt( "n" );
}
```

Метод для элемента **Test** использует кроме метода `txt` еще три библиотечных метода, предназначенные для форматирования генерируемого текста:

- метод `nl` начинает новую строку текста,
- метод `incIndent` увеличивает величину отступа вывода текста,
- метод `popIndent` восстанавливает предыдущее значение величины отступа.

Метод для элемента **Test** генерирует текст C-функции main.

Метод для элемента **Var** генерирует взятие значения переменной n. Для того, чтобы обеспечить семантическую корректность всего теста, в текст функции main генерируется конструкция определения переменной n.

Для того, чтобы иметь возможность получить результат работы откомпилированной тестовой программы, в текст функции main генерируется конструкция вывода результата выражения, содержащегося в тесте, – вызов C-функции printf. Для того, чтобы обеспечить семантическую корректность теста, в начале тестовой программы должна иметься декларация функции printf. Для генерации текста в начало файла нужно определить в меппере метод `beginFile`:

```
public void beginFile()
{
    txt( "#include <stdio.h>" ); nl();
}
```

Убедитесь, что разработанный меппер успешно компилируется. Для этого выберите пункт меню **Build → Rebuild All**.

6 Разработка итератора

У вас в проекте есть формальное описание модельных элементов, а также меппер для преобразования структур, составленных из модельных элементов, в текст на целевом языке. Генератор в процессе работы будет строить различные структуры из модельных элементов. В общем случае из модельных элементов можно построить бесконечно много различных структур. Реальное множество тестов должно быть конечным. Чтобы получить конечное число тестов, требуется выделить конечное подмножество модельных структур и предоставить генератору информацию о том, как строить структуры из выделенного подмножества. Для этой цели используется специальный компонент генератора, называемый *итератором*.

Итератор модельных структур состоит из итераторов модельных элементов. Разработка итератора заключается в следующем:

- Разработка итераторов модельных элементов
- Задание в проекте итератора модельных структур

6.1 Разработка итераторов модельных элементов

Для каждого модельного элемента разрабатывается отдельный итератор, который оформляется в виде специального java-класса.

У вас в проекте уже есть готовые итераторы модельных элементов, которые расположены в пакете `ru.ispras.redverst.optest.examples.expressions.iterator`, поэтому вы можете пропустить описание того, как создавать новые итераторы модельных элементов, и приступить к заданию итератора модельных структур (см. раздел Задание в проекте итератора модельных структур (см. подраздел 6.2)).

Приступим к созданию итераторов модельных элементов. Сначала рассмотрим процесс создания итератора для модельных элементов, которые не имеют полей – в нашем случае это **Var** и **Int**. Начнем с итератора для элемента **Var**.

Для создания нового итератора выберите пункт меню **File → New File → New Iterator**, затем в появившемся окне **New iterator** выберите итератор для неабстрактного модельного элемента – выберите пункт **Non abstract node** (см. рисунок 8). После этого нажмите кнопку **Next**.

На втором шаге нужно из выпадающего выбрать модельный элемент, для которого создается итератор (см. рисунок 9). Для элемента **Var** здесь больше ничего указывать не надо. Нажмите кнопку **Next**.

На третьем шаге нужно ввести имя пакета и имя класса для итератора (см. рисунок 10). После этого нажмите кнопку **Finish**.

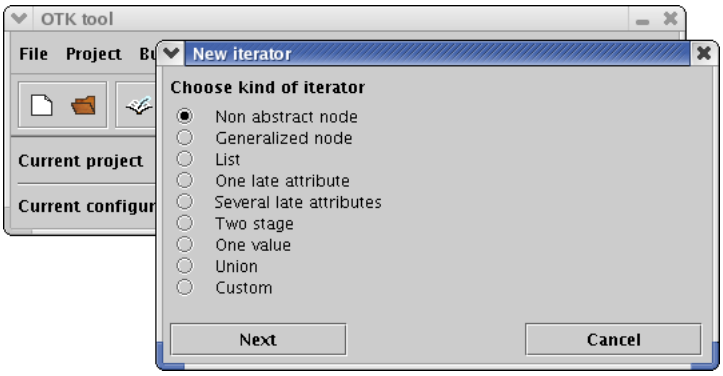

Рис. 8: Выбор вида итератора для неабстрактного модельного элемента.

