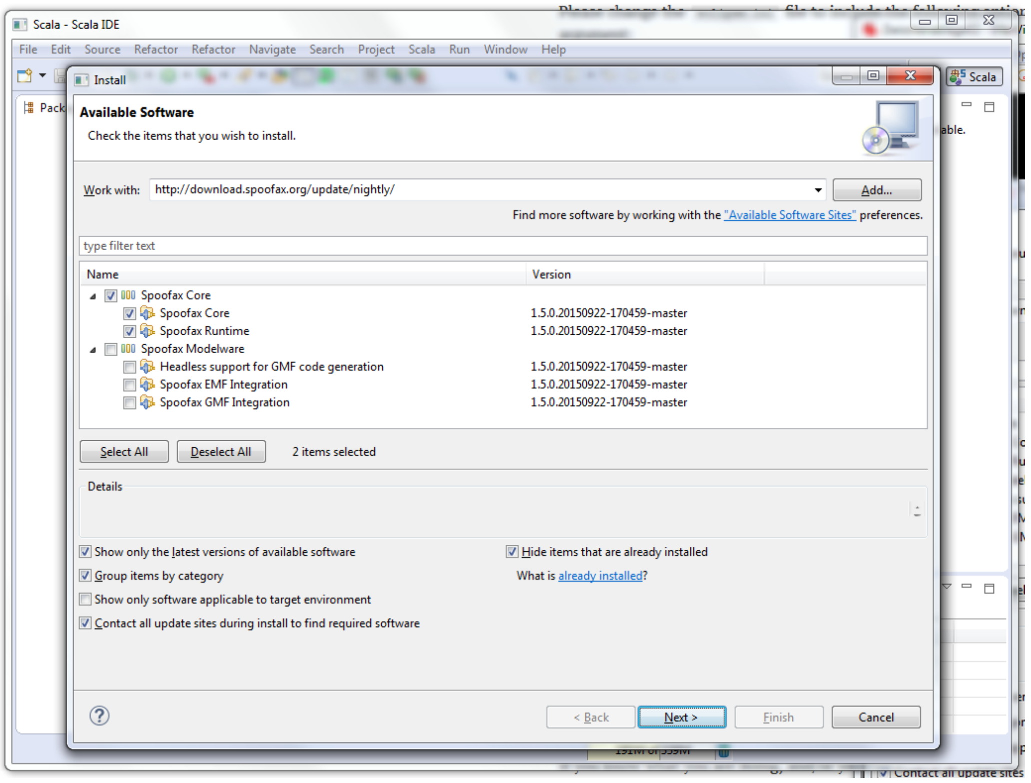
Using Scala strategies in Spoofax

1. Setting up the IDE: Scala IDE with Spoofax plugin

One way of installing Spoofax and the Scala plugin in eclipse is downloading the Eclipse IDE for Java Developers (currently Mars (4.5)) and then adding both plugins through `Help -> Install New Software...`. However, the Scala IDE offers better autocompletion, some additional interface elements and, according to the website, better performance, so we used it instead as a basis for installing the Spoofax plugin.

The steps are as follows:

- Install the JDK 7 or higher. The 64 bit version is recommended, since Spoofax warns you when not run with the server VM configuration (see below). This seems to be only available in the 64 bit version.
- Download and unzip the Scala IDE from above (already includes Scala 2.11, so no need to install it manually beforehand). Make sure to download the 64 bit version.
- Launch the Scala IDE, open `Help -> Install New Software...`, and `Add...` the Spoofax nightly repository (stable should work fine, it is just that we used nightly). As per the Spoofax website the repo URL is `http://download.spoofax.org/update/nightly/`. Select `Spoofax Core` and proceed with the installation.


