Математические структуры и моделирование
2014. № 4(32). С. 198–205

УДК 004.434

ПРЕДМЕТНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ТЕСТИРОВАНИЯ ЭЛЕКТРОННЫХ УСТРОЙСТВ

А.Ю. Воробьёв
аспирант, e-mail: flyenj@gmail.com

Омский государственный университет им. Ф.М. Достоевского

Аннотация. В данной статье обсуждается возможность применения предметно-ориентированного языка программирования для быстрой и эффективной разработки программного обеспечения для автоматизированного контроля параметров электронных устройств, например, радиоприёмных устройств. Предполагается, что для такой проверки необходимо управлять несколькими устройствами: анализаторами, генераторами и т. д. Рассмотрено применение гипотетического внутреннего DSL (Domain Specific Language, DSL). Внутренний DSL — это один из способов использования языка общего назначения. Код во внутреннем DSL — это валидный код в языке общего назначения, но использующий только некоторое подмножество возможностей [1]. Предлагается использовать объектно-функциональный язык общего назначения Scala [2]. Благодаря использованию такого гипотетического DSL при разработке ПО для автоматизированного контроля параметров изделий разработчик может сосредоточиться только на бизнес-логике приложения, что позволяет разрабатывать ПО значительно быстрее и эффективнее. Такой результат достигается за счёт сокращения количества кода, необходимого для написания ПО, и уменьшения сложности разработки благодаря тому, что DSL внутренний и содержит готовые абстракции для решения часто встречающихся задач.

Ключевые слова: программирование, предметно-ориентированные языки, технологическое ПО, автоматизация, автоматизированный контроль параметров.

В настоящее время в сообществе разработчиков программного обеспечения (ПО) возрастает интерес к реализации предметно-ориентированных языков (Domain Specific Language, DSL) программирования для решения узкоспециализированных задач. DSL — это язык программирования с ограниченной выразительностью, специализированный для конкретной области применения [1]. Преимущества использования DSL заключаются в существенном снижении денежных и временных затрат на разработку. В некоторых случаях, благодаря снижению сложности, разрабатывать ПО могут люди, поверхностно знающие программирование.

На предприятиях-производителях различной электроники и радиоаппаратуры разработана система качества выпускаемой продукции. Контроль могут проходить все изделия и комплектующие, поступающие в производство. Для снижения себестоимости продукции, а также сокращения времени на поверку, предприятия данного типа самостоятельно разрабатывают и применяют ПО для автоматизированного контроля параметров выпускаемых изделий. В связи с активной разработкой новых изделий и модификацией существующих остро стоит задача существенного повышения эффективности разработки ПО технической диагностики изделий.

Ниже будет рассмотрено применение гипотетического внутреннего DSL. Внутренний DSL — это один из способов использования языка общего назначения. Код во внутреннем DSL — это валидный код в языке общего назначения, но использующий только некоторое подмножество возможностей [1]. В данном случае будет использован объектно-функциональный язык общего назначения Scala [2]. К преимуществам этого языка для создания DSL относятся [2]:

- строгая статическая типизация, благодаря чему DSL будет типобезопасен;
- функции высшего порядка, которые могут быть присвоены, переданы как аргумент и возвращены;
- выведение типов минимизирует необходимость для явного указания типов;
- краткий синтаксис;
- неявные конвертации (implicits), позволяющие неявно передавать аргументы функции;
- макросы [3].

Благодаря таким возможностям язык Scala хорошо подходит для решения поставленной задачи.

Ниже будет приведён пример гипотетического DSL на Scala для разработки ПО автоматизированного контроля параметров радиоприёмного изделия. Стоит отметить, что такой DSL подойдёт для аналогичных изделий и рабочих мест.

Рассмотрим радиоприёмное изделие с выходом сигналов в аналоговой форме. Для проверки такого устройства необходимо анализировать сигнал анализатором спектра, а также генерировать на антенный вход изделия чистый сигнал генератором сигнала. Стоит отметить, что современные анализаторы спектра и генераторы сигнала обычно имеют интерфейсы для удалённого управления по Ethernet или USB. Такое рабочее место изображено на рис. 1.

