Scala and its Ecosystem

Petr Hošek
D³S Seminar, September 2010
Language is useless without its ecosystem. Tools, libraries, frameworks, etc. Growing usage demands the need of ecosystem. Java ecosystem is not enough for Scala.
01 Scala compiler
02 Scala tools & libraries
03 Scala frameworks
04 Collaborative Scaladoc
05 Lessons learned
06 Questions & Answers
01 Scala compiler
Second generation of Scala compiler.

Officially named *NSC* (New Scala Compiler).

Entirely developed in Scala itself (*self-hosted*).

Compiles Scala directly into Java bytecode.
Allows to be used in several ways. Standalone, interactive, embedded.

Designed to be completely re-entrant. Can be instantiated as any other class.

Compilation consists of several phases. Each phases transforms the syntax tree.
Developed in entirely modular way. Compiler is composed of several component.
  Represented by traits.
Compiler can be easily extended via plugins.
  They can be inserted in all phases of compilation.
Scala tools & libraries
They form the basis of Scala ecosystem. Many new tools appeared in the last few months. Most of them replaces Java equivalents. New tools are coming up every day.
simple-build-tool

Simple but powerful build tool for Scala.
  Configuration is written directly in Scala.
Provides interactive and batch mode.
Dependency management support.
Integrated support for many Scala tools.
  Lot of plugins and extensions do exists.
import sbt._

class ExampleProject(info: ProjectInfo) extends ParentProject(info) {
  override def fork = forkRun(Some(new File("demo")), "-Xmx2G" :: Nil)

  lazy val subA = project("subA", "Sub Project A", new ExampleSubProject(_))
  lazy val subB = project("subB", "Sub Project B", new ExampleSubProject(_))

  class ExampleSubProject(info: ProjectInfo) extends DefaultProject(info) {
    def compileOptions: Seq[CompileOption] = Verbose :: Nil

    val specs = "org.scala-tools.testing" % "specs_2.8.0" % "1.6.5" % "test->default"
  }
}
specs

Behaviour-driven design framework for Scala. Simple and typed language for specifications. Benefits from Scala expressive syntax. Integration with testing tools and frameworks. JUnit, ScalaCheck, Mockito, etc. Support for literate programming.
class GreetingSpecification extends HtmlSpecification with Textile {
    "The greeting application" is <t>

    h3. Presentation

    This new application should say "hello" in different languages.

    For example,<ex>by default, saying hello by default should use English</ex>
    { greet must_== "hello" }

    Then, other languages, like <ex>French and German should be supported too</ex>
    { eg {
        greet("French") must_== "bonjour"
        greet("German") must_== "hallo"
    } }

    <ex>Japanese should be supported also</ex> { notImplemented }

    </t>
}
Configgy

Library for handling configuration and logging.
Simple configuration format.
  But still very powerful.
Support for JMX and notification of changes.
Extremely simple API.
include "/opt/config/local.conf"
log (inherit="log-base") {
    filename = "/var/log/example.log"
    level = "debug"
    utc = true
    verbose {
        node = "com.example.*"
        level = "trace"
    }
}
hostname = "example.com"
port = 3000

Configgy.configure("~/example.conf")

val config = Configgy.config
val hostname = config.getString("hostname", "localhost")
val port = config.getInt("port", 3000)

val log = Logger.get
log.error("Unable to listen on %s:%d!", hostname, port)
Scala frameworks
Akka framework

Framework for concurrent, fault-tolerant, scalable applications development.

- Based on event-driven approach.

Provides actors, STM, transactors, etc.

- Together with Scala as well as Java API.

Many add-on modules available.

- REST, Comet, Spring, Guice, OSGi integration, etc.
Lift web framework

Expressive and elegant web framework.
   Benefiting from Scala language features.
Embraces View-First approach to MVC.
Importance of scalability and security.
   Without loss of performance or maintainability.
Native support for Ajax a Comet.
04 Collaborative Scaladoc
Newest addition to Scala ecosystem.
Consisting currently from two applications.
Contribution of Scala project documentation.
New approach of documentation authoring.
Using the concepts of social collaboration.
Scaladoc

Analogy of Javadoc for Scala.

Part of Scala compiler currently as second generation.
Contains new sleek and modern interface.
Provides improved comment syntax.

Supports wiki-like syntax in the source comments.
Colladoc

Allows to edit Scala symbols documentation. Lift web application running the Scala compiler. Developed as a Google SoC 2010 project.

Now being developed as open-source project. Based heavily upon Scaladoc 2 functionality.

„Do not reinvent the wheel.“
Mergedoc

Allows to merge changes into the source-code.
Simple command-line utility.
   Built on top of Scala compiler.
Not yet officially released.
   Reimplementation of scaladoc-merge tool.
Collaboration is the main goal.
   Using collaborative development tools.
Joining the effort is easy.
   Fork and contribute at GitHub.
   Track the development at Lighthouse.
Learn more at http://petrhosek.name/
05 Lessons learned
Many dimensions of simplicity and complexity. Scala itself is a relatively simple language. Grammar is somewhat smaller than Java or C#. Scala concepts are very general and orthogonal. They can be combined in a large number of ways.
Scala is different from mainstream languages. 
C/C++, Java, C#, etc.
Scala libraries & tools takes different approach. 
  simple-build-tool, specs, Lift framework, etc.
Learning Scala is not difficult.
06 Questions & Answers