Scala - Scala IDE
File Edit Source Refactor Refactor Navigate Search Project Scala Run Window Help
Install
Available Software
Check the items that you wish to install.
Add...
Select All
Deselect All
Details
< Back
Next >
Finish
Cancel

Work with: http://download.spoofax.org/update/nightly/

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
[✓] Spoofax Core	
[✓] Spoofax Core	1.5.0.20150922-170459-master
[✓] Spoofax Runtime	1.5.0.20150922-170459-master
[ ] Spoofax Modelware	
[ ] Headless support for GMF code generation	1.5.0.20150922-170459-master
[ ] Spoofax EMF Integration	1.5.0.20150922-170459-master
[ ] Spoofax GMF Integration	1.5.0.20150922-170459-master

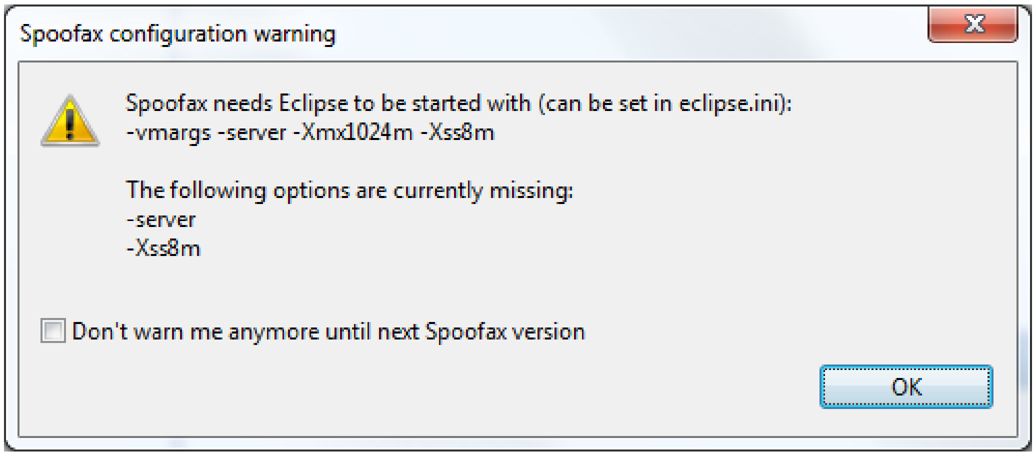
2 items selected

[✓] Show only the latest versions of available software
[✓] Hide items that are already installed
[✓] Group items by category
What is already installed?
[ ] Show only software applicable to target environment
[✓] Contact all update sites during install to find required software

Figure 1: Installing Spoofax Core from Scala IDE

- When you launch Scala IDE for the first time with Spoofax included, you might get the following warning (see the eclipse.ini section on the Spoofax website for more details). In order to fix it, add the following lines to the `eclipse.ini` in the Scala IDE's installation directory. (Note that some of the options might already be present due to the Scala IDE default configuration, just replace them.)

```
-Xss8m
-Xms256m
-Xmx1024m
-XX:MaxPermSize=256m
-server
-Djava.net.preferIPv4Stack=true
```



Spoofax configuration warning

Spoofax needs Eclipse to be started with (can be set in eclipse.ini):
-vmargs -server -Xmx1024m -Xss8m

The following options are currently missing:
-server
-Xss8m

Don't warn me anymore until next Spoofax version

OK

Figure 2: One possible Spoofax configuration warning

2. Creating a Spoofax and a Scala project

After the restart of Eclipse, no warning should show and we are ready to create a new Spoofax editor project by `File -> New -> Project... -> Other -> Spoofax editor project` (or use your existing one). The compilation should start automatically and after some time give you an open editor tab with an example file in your language (here: `example.sam`). You can verfiy that the Spoofax editor is working, if the `Syntax -> Show abstract syntax` button below the menubar is present and gives you the AST of your example module in a `example.aterm` file.

Now, create a Scala project by `File -> New -> Scala Project` in the same workspace. Add and run a main class to verify the Scala IDE part is also working.

