Object Scala Found

a JSR223-compliant version of the scala interpreter

http://github.com/rjolly/scala-compiler

Raphaël Jolly
Databeans
http://databeans.sourceforge.net/
raphael.jolly@free.fr

Scala Days 2011
June 3rd

Overview

- **Introduction**
  - Motivation
  - Implementation
- Three main impediments
- Applications
- Related work
- Conclusion

Motivation

- JSR223 : widely adopted standard
- To have Scala recognized as a true scripting language, on par with (J)ython and (J)Ruby
- Command-line interpreter : jrunscript
- Applet : issues with restricted environment (sandbox)
- Server side scripting : vision of JSP realized (Java PHP)

Command-line interpreter

```
$ scala
Welcome to Scala version 2.8.1.final (Java HotSpot(TM) Client VM, Java
1.6.0_21).
Type in expressions to have them evaluated.
Type :help for more information.

scala> "hello world"
res0: java.lang.String = hello world

scala> $
```

```
$ jrunscript -classpath scala-compiler.jar -l scala
```

Implementation

- Modified Scala interpreter
- Implement javax.script.ScriptEngine
- Provide javax.script.ScriptEngineFactory
- Use of Java 6 java.util.ServiceLoader scheme

scala.tools.nsc.interpreter.IMain

```
class IMain(@BeanProperty val factory: ScriptEngineFactory, val settings: Settings,
  protected val out: PrintWriter) extends AbstractScriptEngine(new SimpleBindings)
  with Compilable with Imports {
  imain =>

  /** construct an interpreter that reports to Console */
  def this(settings: Settings, out: PrintWriter) = this(null, settings, out)
  def this(factory: ScriptEngineFactory, settings: Settings) = this(factory,
  settings, new NewLinePrintWriter(new ConsoleWriter, true))
  def this(settings: Settings) = this(settings, new NewLinePrintWriter(new
  ConsoleWriter, true))
  def this(factory: ScriptEngineFactory) = this(factory, new Settings())
  def this() = this(new Settings())

  def createBindings: Bindings = new SimpleBindings

  @throws(classOf[ScriptException])
  def eval(script: String, context: ScriptContext): Object =
  compile(script).eval(context)

  @throws(classOf[ScriptException])
  def eval(reader: Reader, context: ScriptContext): Object =
  compile(reader).eval(context)

  override def finalize = close
}
```

scala.tools.nsc.interpreter.IMain (factory)

```
object IMain {
  class Factory extends ScriptEngineFactory {
    @BeanProperty
    val engineName = "Scala Interpreter"

    @BeanProperty
    val mimeTypes: JList[String] = Arrays.asList("application/x-scala")

    @BeanProperty
    val names: JList[String] = Arrays.asList("scala")

    def getParameter(key: String): Object = key match {
      case ScriptEngine.ENGINE => engineName
      case ScriptEngine.NAME => names.get(0)
      case _ => null
    }

    def getScriptEngine: ScriptEngine = new IMain(this, new Settings() {
       usemanifestcp.value = true
      })
  }
}
```

build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="scala-compiler" default="default" basedir=".">
    <description>Builds, tests, and runs the project scala-
  compiler.</description>
    <import file="nbproject/build-impl.xml"/>

    <target name="-post-jar">
        <jar destfile="${dist.jar}" update="true">
             <manifest>
               <attribute name="Class-Path" value="scala-library.jar"/>
            </manifest>
             <service type="javax.script.ScriptEngineFactory"
  provider="scala.tools.nsc.interpreter.IMain$Factory"/>
             <zipfileset includes="ch/" src="${scala.compiler}"/>
        </jar>
        <jar destfile="${dist.dir}/scala-library.jar"
  manifest="manifest.mf">
             <zipfileset src="${scala.library}"/>
         </jar>
    </target>
</project>
```

Overview

- Introduction
- **Three main impediments**
  - Passing statically typed arguments to the scripting engine
  - Caching pre-compiled scripts
  - Providing a class path to the scala compiler
- Applications
- Related work
- Conclusion

Passing statically typed arguments to the scripting engine

- JSR223's dynamic approach for passing arguments does not play well with Scala's static types - lack of type information when passing objects past the Java/script boundary
- existing solutions (*) are imperfect - they require some enclosing script ceremony