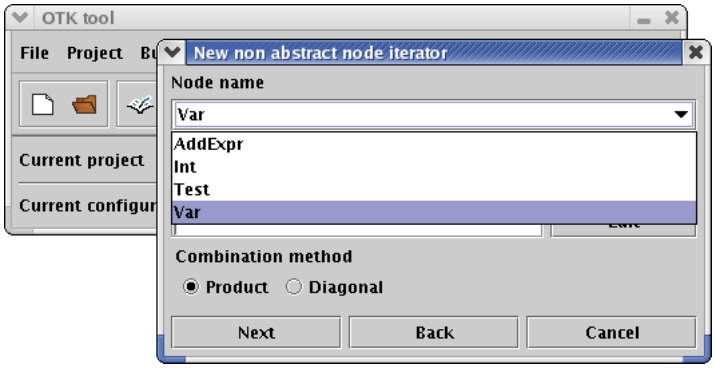

Рис. 9: Выбор модельного элемента, для которого создается итератор

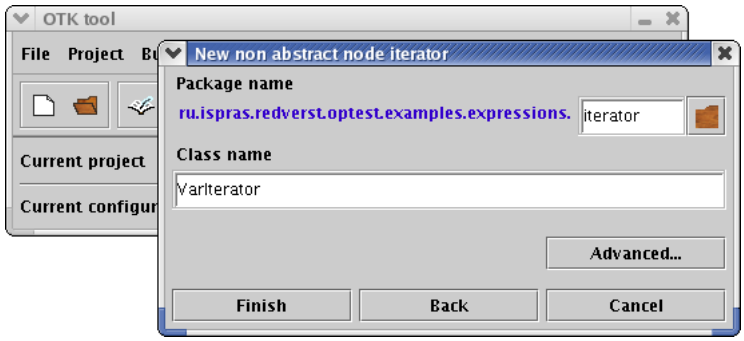

Рис. 10: Указание имени пакета и имени класса итератора.

В результате будет сгенерирован java-класс итератора для элемента **Var**. Аналогично создается java-класс итератора для элемента **Int**.

Теперь рассмотрим процесс создания итератора для модельных элементов, которые имеют поля. Начнем с итератора для элемента **AddExpr**. На втором шаге создания после выбора из списка модельного элемента, для которого создается итератор (т.е. элемента **AddExpr**), нужно указать итераторы для полей **left** и **right** этого элемента (см. рисунок 11). Оба эти поля имеют тип **Expr**, а значит для них нужно указать

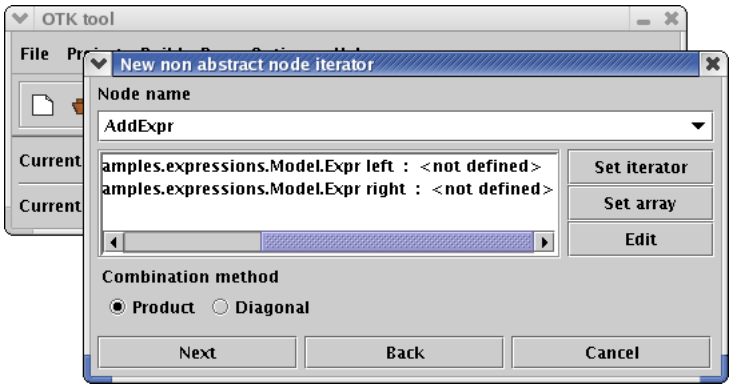

Рис. 11: Второй шаг создания итератора для модельного элемента, имеющего поля.

итератор модельных элементов типа **Expr**. Поскольку итератор для модельного элемента **Expr** еще не создан, можно выбрать итератор для какой-нибудь его конкретизации, например для элемента **Int**. Выделите поле **left**, нажмите кнопку **Set iterator** и в появившемся окне выберите в выпадающем списке java-класс `ru.ispras.redverst.optest.examples.expressions.iterator.IntIterator` (см. рисунок 12). Аналогично укажите итератор для поля **right**. После этого нажмите кнопку

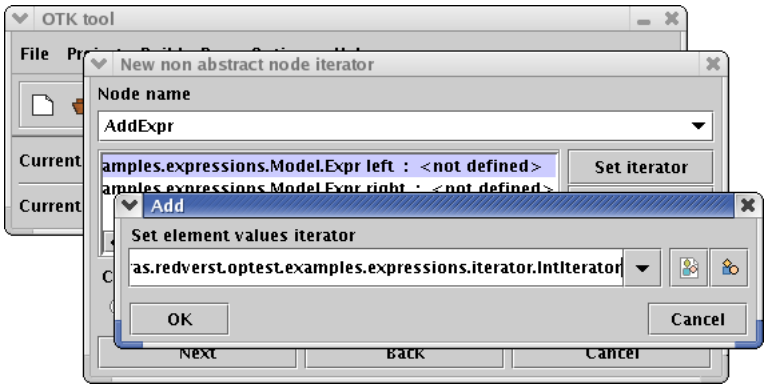

Рис. 12: Выбор итератора для поля.

