Module Developer's Tutorial

Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, Matej Vitásek
Advisors: RNDr. Irena Mlýnková, Ph.D., Martin Nečaský, Ph.D.

Praha, 2011

Module Developer's Tutorial

Target audience: jInfer module developers. Anyone who needs to extend jInfer capabilities by writing a new module.

This tutorial assumes that you are a seasoned Java developer. Having experience with programming in some kind of framework (NetBeans Platform above all) will help you a lot. Make sure you have read the article on architecture, data structures and inference process to understand what you will be implementing. Having read the documentation for remaining modules will help you too. Also, before starting this tutorial, make sure you can build jInfer from sources.

Overview

1. Get NetBeans, jInfer sources, try a build.
2. Decide on the type of module you want to create.
3. Create new NetBeans module.
4. Implement jInfer-specific interfaces.
5. Implement your logic.

Building jInfer from sources

Refer to the official instructions.

What type of module?

First thing you need to realize is what kind of module you will be implementing. Is it going to be a part of the inference? If yes, what stage? Importing the initial grammar? Simplifying it? Exporting to resulting schema? If you are not sure what these terms mean, go read the articles on interence process again.

If your logic doesn't belong to the inference process, it might just extend one of the existing modules. Try looking for the code you would like to change.

New NetBeans module

This section assumes you have successfully imported jInfer in NetBeans.

1. Expand the jInfer suite (brown puzzles icon).
2. Right-click *Modules*, select *Add New...*.
3. On the first page, pick a name for your module. Click *Next*.
4. On the second page, fill out the following:
   - *Code base*: this will be the package structure in which your code is placed. If you want, follow our convention: `cz.cuni.mff.ksi.jinfer.`*`yourmodule`*. Otherwise choose something like *`tld.company.jinfer.yourmodule`*.

- *Module display name*: pretty obvious.
- *Generate XML layer*: you may want to check this option, but it doesn't really matter at this stage.

Rest of the options is uninteresting in most cases.

5. Click *Finish*.

Implement jInfer's specifics

This section will show how to deal with a new inference module - a simplifier that does no actual simplification, just returns the grammar it got on input. Your module will need a "main" class implementing the chosen inference interface, annotated with a `@ServiceProvider` annotation.

But first, we need to do some setup.

1. Right-click the newly created module, select *Properties*.
2. In *Libraries* > *Module Dependencies*, click *Add Dependency...*.
3. Filter for "base" and select *Base* as a dependency. Click *OK*.
4. Still in *Properties*, switch to *Display* category. Fill in:
   - *Display Name*: Fill in a user-friendly name of your module.
   - *Display Category*: type in *jInfer*.
   - *Short & Long Description*: unleash your inner poet!
5. Still in *Properties*, switch to *API Versioning*.
6. Type `cz.cuni.mff.ksi.jinfer.base.interfaces.inference.Simplifier` in *Provided Tokens*.
7. Close *Properties* by clicking *OK*.

Now to the class itself.

1. Add a new Java class: right click *Source Packages* in your new module, select *New* > *Java Class...*.
2. Fill in class name, for example *MySimplifierImpl*.
3. Fill in package based on the *Code base* you selected while creating the module itself.
4. Click *Finish*.
5. In the heading line of the class, add `implements Simplifier` and fix the imports (you need `Simplifier` from Base module).
6. Annotate this class with `@ServiceProvider(service = Simplifier.class)`.
7. NetBeans will complain about missing method implementations, add them.
8. Now fill in method bodies:
   - `getName()`: return a string with module unix name, for example `"mysimplifier"`.
   - `getDisplayName()`: return a user friendly module name, for example `"My First Simplifier"`.
   - `getModuleDescription()`: return a short module description. You can also return `getDisplayName()` in this case.
   - `getCapabilities()`: for the moment, it is enough to return `Collections.emptyList()`.
   - `start()`: here is where your main logic belongs. In the case of a simplifier, you would do something with the rules and return a simplified grammar. For now, just do `callback.finished(initialGrammar);`

Run jInfer

At this moment, run the whole jInfer suite from the NetBeans you imported it into. A new, child NetBeans should open with your module correctly installed. You can now follow the User tutorial, just select your new simplifier while creating a new jInfer project. That's it! Start hacking you logic now!

Implement your logic

This is the part where you actually have to do some thinking :-) Implementing an importer? Take that `InputStream` and create some rules of it! Simplifier? Take the rules you got and compact them somehow! Exporter? Take those rules and write them out as a `String`! It only depends on which module you are in and algorithm you're implementing.

Dealing with jInfer and NetBeans Platform

You will soon get into situation where you need to interact either with jInfer or directly NetBeans. There is a tutorial for dealing with these cases.