```
package forum
class html(args: htmlArgs) {
  import args._
  // ...
}
```

(*) M. Dürig. Scala for scripting. Technical report, Day Software AG, 2010. URL http://days2010.scala-lang.org/sites/days2010/files/15-5-E - Scripting - Dürig.pdf

Passing statically typed arguments to the scripting engine (cont)

- Unsuitable for REPL
- Has a consequence on point 2 (caching of pre-compiled scripts)
- A problem for web templates, but not for other kinds of scripts

```
val request = bindings.get("request").asInstanceOf[HttpServletRequest]
```

- Put aside (not further considered)

Caching pre-compiled scripts

- performance constraints require caching of pre-compiled scripts
- especially useful for web applications
- can not afford to re-compile the dynamic pages at every web request
- using the optional Compilable and CompiledScript interface and class of the javax.script API

scala.tools.nsc.interpreter.IMain (before)

```
  def interpret(line: String, synthetic: Boolean): IR.Result = {
    def loadAndRunReq(req: Request) = {
      val (result, succeeded) = req.loadAndRun
    }

    if (global == null) IR.Error
    else requestFromLine(line, synthetic) match {
      case Left(result) => result
      case Right(req)   =>
        // null indicates a disallowed statement type; otherwise compile
and
        // fail if false (implying e.g. a type error)
        if (req == null || !req.compile) IR.Error
        else loadAndRunReq(req)
    }
  }
```

scala.tools.nsc.interpreter.IMain

```
def interpret(line: String, synthetic: Boolean): IR.Result = compile(line,
synthetic) match {
  case Left(result) => result
  case Right(req)   => new WrappedRequest(req).interpret(synthetic)
}

private def compile(line: String, synthetic: Boolean): Either[IR.Result, Request]
= {
  if (global == null) Left(IR.Error)
  else requestFromLine(line, synthetic) match {
    case Left(result) => Left(result)
    case Right(req)   =>
     // null indicates a disallowed statement type; otherwise compile and
     // fail if false (implying e.g. a type error)
     if (req == null || !req.compile) Left(IR.Error) else Right(req)
  }
}

@throws(classOf[ScriptException])
def compile(reader: Reader): CompiledScript = {
  val writer = new StringWriter()
  var c = reader.read()
  while(c != -1) {
    writer.write(c)
    c = reader.read()
  }
  reader.close()
  compile(writer.toString())
}
```

scala.tools.nsc.interpreter.IMain

```
var code = ""
var bound = false
@throws(classOf[ScriptException])
def compile(script: String): CompiledScript = {
  if (!bound) {
    quietBind("bindings", getBindings(ScriptContext.ENGINE_SCOPE))
    bound = true
  }
  val cat = code + script
  compile(cat, false) match {
    case Left(result) => result match {
      case IR.Incomplete => {
        code = cat + "\n"
        new CompiledScript {
          def eval(context: ScriptContext): Object = null
          def getEngine: ScriptEngine = IMain.this
        }
      }
      case _ => {
        code = ""
        throw new ScriptException("compile-time error")
      }
    }
    case Right(req)   => {
      code = ""
      new WrappedRequest(req)
    }
  }
}
```

scala.tools.nsc.interpreter.IMain (wrapped request)

```
private class WrappedRequest(val req: Request) extends CompiledScript
{
  @throws(classOf[ScriptException])
  def eval(context: ScriptContext): Object = beQuietDuring {
    interpret(false) match {
     case IR.Success =>
       try req.getEvalTyped[Object] orNull
       catch { case e: Exception => throw new ScriptException(e) }
     case _ => throw new ScriptException("run-time error")
    }
  }

  def interpret(synthetic: Boolean): IR.Result =
loadAndRunReq(synthetic)

  def loadAndRunReq(synthetic: Boolean) = {
    val (result, succeeded) = req.loadAndRun
  }

  def getEngine: ScriptEngine = IMain.this
}
```

Reuse of compiled script (1)