Next и введите имя пакета и имя класса для итератора. После этого нажмите кнопку **Finish**.

В результате будет сгенерирован java-класс итератора для элемента **AddExpr**. Поскольку для полей этого элемента был указан итератор элемента **Int**, то получившийся итератор будет строить модельные структуры, соответствующие сложению целых чисел. Для того, чтобы строились модельные структуры, соответствующие сложению различных выражений, нужно, чтобы для полей этого элемента был указан итератор эелемента **Expr**. Прежде всего нужно создать такой итератор.

Элемент **Expr** является обобщенным модельным элементом. Для того, чтобы создать для него новый итератор, выберите пункт меню **File** → **New File** → **New Iterator**, затем в появившемся окне **New iterator** выберите итератор для обобщенного

модельного элемента – выберите пункт **Generalized node** (см. рисунок 13). После этого нажмите кнопку **Next**.

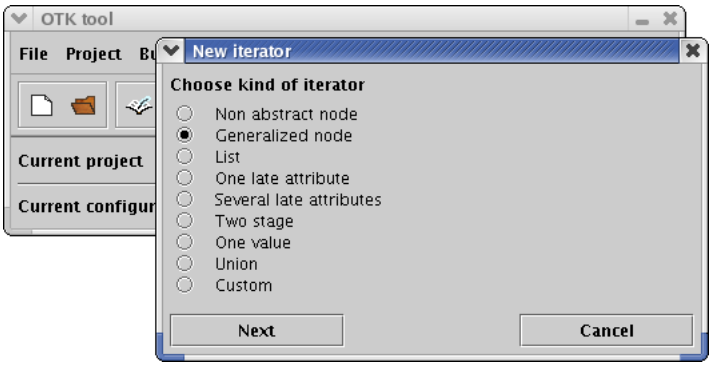

Рис. 13: Выбор вида итератора для обобщенного модельного элемента.

На втором шаге надо выбрать из списка модельный элемент, для которого создается итератор – элемента **Expr**. После этого в окне появятся все элементы-конкретизации этого обобщенного элемента (см. рисунок 14). Для каждой конкретизации нужно задать

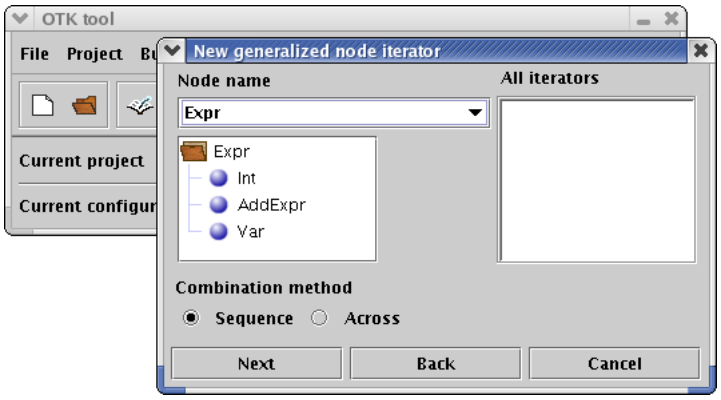

Рис. 14: Окно задания итераторов элементов-конкретизаций обобщенного модельного элемента.

соответствующий итератор. Для этого выделите элемент-конкретизацию и нажмите появившуюся кнопку **Add**, после чего в появившемся окне выберите из выпадающего списка итератор и нажмите кнопку **OK** (см. рисунок 15). После установки итераторов всех элементов-конкретизаций (см. рисунок 16) нажмите кнопку **Next**.

На третьем шаге введите имя пакета и имя класса для итератора. После этого нажмите кнопку **Finish**.

В результате будет сгенерирован java-класс итератора для элемента **Expr**.