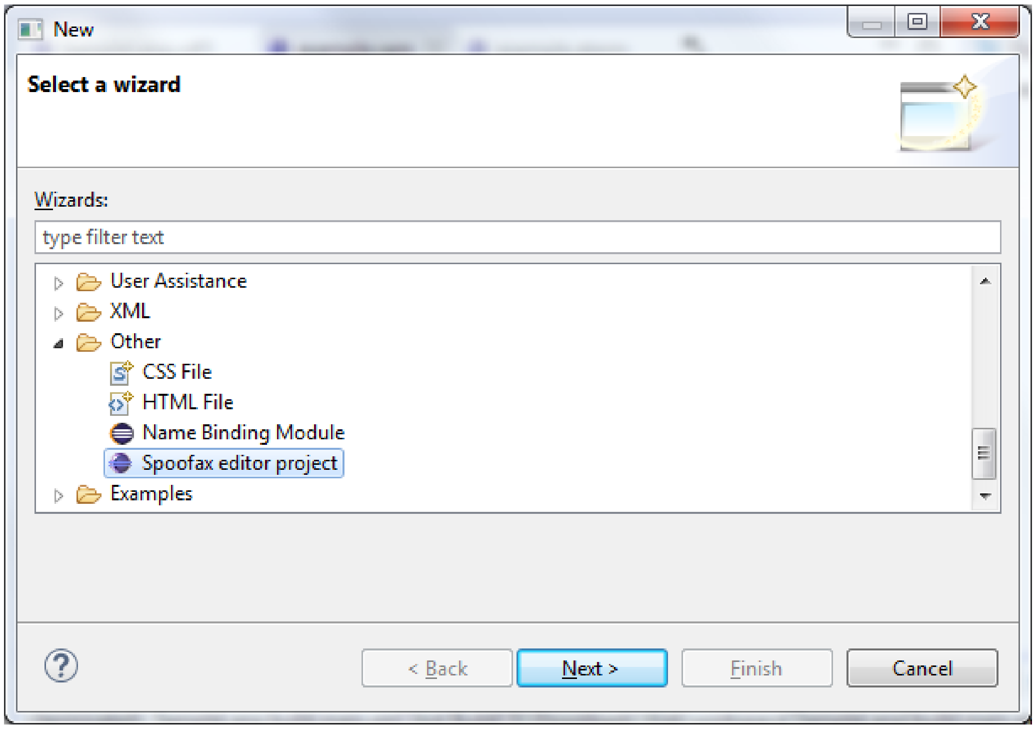
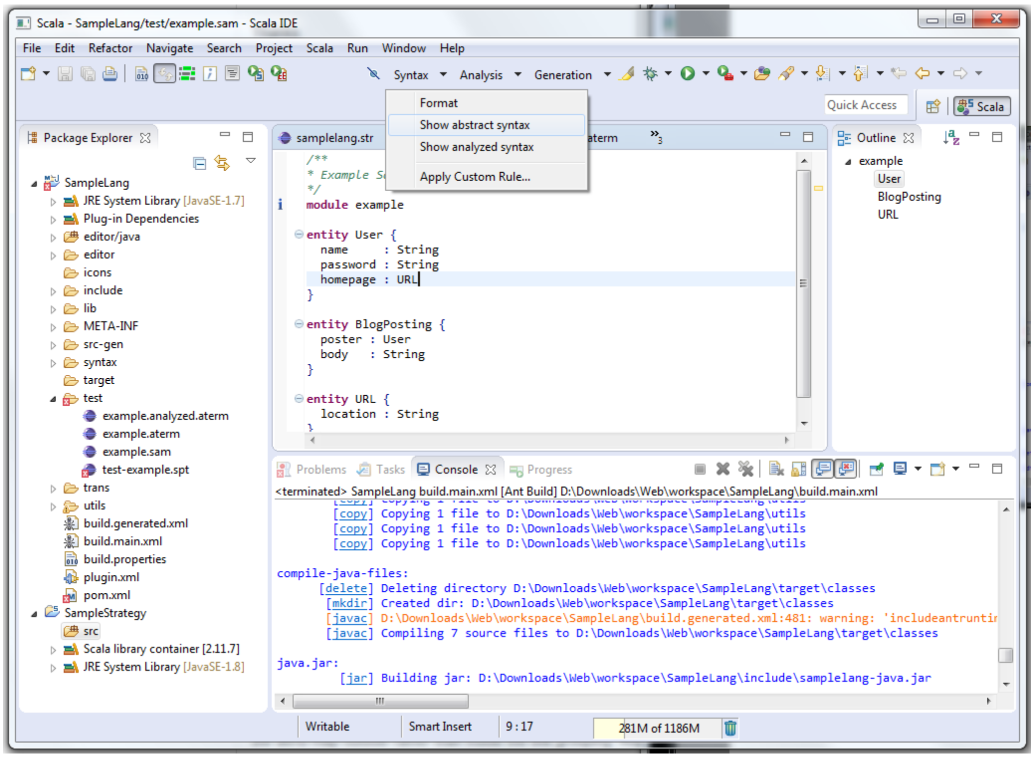


