How to integrate Plug-ins with ProB

Jens Bendisposto
University of Düsseldorf

Outline

- Project organization / Getting involved
- Current architecture
- Upcoming version
- Plan for the afternoon session

Prolog Sources

http://nightly.cobra.cs.uni-duesseldorf.de/source/

Index of /source

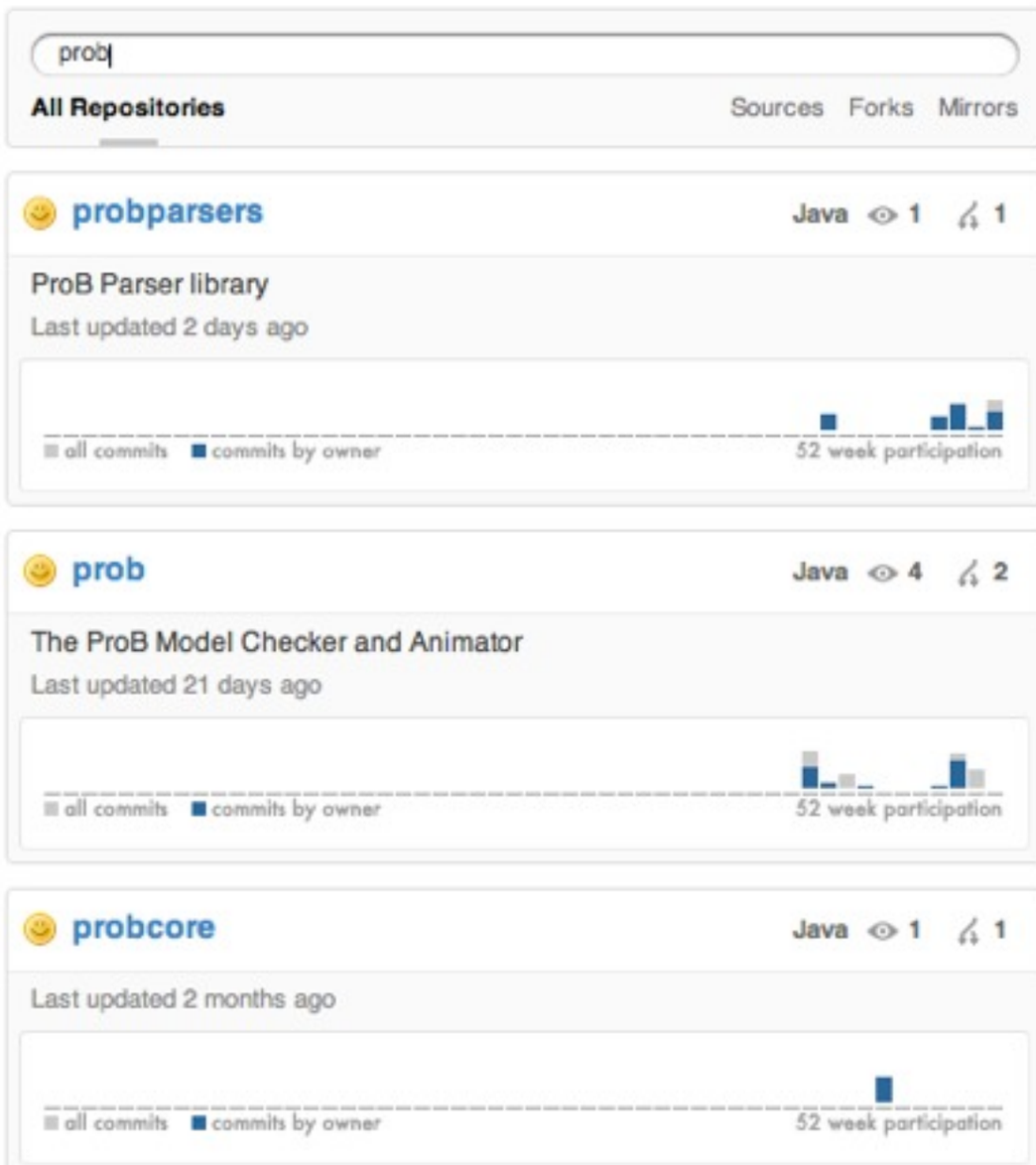
Name	Last modified	Size	Description
Parent Directory		-	
ProB_src.tgz	13-Feb-2012 16:03	21M	