Теперь вернемся к итератору для элемента **AddExpr**. Рассмотрим java-код конструктора этого итератора. Этот конструктор содержит `try`-блок, в котором производится инициализация объекта `main_iterator`. Если отбросить комментарии и

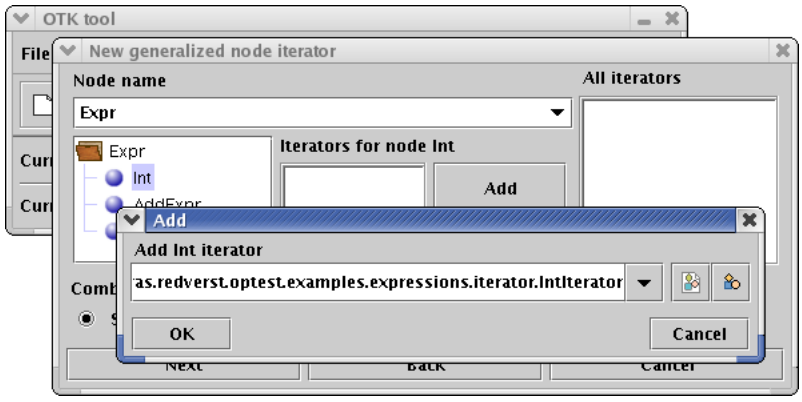

Рис. 15: Задание итератора элемента-конкретизации обобщенного модельного элемента.

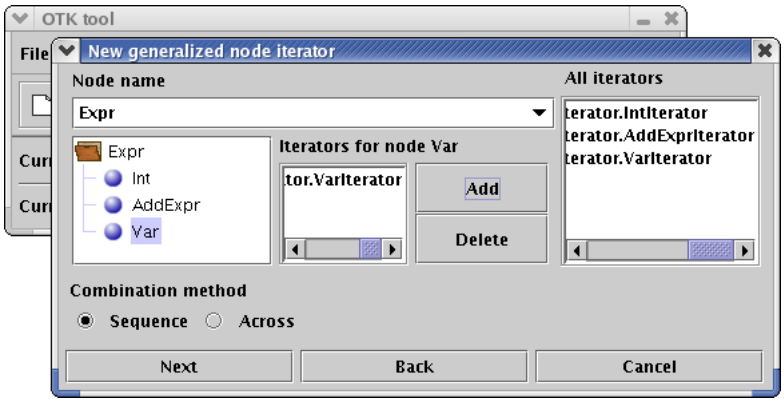

Рис. 16: Заданные итераторы элементов-конкретизаций обобщенного модельного элемента.

использовать java-конструкцию `import` для возможности употреблять короткие имена классов, то java-код инициализации объекта `main_iterator` выглядит так:

```
main_iterator = IteratorFactory.createDetailsIterator
    ( OTK.getIteratorProperty( "iterator.AddExprIterator.combination" )
    , new ValueIterator[]
        { /* left */  new IntIterator()
        , /* right */ new IntIterator()
        }
    );
```

Этот код нужно доработать, чтобы строились модельные структуры, соответствующие сложению различных выражений, а не только целых чисел. Для этого нужно указать для его полей итератор эелемента **Expr**.

Однако, если заменить в процитированном коде конструкцию `new IntIterator()` на конструкцию `new ExprIterator()`, то во время работы генератора возникнет бесконечная рекурсия: для создания итератора элемента **AddExpr** нужно создать

итератор элемента **Expr**, для создания которого в свою очередь нужно создать другой итератор элемента **AddExpr**, и т.д. Для решения этой проблемы итератором предоставляется метод **getDepth**. На некритической глубине, т.е. когда getDepth() $>$ 0, нужно вызывать конструкцию `new ExprIterator()`, а на критической глубине вызывать конструкцию `new IntIterator()`. Доработанный таким образом java-код инициализации объекта `main_iterator` выглядит так:

```
main_iterator = IteratorFactory.createDetailsIterator
    ( OTK.getIteratorProperty( "iterator.AddExprIterator.combination" )
    , new ValueIterator[]
        { getDepth() > 0 ?
          (ValueIterator)new ExprIterator() :
          (ValueIterator)new IntIterator()
        , getDepth() > 0 ?
          (ValueIterator)new ExprIterator() :
          (ValueIterator)new IntIterator()
        }
    );
```

Итератор для элемента **AddExpr** готов.

Итератор для модельного элемента **Test** создается аналогично итератору для **Var** с указанием на втором шаге для поля **expr** итератора элемента **Expr**.

Убедитесь, что разработанные итераторы успешно компилируется. Для этого выберите пункт меню **Build** → **Rebuild All**.