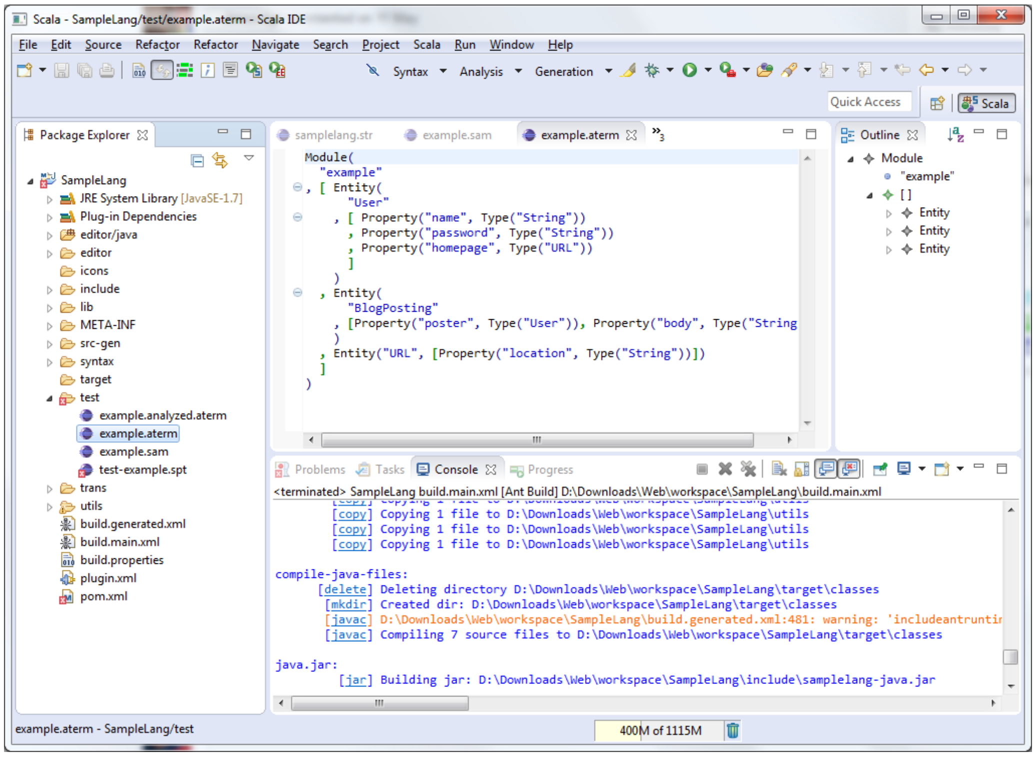
Figure 3: Creating a sample Spoofax editor project



```
/**
* Example S
*/
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}

entity URL {
  location : String
}
```

```
<terminated> SampleLang build.main.xml [Ant Build] D:\Downloads\Web\workspace\SampleLang\build.main.xml
      [copy] Copying 1 file to D:\Downloads\Web\workspace\SampleLang\utils
      [copy] Copying 1 file to D:\Downloads\Web\workspace\SampleLang\utils
      [copy] Copying 1 file to D:\Downloads\Web\workspace\SampleLang\utils

compile-java-files:
    [delete] Deleting directory D:\Downloads\Web\workspace\SampleLang\target\classes
     [mkdir] Created dir: D:\Downloads\Web\workspace\SampleLang\target\classes
     [javac] D:\Downloads\Web\workspace\SampleLang\build.generated.xml:481: warning: 'includeantrunti
     [javac] Compiling 7 source files to D:\Downloads\Web\workspace\SampleLang\target\classes

java.jar:
       [jar] Building jar: D:\Downloads\Web\workspace\SampleLang\include\samplelang-java.jar
```

