Introducing Scala

Developing a new Scala DSL for Apache Camel
Goals

• Goals
  - Introduce a few basic concepts/syntax of Scala
  - How to use these Scala techniques for building a Scala DSL (using Apache Camel as an example)
Planning

• Introduction

• Scala for DSL building
  - Implicit conversion
  - Passing functions as parameters
  - By-name parameters and currying
  - Caveats

• Scala tooling
  - Maven plugin
  - Eclipse plugin
Planning

• Introduction
• Scala for DSL building
  – Implicit conversion
  – Passing functions as parameters
  – By-name parameters and currying
  – Caveats
• Scala tooling
  – Maven plugin
  – Eclipse plugin
Introduction

• Who am I?
  – Gert Vanthienen (gert@anova.be)
  – Independent consultant
    • Open-source (Java/J2EE) technology
    • Legacy integration (System i aka AS/400)
  – Open-source
    • Apache ServiceMix committer / PMC member
    • Contributor to Apache Camel
Introduction

• What is Apache Camel?
  – Spring-based Integration Framework
  – Implements enterprise integration patterns
  – Configured through
    • Java DSL (fluent API)
    • Spring XML configuration file
  – URIs to work with other transports/protocols
  – Routing/mediation for ServiceMix, ActiveMQ, CXF, ... 
  – Check out Bruce Snyder's presentation on Friday!!
Introduction

• Just a small example of the Java DSL

```java
public class FleetRouteBuilder extends RouteBuilder {

    public void configure() throws Exception {
        from("ftp://server.local:10021/traces/out")
            .to("ftp://server.remote/folder/to/upload")
            .splitter(xpath("/traces/trace"))
            .to("activemq:MY.TRACE.QUEUE")
            .filter(xpath("/trace/@type == 'XYZ'"))
            .to("wmq:QLIN.TRACE.QUEUE");
    }
}
```
Introduction

• What is Scala?
  - Sca(lable) la(nguage)
  - Multi-paradigm:
    • Object-oriented: classes, polymorphism, inheritance, ..
    • Functional: function = value, pattern matching, ...
  - Static typing, using type inference
  - Interoperates with JRE (and .NET CLR)
    • Scala code compiles into Java bytecode
    • You can call Java code from Scala (and vica versa)
Introduction

- A simple Scala class example

```scala
class Person(name: String, age: Int) {
  def eat(food: String) {
    println("Eating " + food + " now")
  }

  def isToddler = age > 0 && age < 3

  override def toString() = "Person[" + name + "]"
}
```
Planning

• Introduction

• Scala language
  – Implicit conversion
  – Passing functions as parameters
  – By-name parameters and currying
  – Caveats

• Scala tooling
  – Maven plugin
  – Eclipse plugin
Simple route example

- Example of the simplest route possible in Java
  Just receive a message and forward it

```java
public class MyRouteBuilder extends RouteBuilder {
    
    public void configure() throws Exception {
        from("direct:a").to("mock:a");
        from("direct:b").to("mock:b");
    }
}
```
Simple route example

• In the Scala DSL it looks like this...

```scala
class MyRouteBuilder extends RouteBuilder {

  "direct:a" to "mock:a"
  "direct:b" --> "mock:b"

}
```

• ... using these language features
  - constructor statements go in the class body
  - no need for parentheses, dots and semicolons
  - an operator is implemented like any other method
  - implicit conversion
Implicit conversion

- Strings like "direct:a" and "direct:b" don't have the necessary methods (→ and to)
- String is final so it can't be subclassed
- Using implicit conversion to 'add' the missing methods

```scala
class RouteBuilder {

  implicit def stringToUri(uri: String) = 
    new RichUriString(uri, this)
}
```
Implicit conversion

- Let's look at the RichUriString
  - Primary constructor is in class declaration
  - Defines two methods (return type inference)

```scala
class RichUriString(uri: String, builder: RouteBuilder) {
  def to(target: String) = builder.from(uri).to(target)
  def --> (target: String) = to(target)
}
```
Implicit conversion

• The full Scala RouteBuilder class

```scala
package org.apache.camel.scala.dsl

class RouteBuilder {

    val builder = new org.apache.camel.builder.RouteBuilder()
        override def configure() = {}

    def from(uri: String) = builder.from(uri)

    implicit def stringToUri(uri: String) =
        new RichUriString(uri, this)

}
```
Implicit conversion

• There are a few subtle rules that can bite you when using implicit conversion
  - marking rule
  - scope rule
  - explicit-first rule
  - one-at-a-time rule
  - non-ambiguity rule
  Example: filter method on ProcessorType/RichString
Filter route example

- Java DSL filter looks like this

```java
public class MyRouteBuilder extends RouteBuilder {
    public void configure() throws Exception {
        from("direct:a").filter(body().isEqualTo("<hello/>")).to("mock:a");
    }
}
```
Filter route example

• In the Scala DSL

```scala
class FilterRouteBuilder extends RouteBuilder {

  "direct:a" when(_.in == "<hello/>") to "mock:a"

}
```

• Scala language features
  - passing functions as parameters
  - equals() in Java becomes == in Scala
Passing functions as parameters

• Scala is a functional language
  – functions are variables
  – you can pass functions as method parameters
• Let's pass a function to the when() method

```scala
class RichUriString(uri: String, builder: RouteBuilder) {
  def when(test: Exchange => Boolean) = 
    builder.from(uri).filter(new WhenPredicate(test))
}
```
Passing functions as parameters

