Designing a Development Environment for Logic and Multi-Paradigm Programming

Giulio Piancastelli Enrico Denti
{giulio.piancastelli,enrico.denti}@unibo.it

Alma Mater Studiorum—Università di Bologna

3rd Italian Workshop on Eclipse Technologies (EIT 2008)
Bari, November 17-18, 2008

Eclipse for programming languages

Eclipse is well-known as an open and extensible Integrated Development Environment supporting many languages and systems

- Java, C, C++ (with official packaging and download)
- scripting languages such as Python (see PyDev)
- Web programming languages such as PHP (see PHPEclipse)
- niche languages such as Fortran (see the incubating Photran project)

Support for declarative and logic programming is lagging behind, or it is completely lacking

- Prolog?

An opportunity for logic programming

The interaction with the logic interpreter in major Prolog implementations suffers a number of shortcomings and limitations:

- typically reduced to a command-line prompt in a text-based console
- editors and other support tools are built from scratch every time and each follows its own conventions
- useful and innovative instruments are provided only in a few rare cases

The Eclipse platform is an opportunity for logic programming environments to

- exploit a high quality set of tools supporting a huge number of activities
- achieve uniformity by following a series of user interface standards and usage workflow guidelines

tuProlog: the core for a logic and multi-paradigm IDE

tuProlog is an open source light-weight Prolog engine featuring some interesting properties:

- entirely written in Java
- composed by a minimal core engine
- configurable by means of libraries

The choice of tuProlog as a means to analyse and evaluate the integration of a logic-based language and support within Eclipse may lead to:

- easier platform integration
- innovative forms of interaction with the interpreter

In addition to logic programming, tuProlog also supports multi-paradigm Java/Prolog programming, allowing a wider exploration towards a *multi-paradigm integrated development environment*.

Multi-paradigm programming

A programming paradigm is a style to represent/organise computations

- In logic programming, computations are deductions over a knowledge base formed by relations on which queries can be issued
- Object-oriented programming organises computations into hierarchies of entities encapsulating state and methods to manipulate it

Multi-paradigm programming is the integration of two or more paradigms in a unique model that can be realised

- within a single language by following a multi-style approach
- within a system by following a multi-language approach

tuProlog integrates logic and object-oriented paradigms

- at the system level, where Java and Prolog can be combined to build tuProlog libraries
- at the application level, by means of bidirectional Java/Prolog integration

Support for Java/Prolog multi-paradigm programming

Multi-paradigm programming in multi-language fashion needs to be directly supported in its own right by a development environment, next to tools provided for the combined languages on their own

- environments dedicated to each language
- tools to support the combination of the languages at the system level

In the case of tuProlog, a multi-paradigm Java/Prolog IDE needs to supply for

1. Java development
2. Prolog development
3. integration of the two languages by exploiting the library mechanism

Java development

Use the JDT (Java Development Tools)

Prolog development (1/2)

Provide a JDT-comparable set of tools for tasks related to logic programming

- wizards for the creation of projects, theories/modules, predicates with visibility indicator
- outline view to navigate logic theories and modules
  - directives
  - predicates subdivided in facts and rules
- cross-reference view to reach the clauses of a predicate used in a theory but possibly defined in a different module

Prolog development (2/2)

Editing Prolog programs should be supported by

- syntax highlighting
- code completion for predicate invocation plus templates for parameter insertion
- visibility checks (unreachable predicates)
- availability checks (undefined predicates)
- correction suggestions for syntax errors
- logic programming specific syntax warnings
  - e.g. the use of singleton variables within a clause

More advanced features should also be included

- refactoring
- debugging

Java/Prolog integration (1/2)

The integration between the Java side and the Prolog side of the environment needs to be first performed at the *nature* level

Natures in Eclipse

A nature is used to configure the capabilities of a project, typically to associate behaviour and functionalities in the form of builders that process modified files at specific times

The Eclipse concept of natures is the straightforward counterpart of the multiple traits of a programming system involving more than one paradigm