При разработке ПО для контроля параметров такого изделия необходимо, прежде всего, создать классы для взаимодействия с устройствами. Исходный код таких классов можно разделить на две составляющие. Первая содержит команды и запросы из протокола сопряжения устройства, логику их отправки и обработку ошибок. А вторая — код, не относящийся к бизнес-логике, например, открытие и закрытие сокетов, процедура соединения, проверка доступности устройства и т.д. Код из второй составляющей обычно частично переносят в отдельную программную библиотеку. Время разработки при этом сокращается, но не избавляет разработчика от необходимости реализовывать методы для каждой команды, а также знать и уметь применять API этой библиотеки.

Рис. 1. Рабочее место для проверки радиоприёмного изделия

Эффективнее будет подход, при котором классы для управления устройством генерируются по протоколу сопряжения этого устройства. Рассмотрим пример на DSL:

```
class AgilentGeneratorDefinition
     extends RemoteDeviceDefinition{
 name = "Agilent"
 // тип протокола
 protocolType = tcp
 // адрес устройства по умолчанию
 defaultAddress = "192.168.0.5"

 // Список команд (не возвращают значения)
 def commands = Seq(
  TcpDeviceCommand(
   "setFrequency",
   "setFreq x",
   "On error message"
  ).withArgs(classOf[Int])).withArgsMapper{
   case arg: Int => String.valueOf(arg)
  }
 )

 // Список запросов (возвращают значение)
 def requests = Seq(
  TcpDeviceRequest(
   "getModuleStatus",
   "getModStat x",
   "On error message"
```

```
  ).withArgs(classOf[String]).withResultMapper{
   // Преобразование результата
   case s: String => new ModuleStatus(s)
  }
 )
}
```

Класс "AgilentGeneratorDefinition" представляет собой определение протокола устройства, а также задаёт некоторые настройки по умолчанию. Он может быть расширен с добавлением перехватов и специфичных настроек для определённых протоколов. Предполагается, что такой класс будет обработан с помощью техники Reflection [4], после чего будет сгенерирован класс управления устройством. Структура типичной команды такова: название метода в сгенерированном классе, название команды из протокола сопряжения устройства с указанием места, в который подставляется аргумент, сообщение об ошибке, типы аргументов и функция конвертации аргумента. Если будет указано два аргумента, например x и y, то для генерации класса управления будет необходимо указать оба типа. Для запроса можно указать алгоритм преобразования результата с помощью метода "withResultMapper" и техники сопоставления с образцом (pattern matching) [2]. Сгенерированные классы для управления устройствами будут объявлены с помощью ключевого слова "object", что позволит обращаться к ним из любой точки ПО и передавать как аргумент. При этом они будут существовать в единственном экземпляре.

Такой подход эффективен тем, что разработчик указывает только те настройки и параметры, которые относятся к бизнес-логике. Нет необходимости описывать реализацию взаимодействия через TCP или UDP сокеты.

Кроме управления устройствами необходимо описывать испытания. В испытании обычно присутствуют четыре компонента:

- алгоритм испытания;
- таблица результатов, которая будет отображаться в интерфейсе пользователя;
- отчет с результатами испытания в формате PDF;
- информация об испытании (цель испытания, методы контроля, нормы и т.д.).

На DSL описание испытания выглядит следующим образом:

```
class Sensetivity extends
     Challenge[List[Frequency], List[BigDecimal]]{
 requirements = List(
  new DeviceRequirement(AgilentSpectrumAnalyzer),
  new DeviceRequirement(AgilentGenerator)
 )
```

```
 // Исходные данные
 data = List(4000 kHz,  5000 kHz,  7000 kHz)

 override def algorithm {
  results = List()
  // Алгоритм испытания
  for (Frequency f : data) {
   if (interrupted) return
   AgilentGeneartor.setFrequency(f + 1 kHz)
   Receiver.getChannelByNumber(1).setFrequency(f)
   results :+ new BigDecimal(SpectrumAnalyzer.measureSnr())
 }
}

def report() {
 // Генерирование отчета
}

def view: ChallengeTable = {
 // Генерирование таблицы результатов
}

def info: ChallengeInfo = {
 // Информация об испытании
}
```

Класс "Challenge", означающий испытание, параметризованный. Первый тип указывается для исходных данных испытания, например, проверяемых частот. Второй — для результатов. Алгоритм генерации отчета располагается в методе "report". Для создания PDF документов можно использовать программную библиотеку iText [5]. Генерирование таблицы результатов осуществляется через абстракцию "ChallengeTable", скрывая тонкости реализации. Благодаря неявным преобразованиям возможно использование конструкции "4000 kHz". Информацию об испытании можно получить с помощью функции "info".