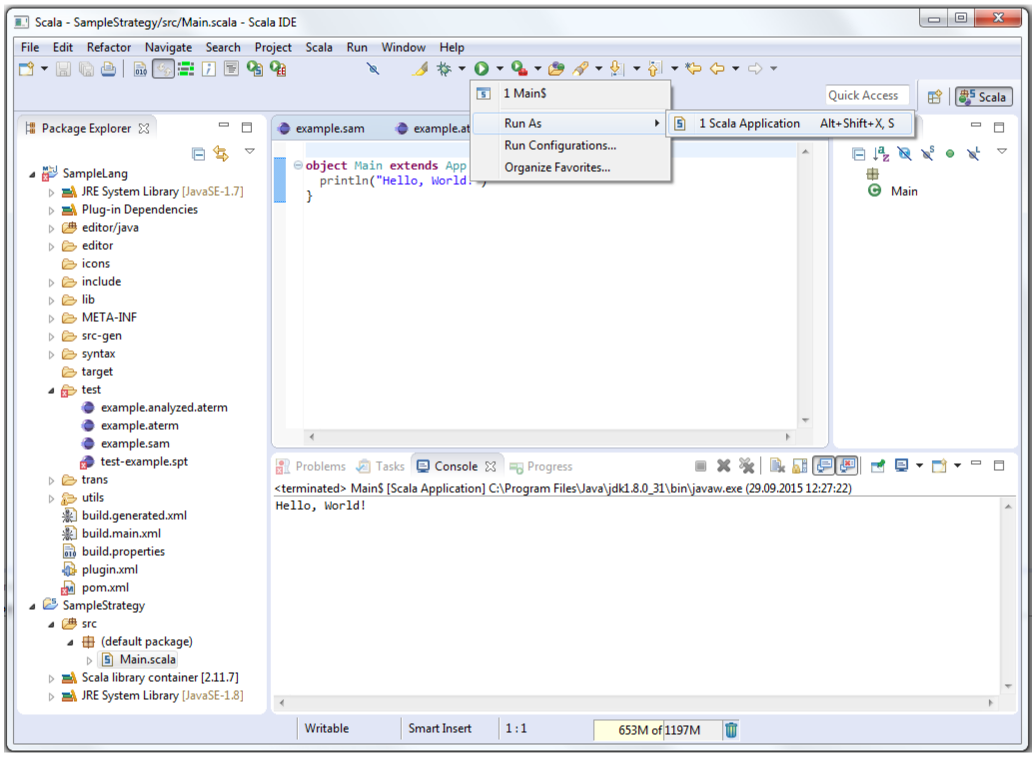
Figure 4: Opened `example.sam` with Spoofax editor buttons `Syntax`, `Analysis`, and `Generation`



```
Module(
  "example"
, [ Entity(
      "User"
    , [ Property("name", Type("String"))
      , Property("password", Type("String"))
      , Property("homepage", Type("URL"))
      ]
    )
  , Entity(
      "BlogPosting"
    , [Property("poster", Type("User")), Property("body", Type("String
    )
  , Entity("URL", [Property("location", Type("String"))])
  ]
)
```

```
<terminated> SampleLang build.main.xml [Ant Build] D:\Downloads\Web\workspace\SampleLang\build.main.xml
        [copy] Copying 1 file to D:\Downloads\Web\workspace\SampleLang\utils
        [copy] Copying 1 file to D:\Downloads\Web\workspace\SampleLang\utils
        [copy] Copying 1 file to D:\Downloads\Web\workspace\SampleLang\utils

compile-java-files:
      [delete] Deleting directory D:\Downloads\Web\workspace\SampleLang\target\classes
       [mkdir] Created dir: D:\Downloads\Web\workspace\SampleLang\target\classes
       [javac] D:\Downloads\Web\workspace\SampleLang\build.generated.xml:481: warning: 'includeantruntir
       [javac] Compiling 7 source files to D:\Downloads\Web\workspace\SampleLang\target\classes

java.jar:
         [jar] Building jar: D:\Downloads\Web\workspace\SampleLang\include\samplelang-java.jar
```