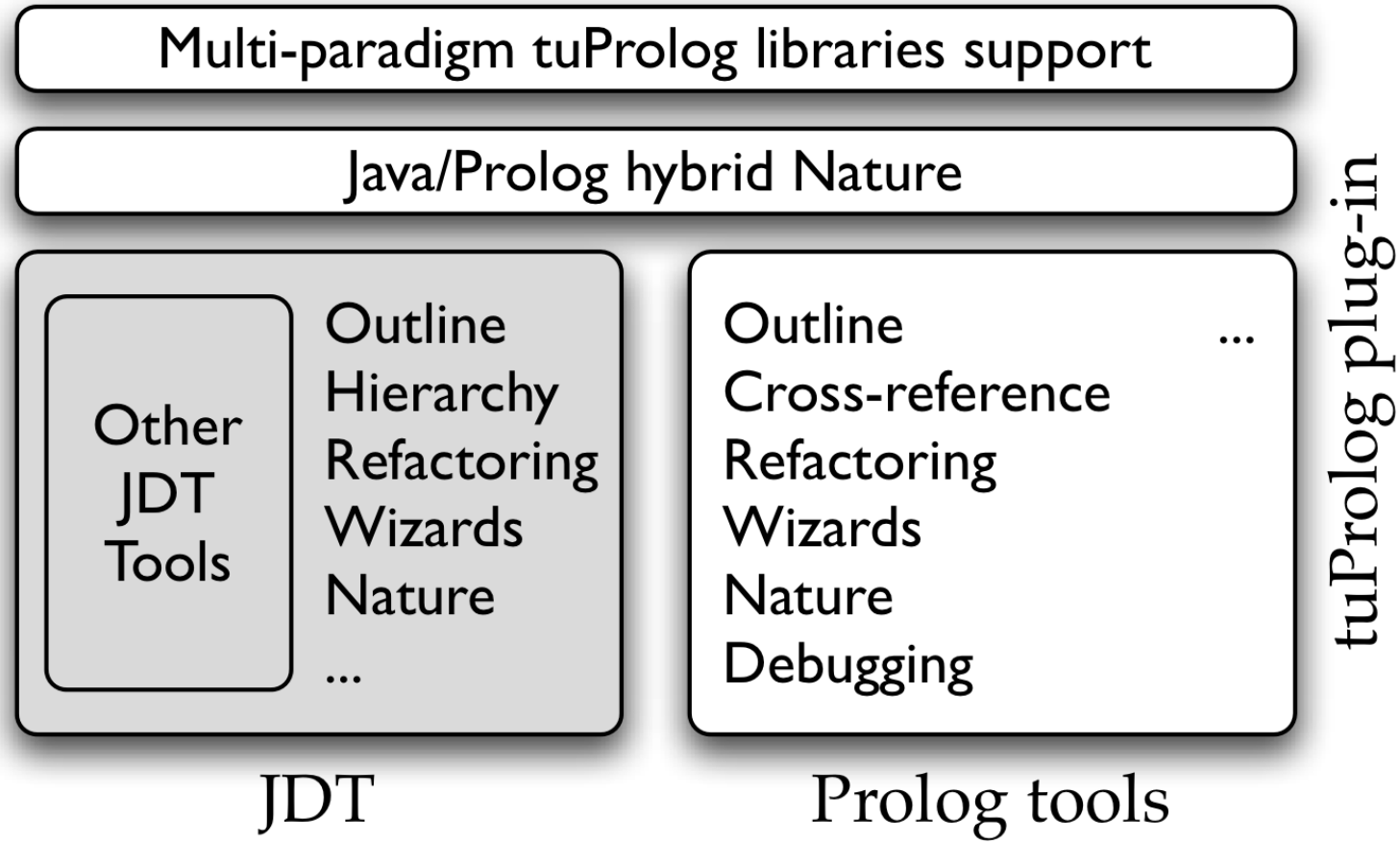
Java/Prolog integration (2/2)

Next to the Java and Prolog nature, we need to provide an additional nature that is able to carry out tasks belonging to the hybrid quality of multi-paradigm programming projects

tuProlog projects

In the case of tuProlog projects, such tasks are

- make Java libraries under development immediately available on the Prolog programming side, without forcing the user to explicitly deal with classpath settings
- supply wizards for the creation of Java/Prolog libraries and new predicates/functors/directives/operators within a library
- instruct the outline view to show the contents of a library as a list of logic programming elements rather than Java fields and methods


Multi-paradigm tuProlog libraries support
Java/Prolog hybrid Nature
Other JDT Tools
Outline
Hierarchy
Refactoring
Wizards
Nature
...
Outline
Cross-reference
Refactoring
Wizards
Nature
Debugging
...
JDT
Prolog tools
tuProlog plug-in

Deployment

The software should be delivered in two different but integrated sets of Eclipse plug-ins

Prolog Development Tools

Form an environment for Prolog programming on their own, and should be made available both as an Eclipse plug-in and a Rich Client Platform application, in order to deploy only the minimal set of Eclipse features to users interested only in Prolog programming

tuProlog plug-in

Comprises the Prolog tools and the multi-paradigm Java/Prolog functionalities based on tuProlog, and has to be deployed only as a platform extension, so as to exploit the JDT already bundled with the standard Eclipse download

Components of the tuProlog plug-in for Eclipse

The current prototype of the tuProlog plug-in for Eclipse is composed of a single feature comprising three different components