Интерфейс пользователя предполагается создавать с помощью технологии JavaFX 8 [6]. Автоматическая генерация интерфейса в данном случае нецелесообразна, так как могут применяться разные устройства с разными типами команд. Тогда бы пришлось описывать логику генерирования для каждого типа команды. Технология JavaFX 8 способствует быстрой разработке интерфейса пользователя, так как содержит развитые средства визуального конструирования интерфейсов.

Входная точка программы выглядит следующим образом:

```
class MainApp extends MainApplication {
```

```
 // Регистрация испытаний
 challenges += Seq(new Sensetivity,
      new Selectivity, new Intermodulation)

 // Панель для управления устройствами
 deviceControlPanel = new DeviceControlPanel

 // Верхнее меню
 menuBar = new MyMenuBar

 // Дополнительная панель
 additionalPanel = new SpectrumAnalyzer
}
```

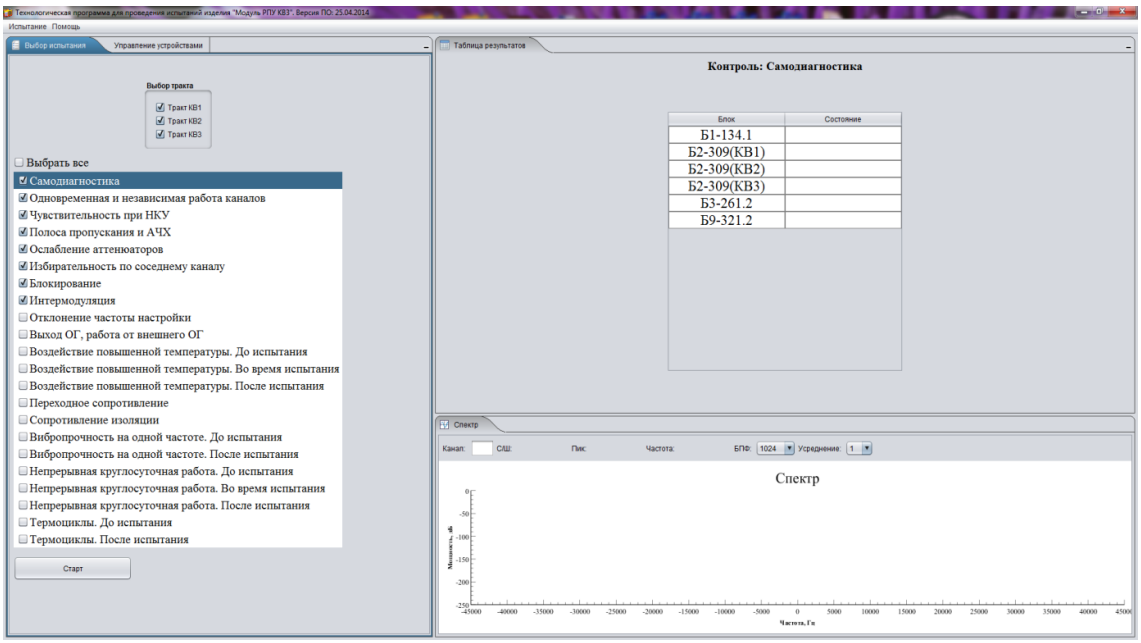

Рис. 2. Пример панели испытаний

Класс "MainApplication" содержит метод "Application.launch", который запускает программу. Здесь есть возможность настраивать интерфейс пользователя и добавлять на него пользовательские элементы, например, верхнее меню, панель для выбора проверяемого тракта и программный анализатор спектра.

Так как классы для управления устройствами объявлены с помощью ключевого слова "object", то нет необходимости регистрировать их. Обращение к ним возможно из любой точки программы. Испытания регистрируются в специальной коллекции, из которой они затем будут доставаться и преобразовываться к нужному виду. Логика работы с испытаниями, а именно запуск, остановка, генерация отчёта и т.д., располагается в классе "MainApplication". Пример возможного интерфейса пользователя изображён на рис. 2 и 3.