6.2 Задание в проекте итератора модельных структур

После создания итераторов для модельных элементов нужно задать в проекте итератор модельных структур, который будет использоваться при работе генератора.

У вас в проекте итератор модельных структур уже задан, поэтому вы можете пропустить описание того, как задавать итератор модельных структур, и приступить к генерации тестов (см. раздел Генерация тестов (см. раздел 7)).

Для того, чтобы задать в проекте итератор модельных структур, выберите пункт меню **Project** → **Default iterator**, в появившемся окне введите имя пакета и имя класса для итератора модельных структур, а также в поле **Tree iterator class** выберите из выпадающего списка имя итератора головного модельного элемента, т.е. элемента **Test** (см. рисунок 17). После этого нажмине кнопку **OK**.

Убедитесь, что проект успешно компилируется. Для этого выберите пункт меню **Build** → **Rebuild All**.

7 Генерация тестов

У вас в проекте есть формальное описание модельных элементов, меппер для

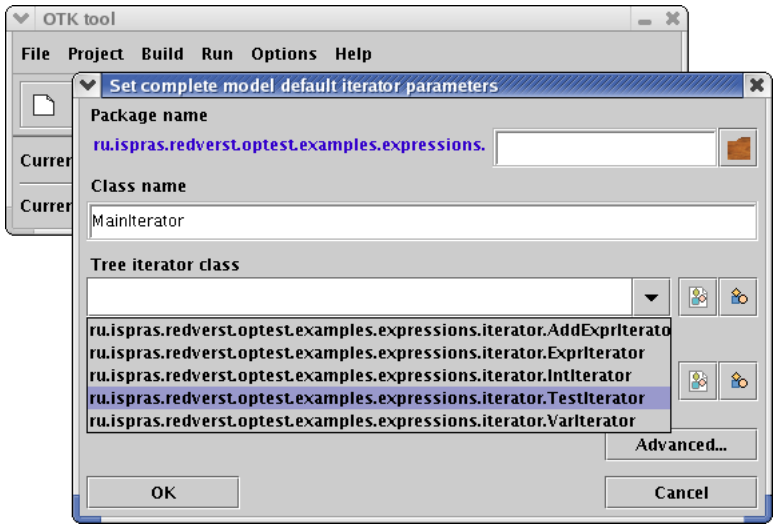

Рис. 17: Задание итератора модельных структур.

преобразования структур, составленных из модельных элементов, в текст на целевом языке, а также итератор модельных структур.

Перед запуском генератора проект нужно откомпилировать. Для этого выберите пункт меню **Build → Rebuild All**.

Генерация тестов заключается в следующем:

- Конфигурирование генератора (см. подраздел 7.1)
- Запуск генератора (см. подраздел 7.2)

7.1 Конфигурирование генератора

Генератор в процессе работы будет строить различные структуры из модельных элементов и преобразовывать их в текст на целевом языке. Перед запуском генератор нужно сконфигурировать: указать итератор модельных структур и меппер.

У вас в проекте уже имеется готовая конфигурация, поэтому вы можете пропустить описание того, как создавать новую конфигурацию, и приступить к запуску генератора (см. раздел Запуск генератора (см. подраздел 7.2)).

Для создания новой конфигурации выберите пункт меню **Run → New Configuration**, затем в появившемся окне **New configuration** введите идентификатор конфигурации (см. рисунок 18). После этого нажмите кнопку **Next**. Появится окно конфигурации. Разработанные в вашем проекте компоненты генератора нужно указать в конфигурации в виде подключаемых модулей (*plugin*). Прежде всего нужно указать итератор модельных структур. Для этого введите имя подключаемого модуля итератора в поле **Test iterator plugin** после чего нажмите кнопку **Default**, расположенную справа от этого поля. В результате в окне конфигурации появится папка с информацией о подключаемом модуле для итератора (см. рисунок 19).

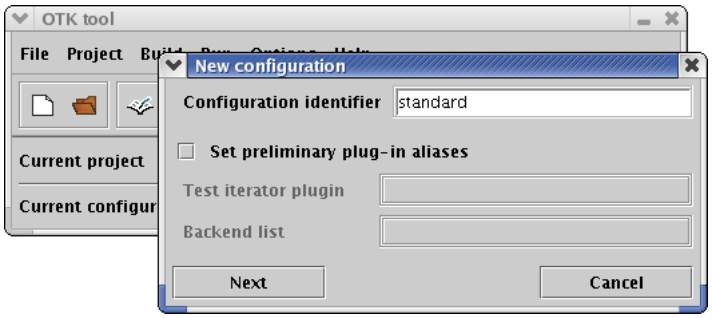

Рис. 18: Создание новой конфигурации.