Apache/2.2.14 (Ubuntu) Server at nightly.cobra.cs.uni-duesseldorf.de Port 80

- Building from source requires a Sicstus Prolog Licence.
- You can use our nightly builds:
  - http://nightly.cobra.cs.uni-duesseldorf.de/cli/ (console version)
  - http://nightly.cobra.cs.uni-duesseldorf.de/tcl/ (tcl/tk version)
- Build process: Build tcl version, if it succeeds release the sources

Java Sources

http://github.com/bendisposto



Parser Libraries

Current ProB plug-in

ProB 2.0 core

prob

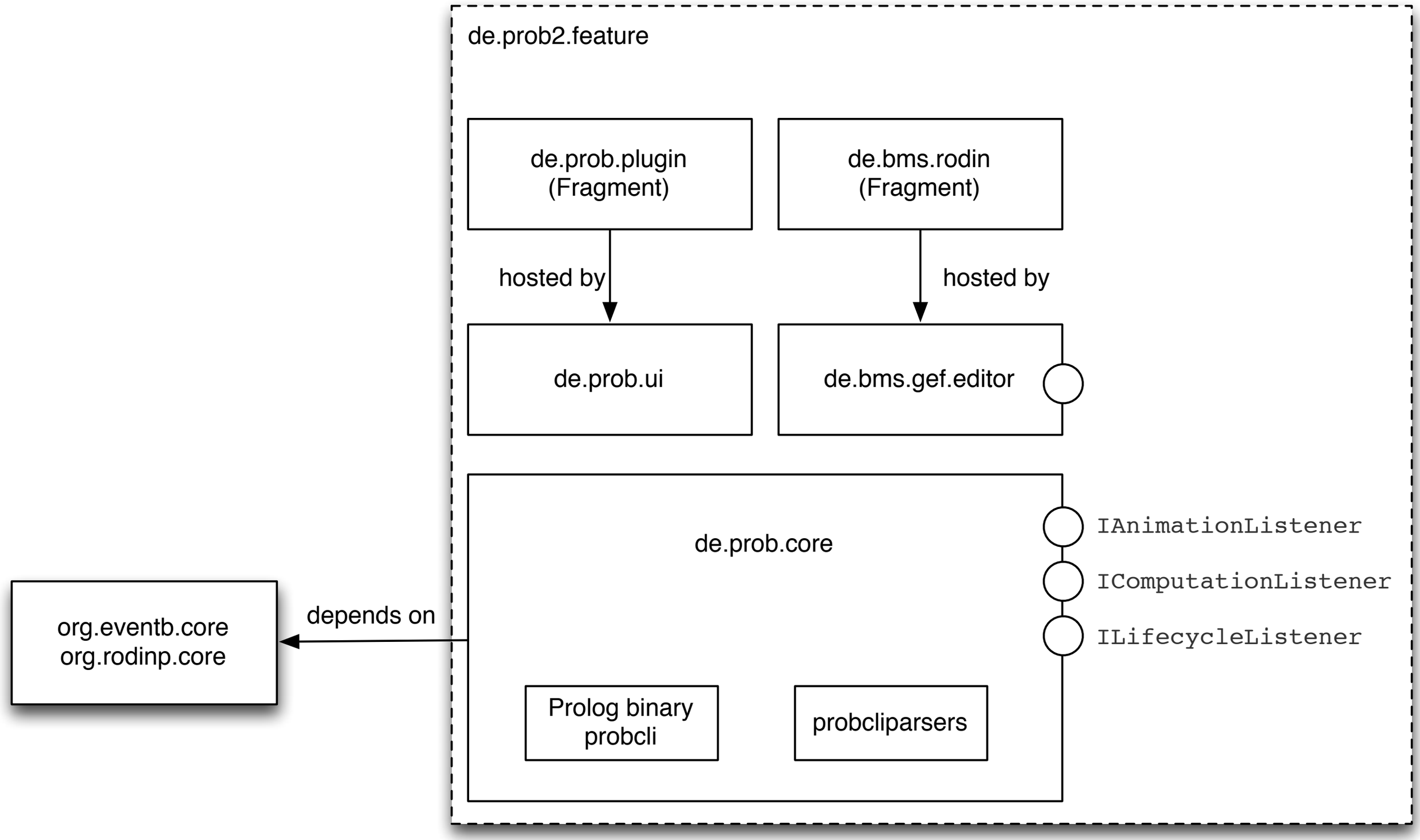
prob /

name
de.bmotionstudio.gef.editor
de.bmotionstudio.rodin
de.prob.core
de.prob.plugin
de.prob.ui
de.prob.update_site
de.prob2.feature
.gitignore
README.md
epl-v10.html