```
private def requestFromLine(line: String, synthetic: Boolean): Either[IR.Result,
Request] = {
  val trees = parse(indentCode(line)) match {
    case None         => return Left(IR.Incomplete)
    case Some(Nil)    => return Left(IR.Error) // parse error or empty input
    case Some(trees)  => trees
  }
  // use synthetic vars to avoid filling up the resXX slots
  def varName = if (synthetic) freshInternalVarName() else freshUserVarName()
  // Treat a single bare expression specially. This is necessary due to it being
hard to
  // modify code at a textual level, and it being hard to submit an AST to the
compiler.
  if (trees.size == 1) trees.head match {
    case _:Assign                         => // we don't want to include
assignments
    case _:TermTree | _:Ident | _:Select  => // ... but do want these as valdefs.
      requestFromLine("def %s =\n%s".format(varName, line), synthetic) match {
        case Right(req) => return Right(req withOriginalLine line)
        case x          => return x
      }
    case _                                =>
  }
  // figure out what kind of request
  Right(buildRequest(line, trees))
}
```

Reuse of compiled script (2)

```
class Request(val line: String, val trees: List[Tree]) {
  private object ResultObjectSourceCode extends
CodeAssembler[MemberHandler] {
    /** We only want to generate this code when the result
     *  is a value which can be referred to as-is.
     */
    val evalResult =
      if (!handlers.last.definesValue) ""
      else handlers.last.definesTerm match {
        case Some(vname) if typeOf contains vname =>
          """
          |def $result = {
          |  $export
          |  %s
          |}""".stripMargin.format(fullPath(vname))
        case _  => ""
      }
  }
}
```

Curiosity

```
    private def evalMethod(name: String) = {
      val methods = evalClass.getMethods filter (_.getName == name)
      assert(methods.size <= 1, "Internal error - eval object method " +
name + " is overloaded: " + Arrays.asList(methods))
      methods.head
    }
```

Providing a class path to the scala compiler

- This issue = the serious one
- The Scala compiler needs a class path to load class files
- Usually only a class loader is provided

```
« All languages which I looked at and which do
symbol resolution at compile time use the same
'classloader.getResource("foo.class")' hack to
get access to a class file. Languages which
require to browse the classes available to them
at compile time need to resort to even more
esoteric hacks » M. Dürig
```

Command-line interpreter

```
$ jrunscript -classpath scala-compiler.jar -l scala
scala> "hello world"

Failed to initialize compiler: object scala not found.
** Note that as of 2.8 scala does not assume use of the java classpath.
** For the old behavior pass -usejavacp to scala, or if using a Settings
** object programatically, settings.usejavacp.value = true.
java.lang.NullPointerException
scala> $
```

```
$ jrunscript -Djava.class.path=scala-library.jar -Dscala.usejavacp=true
-classpath scala-compiler.jar -l scala
scala> "hello world"
hello world
scala> $
```

- Problem with restricted environments (applet, servlet)
- Security issues

Listing classes

- need to know what classes are available and what classes are not, for instance : wildcard imports

```
import scala._
```

- solution : provide list of classes through jar manifest file

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 1.5.0_22-b03 (Sun Microsystems
Inc.)
Name: scala/xml/parsing/TokenTests.class
Name: scala/reflect/NamedType.class
...
```

Listing classes (cont)