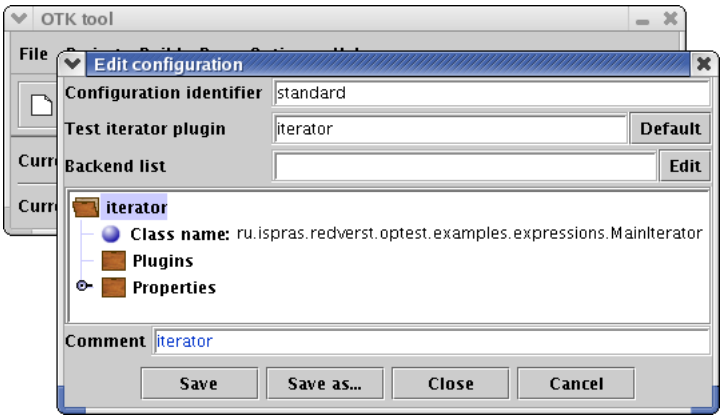

Рис. 19: Задание в конфигурации итератора модельных структур.

Для указания меппера нужно открыть окно редактирования списка обработчиков модельных структур. Нажмите кнопку **Edit**, расположенную справа от поля **Backend list**. В появившемся окне в правой колонке кнопок нажмите шестую сверху кнопку, для которой появляется всплывающая подсказка '**Insert new standard printer before selected item**' (см. рисунок 20). В новом появившемся окне введите имя подключаемого модуля принтера в поле **Printer plugin name**, выберите java-класс меппера из выпадающего списка в поле **Mapper class name**, а также введите имя подключаемого модуля меппера в поле **Mapper plugin name** (см. рисунок 21). После этого нажмите кнопку **OK**. В окне редактирования списка обработчиков модельных структур также нажмите кнопку **OK**.

Теперь в конфигурации указаны все необходимые подключаемые модули. Информация о них отображается в виде папок к окне конфигурации (см. рисунок 22).

Теперь нужно сконфигурировать принтер. В окне конфигурации в папке для подключаемого модуля **printer** откройте папку свойств **Properties**. Принтер имеет четыре свойства (см. рисунок 23):

- `output.dir` – каталог для вывода файлов с тестами;
- `file.prename` – начало имен генерируемых файлов;

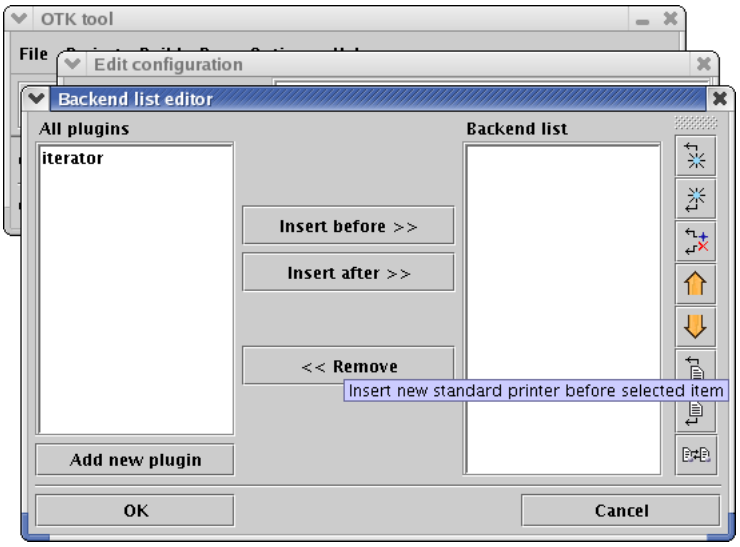

Рис. 20: Вставка принтера в список обработчиков модельных структур.