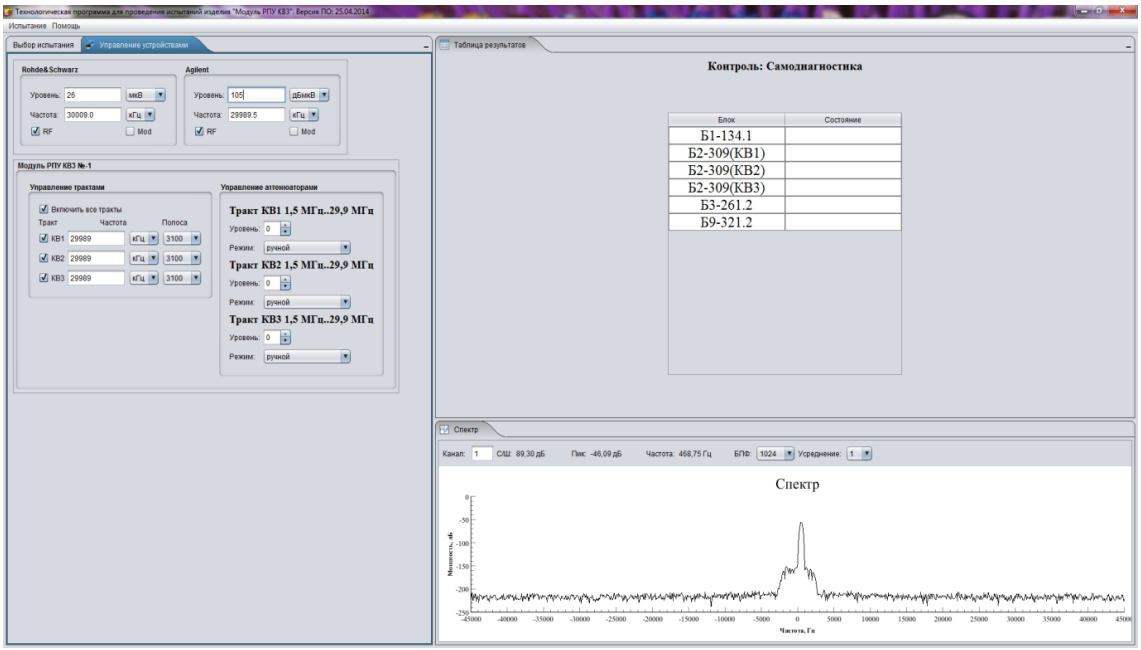

Рис. 3. Пример управления устройствами

Такой интерфейс удобен. В нем можно выделить два режима работы: ручной и автоматический. На вкладке "Выбор испытания" размещён список испытаний для автоматической диагностики изделия. На вкладке "Управление устройствами" располагается интерфейс для ручного управления устройствами. Размеры каждой вкладки могут быть изменены по усмотрению пользователя.

Благодаря использованию такого гипотетического DSL при разработке ПО для автоматизированного контроля параметров изделий разработчик может сосредоточиться только на бизнес-логике приложения, что позволяет разрабатывать ПО значительно быстрее и эффективнее. Такой результат достигается за счёт сокращения количества кода, необходимого для написания ПО, и уменьшения сложности разработки благодаря тому, что DSL внутренний и содержит готовые абстракции для решения часто встречающихся задач.

Литература

1. Fowler M. Domain-Specific Languages. Addison-Wesley Professional; 1 edition, 2010.
2. Odersky M., Spoon L., Venners B. Programming in Scala: A Comprehensive Step-by-Step Guide. Artima Inc; 2 edition, 2011.
3. Scala Macros. URL: `http://scalamacros.org/` (дата обращения 26.10.2014).
4. Reflection. URL: `http://docs.scala-lang.org/overviews/reflection/overview.html` (дата обращения 26.10.2014).
5. iText. URL: `http://itextpdf.com/` (дата обращения 26.10.2014).
6. Java FX 8. URL: `http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm` (дата обращения 26.10.2014).

DOMAIN-SPECIFIC LANGUAGE FOR DEVELOPING SOFTWARE FOR AUTOMATED TESTING OF ELECTRONIC DEVICES

A.Y. Vorobyev
postgraduate student, e-mail: flyenj@gmail.com

Omsk State University n.a. F.M. Dostoevskiy

Abstract. This article discusses the possibility of use of domain-specific programming language for rapid and effective develop of software for the automated testing of physical devices, such as radio receivers. It is assumed that for such a test is necessary to control multiple devices, for example analyzers, generators, etc.

Keywords: domain-specific languages, technological software, automation, automated control of parameters.