Figure 5: Generated AST in `example.aterm`



```
object Main extends App
  println("Hello, World!")
}
```

```
<terminated> Main$ [Scala Application] C:\Program Files\Java\jdk1.8.0_31\bin\javaw.exe (29.09.2015 12:27:22)
Hello, World!
```

Figure 6: Verify that Scala compilation and execution is working

3a. Running Scala strategies from Spoofax

The first variant of interoperability between Spoofax and Scala is to execute strategies written in Scala on the AST we have seen earlier in `example.aterm` from inside the Spoofax editor. (This is useful when your Scala strategy is currently not in focus, but you are making changes to your SampleLanguage source files. However, when developing/debugging the Scala strategy, directly executing the Scala strategy on `aterm` files (see 3b.) might be preferred.)

3a.1. Setting up the Scala strategy project

First, let us write a trivial Scala strategy that conforms with the Java interface for strategies in Spoofax. To make the Stratego Java classes available, go to the Scala project's `Properties -> Java Build Path -> Libraries -> Add JARs...` and select the `strategoxt.jar` inside `SampleLang/utils/`.

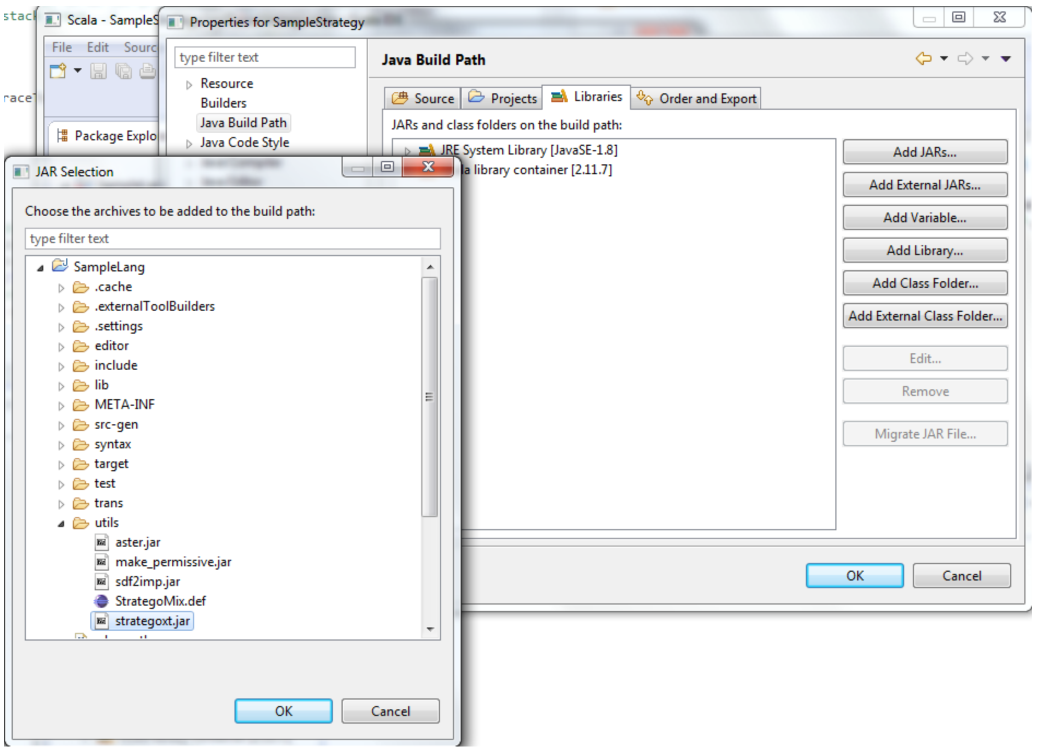


Figure 7: Add the StrategoXT jar to the Scala strategy's build path

As we want most of our code to be in Scala, but the Stratego interface requires us to create a static variable (which we can't in Scala), we write a very light wrapper around our actual Strategy. So we create two files:

`scala_strategy_0_0.java` (lowercase is mandatory! Also see the comment for the `_0_0` suffix.):

```
import org.spoofax.interpreter.terms.IStrategoTerm;
import org.strategoxt.lang.Context;
```

```java
import org.strategoxt.lang.Strategy;

/**
 * Java interface of the Scala strategy, such that it can be used from
 * Stratego as an external strategy.
 *
 * NOTE: Apparently, the Java strategy name uses these "_args1_args2"
 * suffixes for name mangling, more specifically for signifying the
 * number of term/strategy arguments.
 */
public class scala_strategy_0_0 extends Strategy {
    public static scala_strategy_0_0 instance = new scala_strategy_0_0();

    @Override
    public IStrategoTerm invoke(Context context, IStrategoTerm current) {
        return ScalaStrategy$.MODULE$.runAsStrategy(context, current);
    }
}
```

And ScalaStrategy.scala:

```scala
import org.spoofax.interpreter.terms.IStrategoList