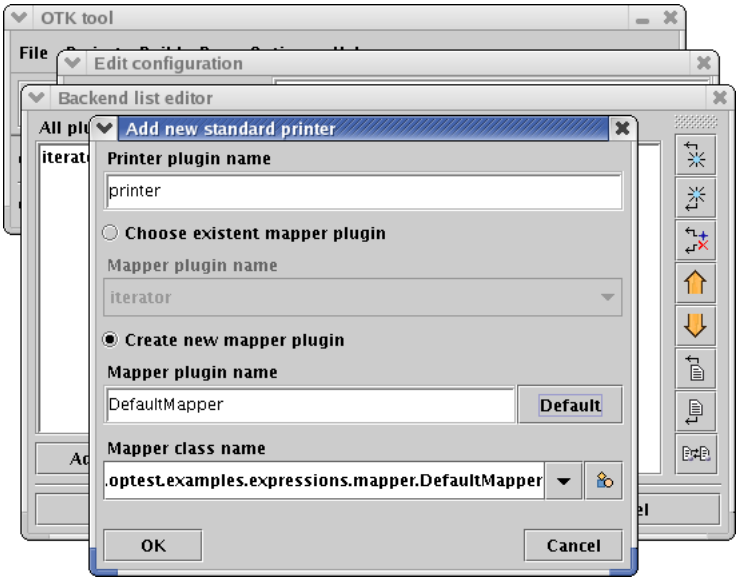

Рис. 21: Указание меппера в принтере.

- `file.postname` – конец имен генерируемых файлов;
- `file.size` – количество тестов, генерируемых в один файл.

Выберите каталог для вывода файлов с тестами. Для этого щелкните два раза левой кнопкой мыши (нужен не "двойной щелчок"кнопки мыши, а именно нажатие два раза "медленно", т.е. с некоторым перерывом между нажатиями) по строке свойства `output.dir`, а затем нажмите появившуюся справа в этой строке кнопку обзора каталогов вашей файловой системы.

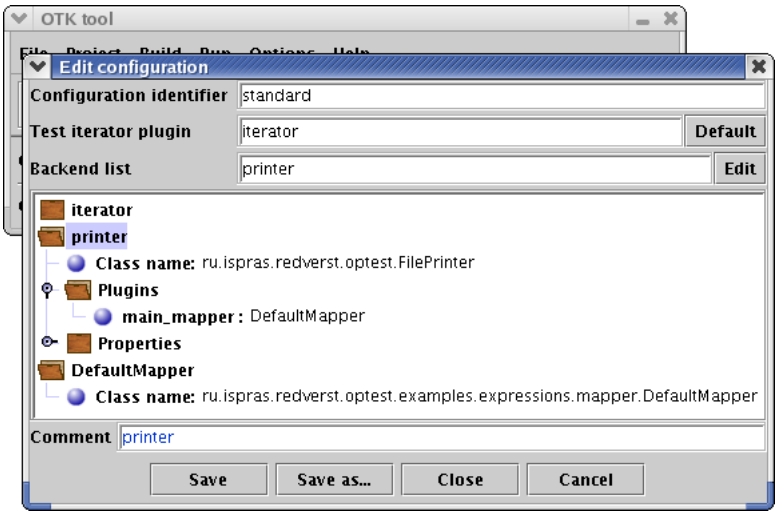

Рис. 22: Окно конфигурации после указания всех подключаемых модулей.

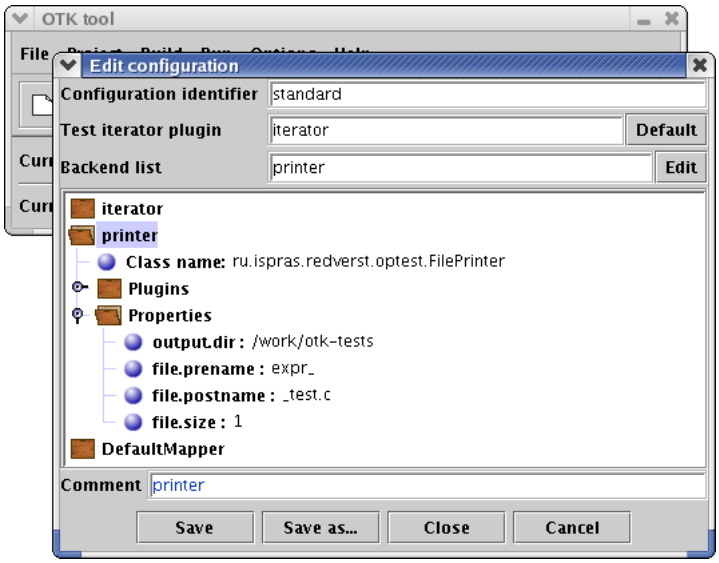

Рис. 23: Конфигурирование принтера.

Введите начало имен генерируемых файлов в поле `file.prename`.
Введите конец имен генерируемых файлов в поле `file.postname`.
Установите количество тестов, генерируемых в один файл, равным 1.