B-Motion Studio

ProB Plug-in

Metadata


de.prob2.feature
de.prob.plugin
(Fragment)
de.bms.rodin
(Fragment)
hosted by
hosted by
de.prob.ui
de.bms.gef.editor
de.prob.core
IAnimationListener
IComputationListener
ILifecycleListener
org.eventb.core
org.rodinp.core
depends on
Prolog binary
probcli
probcliparsers

Using ProB in your tool

- No particular extension mechanism for the UI
- Except for the usual Eclipse mechanisms
  - You can add toolbar buttons, context menus, ...
  - or even disable/replace existing implementation
- Reuse of existing view is simple (via part id)

de.prob.core

- `Animator a = Animator.getAnimator()`
  - Will transparently start a new animator if neccessary
  - Otherwise it returns a singleton instance of the Animator
- `a.execute(someCommand);`
  - executes the command
  - catches and reports errors from Prolog
  - stores results inside the command object
  - never reuse a command instance!

Commands

- A command encapsulates a query to the Prolog core and its result

```
public interface IComposableCommand {

  void writeCommand(IPrologTermOutput pto)
       throws CommandException;

  void processResult(ISimplifiedROMap<String, PrologTerm> bindings)
       throws CommandException;
}
```

Commands

- Assume we want to call the foo/3 Prolog Predicate
- The first argument is a number and Prolog will return values for the other arguments

```
| ?- foo(1,X,Y).
X = bar,
Y = baz ? ;
no
```

Foo Command

```
public class FooCommand implements IComposableCommand {

  void writeCommand(IPrologTermOutput pto) { // foo(1,X,Y)
     pto.openTerm("foo").printNumber(1).printVariable("X").print
     Variable("Y").closeTerm();
   }

  void processResult(ISimplifiedROMap<String, PrologTerm> b) {
     CompoundPrologTerm x = (CompoundPrologTerm) b.get("X");
     System.out.println(x.getFunctor()); // bar
  }
}
```

IntegerPrologTerm	numbers (Java BigInteger)
CompoundPrologTerm	functor with some arguments (atoms = no args)
ListPrologTerm	Prolog Lists (implements Java List interface)
VariablePrologTerm	must not occur in answers

Mesh-ups

- Implementing new commands is most likely a job for the Düsseldorf team
- But most of the time new Commands are not required
- Instead you write mesh-ups of existing commands

ExploreStateCommand

```
public final class ExploreStateCommand implements IComposableCommand {

  public ExploreStateCommand(String stateID) {
     getOpsCmd = new GetEnabledOperationsCommand(stateId);
     checkInvCmd = new CheckInvariantStatusCommand(stateId);
     allCommands = new ComposedCommand(getOpsCmd, checkInvCmd);
  }

  public void processResult(ISimplifiedROMap<String, PrologTerm> b) {
     allCommands.processResult(b);
     invariantOk = checkInvCmd.getResult();
     enabledOperations = getOpsCmd.getEnabledOperations();
  }

  public void writeCommand(IPrologTermOutput pto){
     allCommands.writeCommand(pto);
  }

}
```

Recipe for Mesh-ups

- Instanciate the commands you want to call
- Stick them into a ComposedCommand
- Delegate writeCommand to the ComposedCommand
- processResult must call the ComposedCommand's processResult first
- Then get the results from the contained commands