import org.spoofax.interpreter.terms.IStrategoTerm

object ScalaStrategy {
  def runAsStrategy(context: org.strategoxt.lang.Context,
                    inputFromEditor: IStrategoTerm): IStrategoList = {
    // output to the Spoofax console is handled via Context.getIOAgent.printXYZ
    context.getIOAgent.printError("Hello, World!")
    // you return a list of tuples
    // (filename as IStrategoString, contents as IStrategoString)
    // trivial strategy: don't output any files == empty list
    context.getFactory.makeList()
  }
}
```

We will include the compiled strategy as a jar file into the Spoofax editor project. For that, create a new Ant build file (or really any other build system that can give you a jar file) build.xml with:

```xml
<?xml version="1.0" ?>
<project default="create-jar">
    <target name="create-jar">
        <jar jarfile="scala-strategy.jar" basedir="bin" includes="**" compress="false" >
            <!-- TODO! -->
        </jar>
    </target>
    <target name="clean-jar">
        <delete file="scala-strategy.jar" />
    </target>
</project>
```

Run the `build.xml` as an Ant build, you should now have a `scala-strategy.jar` in your Scala project.

3a.2. Setting up the Spoofax project

To make the strategy available to Spoofax, we need to

- Add the Scala project to the Spoofax project's build path. Go the the Spoofax project's `Properties -> Java Build Path -> Projects -> Add...` and add the Scala project.
- Add the `scala-strategy.jar` to the `build.xml` of the Spoofax project: TODO
- Fix the Ant build configuration of the Spoofax project to include the correct classpath: TODO
- Register the `scala_strategy_0_0` Java class as an external strategy in `InteropRegisterer.java`.
- (FIXME we don't need this, if the Scala code writes the output files) Add a stratego wrapper around the external strategy
- Add a button to invoke the Scala strategy to the editor

3a.3. Fixing the project dependencies and build order

TODO: Turn off automatic build

3b. Directly running the Scala strategy, not from within the Spoofax editor

As we have seen, to update Spoofax with the changes made to the Scala strategy, we need to reload or recompile the editor component everytime there are changes. To speed up the develop-run cycle, it might be worth to set up your strategy, such that it can be directly run from the commandline on your `aterm` files.

TODO

4. Optional: Spoofax/Stratego Sources

TODO