- Comes at no cost in the case of signed jar files (applets)
- Create the list manually if problem with signing (GAE - would need a `–list-only` option to `jarsigner`)
- Can give access to any number of third party libraries (a must have)

```
jrunscript -classpath commons-codec-1.4.jar:scala-compiler.jar -l scala

import org.apache.commons.codec.binary.Base64
new String(Base64.decodeBase64("="))
new String(Base64.encodeBase64("".getBytes()))
```

scala.tools.util.PathResolver

```
object PathResolver {
  object Environment {
    def classPathEnv         = try envOrElse("CLASSPATH", "") catch
  { case ex: SecurityException => "" }
    def sourcePathEnv        = try envOrElse("SOURCEPATH", "") catch
  { case ex: SecurityException => "" }

    def javaBootClassPath    = try propOrElse("sun.boot.class.path",
  searchForBootClasspath) catch { case ex: SecurityException => "" }
    def javaExtDirs          = try propOrEmpty("java.ext.dirs") catch
  { case ex: SecurityException => "" }
    def scalaHome            = try propOrEmpty("scala.home") catch { case
  ex: SecurityException => "" }
    def scalaExtDirs         = try propOrEmpty("scala.ext.dirs") catch
  { case ex: SecurityException => "" }

    def javaUserClassPath    = propOrElse("java.class.path", "")
    def useJavaClassPath     = try propOrFalse("scala.usejavacp") catch {
  case ex: SecurityException => false }
  }
```

scala.tools.util.PathResolver (cont)

```
class PathResolver(settings: Settings, context: JavaContext) {
   object Calculated {
    def useJavaClassPath    = settings.usejavacp.value ||
  Defaults.useJavaClassPath
    def useManifestClassPath: Boolean = settings.usemanifestcp.value
    def javaUserClassPath   = if (useJavaClassPath)
  Defaults.javaUserClassPath else ""
    def sourcePath          = cmdLineOrElse("sourcepath",
  Defaults.scalaSourcePath)

    def basis = List(
      classesInExpandedPath(javaUserClassPath),     // 3. The Java
  application class path.
      classesInManifest(useManifestClassPath),      // 8. The Manifest
  class path.
      sourcesInPath(sourcePath)                     // 7. The Scala
  source path.
    )
  }
}
```

scala.tools.nsc.settings. StandardScalaSettings

```
trait StandardScalaSettings {
  val sourcepath =        PathSetting ("-sourcepath", "Specify
  location(s) of source files.", "") // Defaults.scalaSourcePath

  val usejavacp =      BooleanSetting ("-usejavacp", "Utilize the
  java.class.path in classpath resolution.")
  val usemanifestcp =  BooleanSetting ("-usemanifestcp", "Utilize the
  manifest in classpath resolution.")
}
```

scala.tools.nsc.util.ClassPath

```
object ClassPath {
  abstract class ClassPathContext[T] {
    def classesInExpandedPath(path: String) = classesInPathImpl(path,
  true)

    private def classesInPathImpl(path: String, expand: Boolean) =
      for (file <- expandPath(path, expand) ; dir <- Option(AbstractFile
  getDirectory file)) yield
        newClassPath(dir)

    def classesInManifest(used: Boolean) =
      if (used) for (url <- manifests) yield newClassPath(AbstractFile
  getResources url) else Nil
  }

  def manifests =
  classOf[ScalaObject].getClassLoader().getResources("META-
  INF/MANIFEST.MF").toList
}
```

scala.tools.nsc.io.AbstractFile

```
object AbstractFile {
  def getDirectory(file: File): AbstractFile = try {
    if (file.isDirectory) new PlainFile(file)
    else if (file.isFile && Path.isJarOrZip(file)) ZipArchive fromFile
  file
    else null
  } catch { case ex: SecurityException => null }
  def getURL(url: URL): AbstractFile = {
    if (url == null || !Path.isJarOrZip(url.getPath)) null
    else ZipArchive fromURL url
  }
  def getResources(url: URL): AbstractFile = ZipArchive fromManifestURL
  url
}
```

scala.tools.nsc.io.ZipArchive

```
object ZipArchive {
  def fromFile(file: File): ZipArchive =
    try new ZipArchive(file, new ZipFile(file.jfile))
    catch { case _: IOException => null }

  def fromArchive(archive: ZipFile): ZipArchive =
    new ZipArchive(File(archive.getName()), archive)

  def fromURL(url: URL): AbstractFile = new URLZipArchive(url)

  def fromManifestURL(url: URL): AbstractFile = new
  ManifestResources(url)

  private[io] class ManifestEntryTraversableClass(in: InputStream)
  extends Iterable[ZipEntry] with ZipTrav {
    val manifest: Manifest = new Manifest(in)
    def iterator = manifest.getEntries().keySet().iterator().map(new
  ZipEntry(_))
  }
}
```