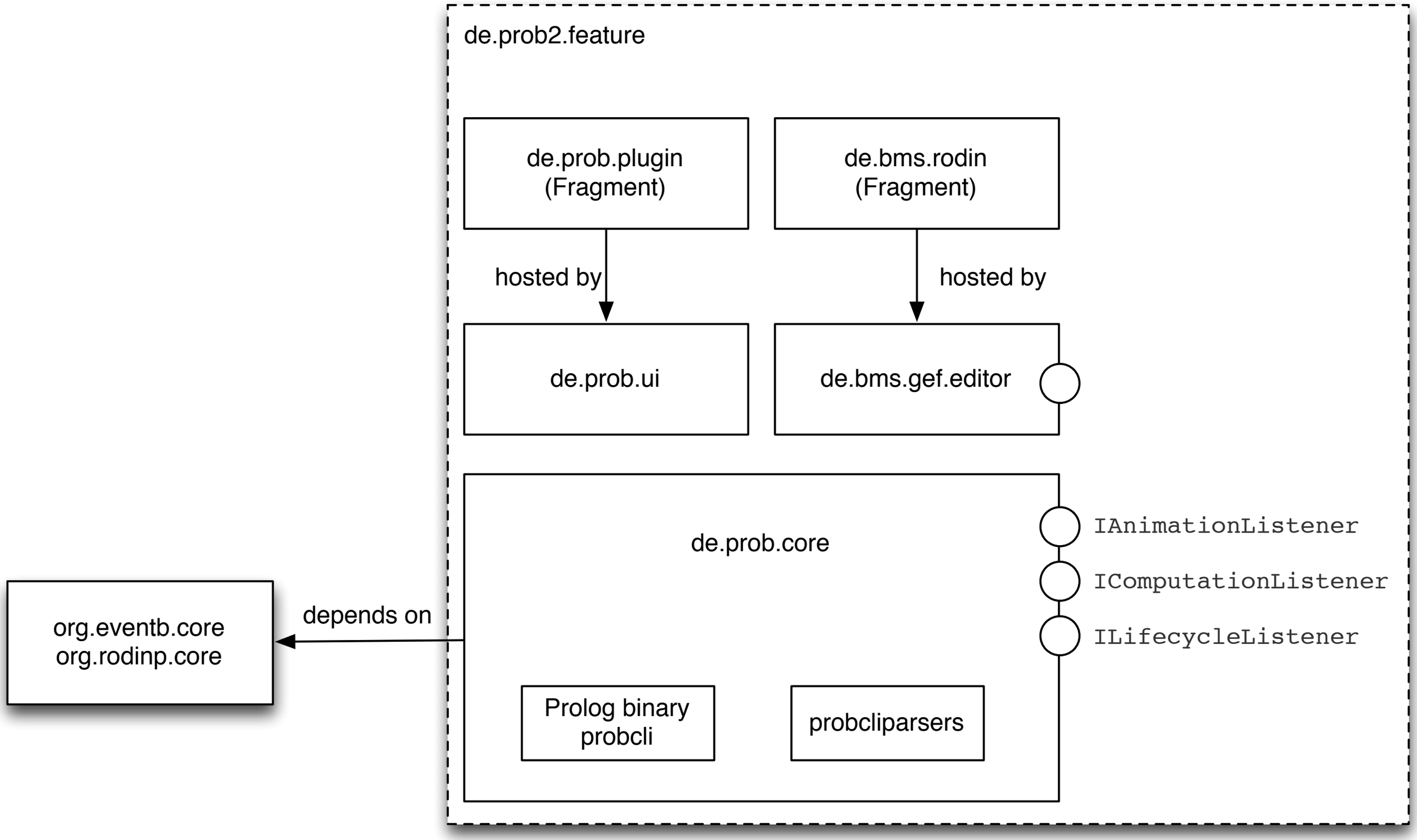
Static methods

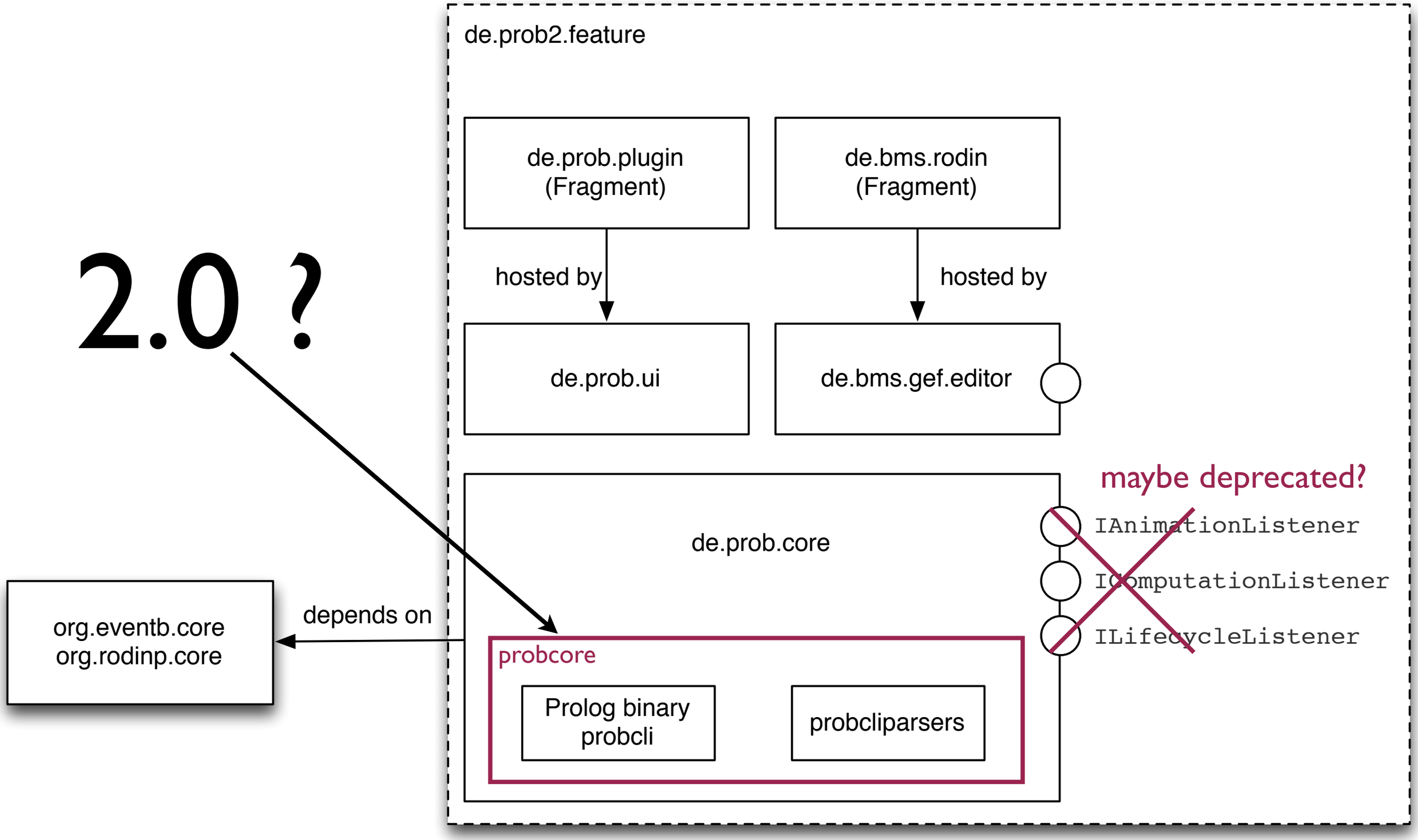
- Most commands have static methods

```
public static State exploreState(Animator a, String id) {
        ExploreStateCommand command = new
                ExploreStateCommand(stateID);
        a.execute(command);
        return command.getState();
}
```

- The idea was to use the static methods as a "DSL"
- Be prepared that these methods will disapear in the future

ProB 2.0 ?


de.prob2.feature
de.prob.plugin
(Fragment)
de.bms.rodin
(Fragment)
hosted by
hosted by
de.prob.ui
de.bms.gef.editor
de.prob.core
IAnimationListener
IComputationListener
ILifecycleListener
org.eventb.core
org.rodinp.core
depends on
Prolog binary
probcli
probcliparsers



At this level of abstraction only minor changes

probcore

- Work in progress (on github: probcore)
- Features
  - Commandline version of ProB + REPL
  - Improved architecture
  - Embedded Groovy + internal ProB DSL
  - State Space Abstraction
  - Support for multiple languages (classical B, CSP, Z, ...)

ProB 2.0 UI

- Will use Eclipse 4.x
- Stand-alone version, but we reuse parts of the implementation for the Rodin plug-in
- The latest improvements will be automatically available in the Rodin plug-in
- Requirements document (draft): http://goo.gl/KS2bh

This afternoon

- Bring your own problem
  Backup plan: Hands-on session
  (http://www.stups.uni-duesseldorf.de/ProB/developer_tutorial/)
- Discussion on ProB 2.0 requirements