- Predicate\}<E> is an interface in the Camel API
  - WhenPredicate is a Scala class that implements it
  - Use the function with an Exchange to evaluate

```scala
package org.apache.camel.scala.dsl

class WhenPredicate(function: Exchange => Boolean)
  extends Predicate[Exchange]{

  override def matches(exchange: Exchange) = function(exchange)

  //assertMatches is also here

}
```
Passing functions as parameters

- Passing a function literal in the RouteBuilder

```java
class FilterRouteBuilder extends RouteBuilder {
    "direct:a" when(
        (exchange:Exchange) => exchange.in == "<hello/>",
    ) to "mock:a"
}
```

- Shorthand notation
  - with parameter type inference...
    ```java
    exchange => exchange.in == "<hello/>
    ```
  - and placeholders
    ```java
    _.in == "<hello/>
    ```
• Java DSL for a simple content-based router

```java
public class MyRouteBuilder extends RouteBuilder {

    public void configure() throws Exception {
        from("direct:a")
            .to("mock:polyglot")
            .choice()
                .when(body().isEqualTo("<hallo/>"))
                    .to("mock:dutch")
                    .to("mock:german");
                .when(body().isEqualTo("<hello/>"))
                    .to("mock:english")
            .otherwise()
                .to("mock:french");
    }
}
```
CBR example

• Scala DSL adds code blocks for supporting more advanced route definitions

class CBRRouteBuilder extends RouteBuilder {

"direct:a" => {
  to ("mock:polyglot")
  choice {
    when (_.in == "<hello/>") to ("mock:english")
    when (_.in == "<hallo/>") {
      to ("mock:dutch")
      to ("mock:german")
    }
    otherwise to ("mock:french")
  }
}
}
By-name parameters and currying

- By-name parameters allow you to just pass a block of code that takes no parameters

```scala
class RouteBuilder {

  // instead of : def choice(block: () => Unit)
  def choice(block: => Unit) = {
    // just execute the block (no parentheses)
    block
  }
}
```
By-name parameters and currying

- Currying allows you to use a method that takes multiple arguments lists

```scala
class RouteBuilder {

//snip

  def when(test: Exchange => Boolean)(block: => Unit) = {
    val when = choice.when(new WhenPredicate(test))
    build(when, block)
  }
}
```
Caveats

- Interaction between Java and Scala generics
- Java varargs versus Scala repeated parameters
- Operator precedence
Operator precedence

- Scala allows you to override operators or declare symbol named methods
  - precedence is determined on the first character

```scala
class SimpleRouteBuilder extends RouteBuilder {

    // these are all the same
    "direct:a" to "mock:a1" to "mock:a2"
    "direct:b" --> "mock:b1" --> "mock:b2"
    "direct:c" --> "mock:c1" to "mock:c2"

    // but this is something entirely different
    "direct:d" to "mock:d1" --> "mock:d2"
}
```
Java/Scala generics

• Most of the times, you can simply replace < > by []

• A Java type defined as...

```java
public class ProcessorType<Type extends ProcessorType> {}
```

• In Java, you can also declare the raw type ... (you'll only get compiler warnings)

• ... but in Scala this doesn't work. The solution is this (ugly-looking) syntax (existential type).

```scala
implicit def processorWrapper(
  processor: ProcessorType[T] forSome {type T}) =
  new RichProcessor(processor)
```
Varargs/repeated parameters

• Java varargs...

```java
public Type to(String... uri) {
    //does some work
}
```

• ... are like Scala repeated parameters

```scala
def to(uris: String*) = //implementation goes here
```

• Caveats:

```scala
def to(uris: String*) = {
    val processor = builder.from(uri)
    processor.to(uris.toArray[String])
}

def -->(uris: String*) = to(uris:_*)
```
Other language features

- What else is there?
  - traits and mixins
  - pattern matching
  - partially applied functions
  - apply() and unapply()
  - language support for XML
    XML literals, pattern matching for XML, ...
  - actors
  - annotation support
  - ...

Planning

• Introduction
• Scala for DSL building
  – Implicit conversion
  – Passing functions as parameters
  – By-name parameters and currying
  – Caveats
• Scala tooling
  – Maven plugin
  – Eclipse plugin
Scala Maven plugin

• Integrate Scala in your current Maven build
  - http://scala-tools.org/mvnsites/maven-scala-plugin/
  - specify repository and plugin
  - also need to specify source/test folders

• Other features
  - continuous compilation (scala:cc)
  - scaladoc generation (scala:doc)
  - scala interactive console (scala:console)
Scala Eclipse plugin

- Scala plugin for Eclipse
    - Scala development perspective
    - Syntax highlighting and formatting
    - Wizards for classes, traits, objects, ...

- But...
  - If you have problems, resort to manual building (Ctrl-B)
  - Occasionally, you may have to clean your project to get up-to-date compile messages
Scala Eclipse plugin

- Configuring Maven Eclipse plugin to generate Scala project descriptors
  - add a nature:
    ch.epfl.lamp.sdt.core.scalanature
  - add a builder:
    ch.epfl.lamp.sdt.core.scalabuilder
  - add a build classpath container:
    ch.epfl.lamp.sdt.launching.SCALA_CONTAINER
Thanks for attending... 

Questions? Remarks?