scala.tools.nsc.io.ZipArchive (cont)

```
  private[io] trait ZipContainer extends AbstractFile {
    protected[io] trait FileEntryInterface extends EntryInterface {
      override def sizeOption = entry.getSize().toInt match {
        case n if (n < 0) => None
        case n => Some(n)
      }
    }
  }
```

```
final class ManifestResources(url: URL) extends AbstractFile with ZipContainer {
  protected lazy val root = new ZipRootCreator(x => resourceInputStream(x.path))()

  protected def ZipTravConstructor    = new
ZipArchive.ManifestEntryTraversableClass(_)

  private def resourceInputStream(path: String): InputStream = {
    new FilterInputStream(null) {
      override def read(): Int = {
        if(in == null) in = classOf[ScalaObject].getResourceAsStream("/" + path);
        if(in == null) throw new RuntimeException("/" + path + " not found")
        super.read();
      }
    }
  }
}
```

scala.tools.nsc.io.AbstractFile

```
@throws(classOf[IOException])
def toByteArray: Array[Byte] = {
  val in = input
  sizeOption match {
    case Some(size) =>
      var rest = size
      val arr = new Array[Byte](rest)
      while (rest > 0) {
        val res = in.read(arr, arr.length - rest, rest)
        if (res == -1)
          throw new IOException("read error")
        rest -= res
      }
      in.close()
      arr
    case None =>
      val out = new ByteArrayOutputStream()
      var c = in.read()
      while(c != -1) {
        out.write(c)
        c = in.read()
      }
      in.close()
      out.toByteArray()
  }
}
```

scala.tools.nsc.interpreter.AbstractFileClassLoader

```
class AbstractFileClassLoader(root: AbstractFile, parent: ClassLoader)
  extends ClassLoader(parent) with ScalaClassLoader {
  lazy val protectionDomain = new ProtectionDomain(new CodeSource(new
  URL(path), null.asInstanceOf[Array[Certificate]]), null, this, null)

  def path =
  classOf[ScalaObject].getResource("/scala/ScalaObject.class").getPath
  match { case s => s.substring(0, s.lastIndexOf('!')) }

  override def findClass(name: String): JClass = {
    val bytes = classBytes(name)
    if (bytes.isEmpty) throw new ClassNotFoundException(name)
    else defineClass(name, bytes, 0, bytes.length, protectionDomain)
  }
}
```

scala.tools.nsc.settings. MutableSettings

```
    private def checkDir(dir: AbstractFile, name: String, allowJar:
Boolean = false): AbstractFile = (
      if (dir != null && dir.isDirectory)
        dir
      else if (allowJar && dir == null && Path.isJarOrZip(name, false))
        new PlainFile(Path(name))
      else
//      throw new FatalError(name + " does not exist or is not a
directory")
        dir
    )
```

Overview

- Introduction
- Three main impediments
- **Applications**
  - Servlet
  - Applet
- Related work
- Conclusion

Applications

- `jrunscript` command
- Google App Engine
- Tomcat web server version 5.5 (with security enbled)
- Java web start applet
- Netbeans module

ScalaServlet

```
public class ScalaServlet extends HttpServlet {
      protected void processRequest(HttpServletRequest req, HttpServletResponse
  resp) throws ServletException, IOException {
          ServletContext context = getServletContext();
          if (context.getAttribute("interpreter") == null) {
              context.setAttribute("interpreter", getEngineFactory("Scala
  Interpreter").getScriptEngine());
              context.setAttribute("cache", new HashMap());
          }
          // continued
      }
      static void pipe(Reader in, Writer out) throws IOException {
          while (true) {
              int c = in.read();
              if (c == -1) break;
              out.write(c);
          }
      }
      public static ScriptEngineFactory getEngineFactory(String name) {
          ServiceLoader<ScriptEngineFactory> sefLoader =
  ServiceLoader.load(ScriptEngineFactory.class);
          for (ScriptEngineFactory sef : sefLoader) {
              if(name.equals(sef.getEngineName())) return sef;
          }
          return null;
      }
}
```