Принтер сконфигурирован (см. рисунок 23).
Теперь конфигурация полностью готова. Нажмите в окне редактирования конфигурации кнопку **Save**, затем нажмите кнопку **Close**.

7.2 Запуск генератора

Перед запуском генератора убедитесь, что в конфигурации указан корректный

каталог для вывода файлов с тестами. Для этого выберите пункт меню **Run** → **Edit Configuration**, затем в появившемся окне **Edit configuration** откройте папку **printer**, в ней откройте подпапку **Properties** (см. рисунок 23), щелкните два раза левой кнопкой мыши (нужен не "двойной щелчок"кнопки мыши, а именно нажатие два раза "медленно", т.е. с некоторым перерывом между нажатиями) по строке свойства `output.dir`, а затем нажмите появившуюся справа в этой строке кнопку обзора каталогов вашей файловой системы и выберите подходящий каталог для тестов. После этого нажмите в окне редактирования конфигурации кнопку **Save**, а затем нажмите кнопку **Close**.

Для запуска генератора выберите пункт меню **Run** → **Start Generator**. В появившемся окне **Generate** будет отображаться ход генерации тестов – каждой точке соответствует один сгенерированный файл. При успешном завершении генерации будет выведено количество сгенерированных тестов (см. рисунок 24). Нажмите кнопку **Close**.

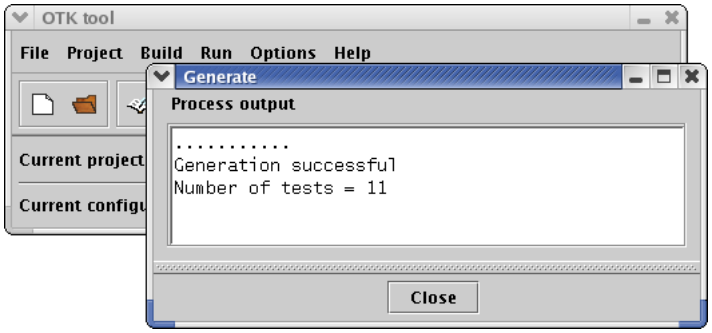

Рис. 24: Успешное завершение генерации тестов.

По умолчанию генератором будут созданы тесты, у которых глубина вложенности подвыражений не превышает 2. Для того, чтобы сгенерировать тесты с большей глубиной вложенности подвыражений, нужно указать значение этой глубины в конфигурации. Для этого выберите пункт меню **Run** → **Edit Configuration**, затем в появившемся окне **Edit configuration** откройте папку **iterator**, в ней откройте подпапку **Properties**, щелкните два раза левой кнопкой мыши по строке свойства **AddExpr.depth**, поставьте галочку в квадратике слева в этой строке и измените значение свойства (см. рисунок 25). После этого нажмите в окне редактирования конфигурации кнопку **Save**, а затем нажмите кнопку **Close**. После этого запустите генератор, для чего выберите пункт меню **Run** → **Start Generator**.

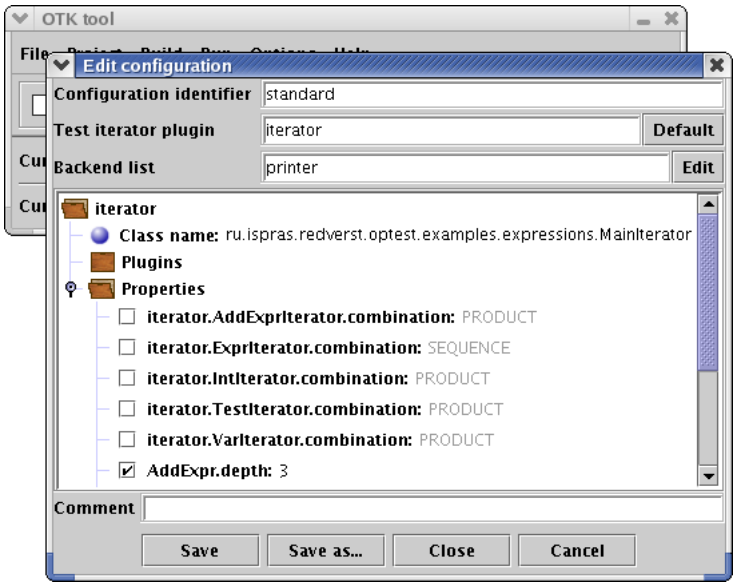

Рис. 25: Изменение значения свойства итератора модельных структур.