ScalaServlet (cont)

```
            // continued
            ScriptEngine engine = (ScriptEngine)context.getAttribute("interpreter");
            Map cache = (Map)context.getAttribute("cache");
            synchronized(engine) {
                engine.put("request", req);
                engine.put("response", resp);
                String uri =
    req.getRequestURI().substring(req.getContextPath().length());
                InputStream in = context.getResourceAsStream(uri);
                if (in == null) {
                    resp.sendError(HttpURLConnection.HTTP_NOT_FOUND, uri);
                } else try {
                    resp.setContentType("text/html; charset=utf-8");
                    Reader r = new InputStreamReader(in);
                    Writer w = new StringWriter();
                    pipe(r, w);
                    w.close();
                    r.close();
                    String str = "{" + w + "}";
                    if(!cache.containsKey(str)) cache.put(str,
    ((Compilable)engine).compile(str));
                    CompiledScript cs = (CompiledScript)cache.get(str);
                    r = new StringReader(cs.eval().toString());
                    pipe(r, resp.getWriter());
                    r.close();
                } catch (ScriptException e) {
                    throw new ServletException(e);
                }
            }
```

/etc/tomcat5.5/policy.d/50user.policy

```
 grant codeBase "file:${catalina.base}/webapps/mywebapp/-" {
      permission java.util.PropertyPermission "*", "read";
      permission java.lang.RuntimePermission "createClassLoader";
      permission java.io.FilePermission "$
   {catalina.base}/webapps/mywebapp/-", "read";
 }
```

/usr/share/tomcat5.5/common/lib

```
scala-compiler.jar
scala-library.jar
scalaservlet.jar
```

ScalaApplet

```
62.201.135.246 - - [12/Apr/2011:11:12:14 +0200] "GET /-rjolly/jnlp/myapp.jar HTTP/1.1" 200 5925
62.201.139.56 - - [12/Apr/2011:11:14:15 +0200] "GET /-rjolly/jnlp/lib/scala-library.jar HTTP/1.1" 200
  6834743
62.201.139.56 - - [12/Apr/2011:11:14:27 +0200] "GET /-rjolly/jnlp/lib/scala-compiler.jar HTTP/1.1" 200
  9896181
62.201.135.246 - - [12/Apr/2011:11:14:35 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:35 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:35 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:35 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:36 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:36 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:36 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:36 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:36 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:38 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:38 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:38 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:39 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:39 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:39 +0200] "GET /-rjolly/jnlp/scala-compiler.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:41 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
62.201.135.246 - - [12/Apr/2011:11:14:41 +0200] "GET /-rjolly/jnlp/scala-library.jar HTTP/1.1" 404 -
```

Overview

- Introduction
- Three main impediments
- Applications
- **Related work**
- Conclusion

Related work

- SimplyScala (*)
- URL of jar file derived from the one of a class file ressource
- added as-is to the classpath (ZipArchive.fromURL)
- Work well on the server side
- On the client side the jar file is downloaded each time the applet is run
- Problem prompted the present work (computer algebra application through java web start)

(*) A. Bagwell. Simply scala. Technical report, http://www.simplyscala.com/, 2010-

Overview

- Introduction
- Three main impediments
- Applications
- Related work
- **Conclusion**

Conclusion

- Modified Scala interpreter
- Build classpath from libraries' jar manifest files
- Full independance w.r.t. hosting plateform
- Tested with success on several environments
- Up-to-date with Scala 2.9

Thank you !

http://github.com/rjolly/scala-compiler
http://github.com/rjolly/scala-compiler-samples