1. `alice.tuprolog` makes tuProlog available as a software library within the platform
2. `alice.tuprologx.eclipse` contains everything related to the interaction between tuProlog and Eclipse, including the user interface
3. `alice.tuprologx.eclipse.doc` integrates the documentation (a getting started guide for the plug-in and relevant parts of the tuProlog manual) with the Eclipse help system

Elements in `alice.tuprologx.eclipse`

The `alice.tuprologx.eclipse` component provides

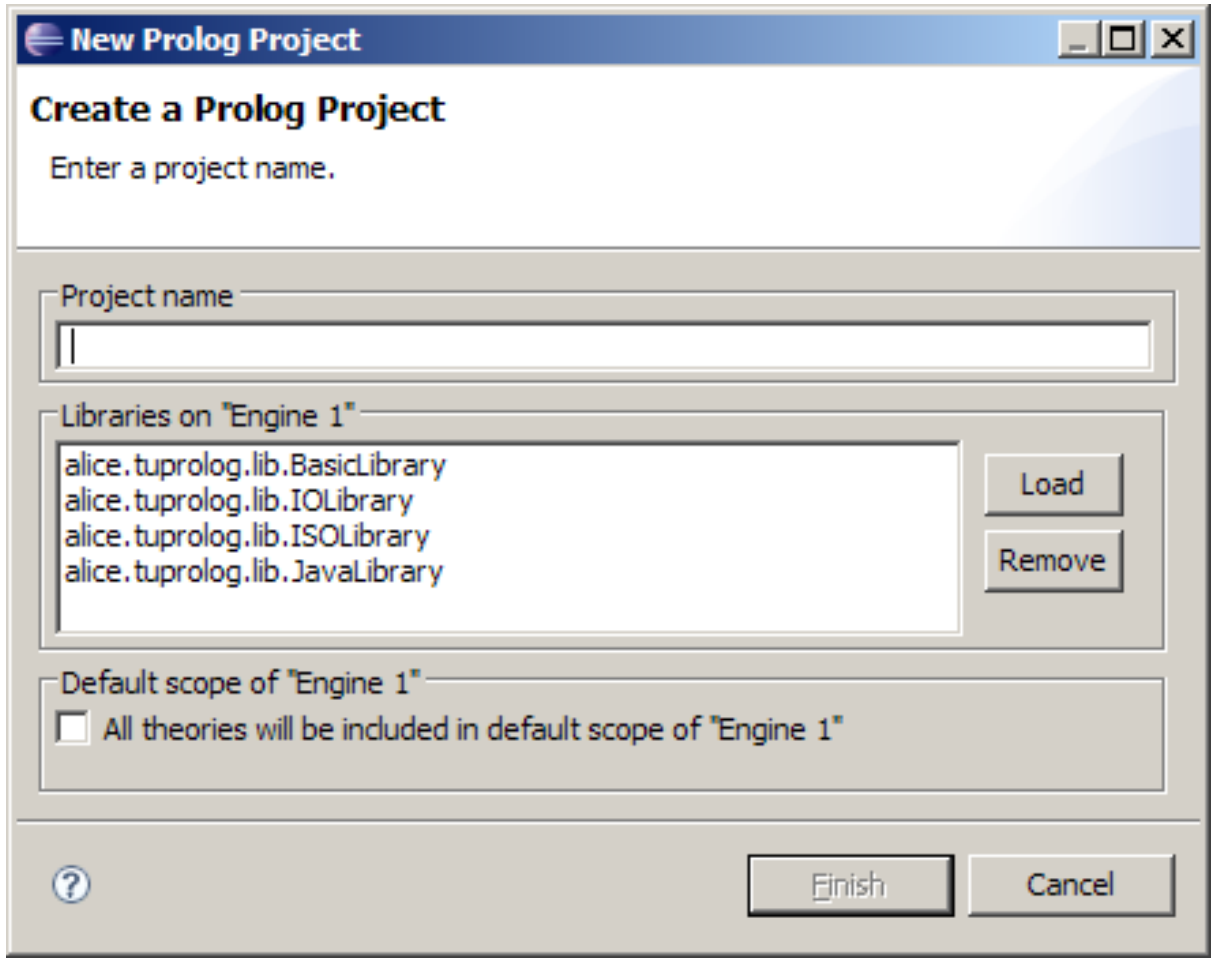
- a perspective
- two wizards to help create projects and logic theories, also invocable from buttons in a toolbar
- a nature and a builder associated to Prolog projects
- an editor with syntax highlight and a basic form of syntax checking for the Prolog language
- property pages to change editor options (such as font and colours) and manage the possibly many tuProlog engines associated with a single project

Views in `alice.tuprologx.eclipse`

The `alice.tuprologx.eclipse` component also provides four views to display information related to Prolog development and the execution of Prolog programs

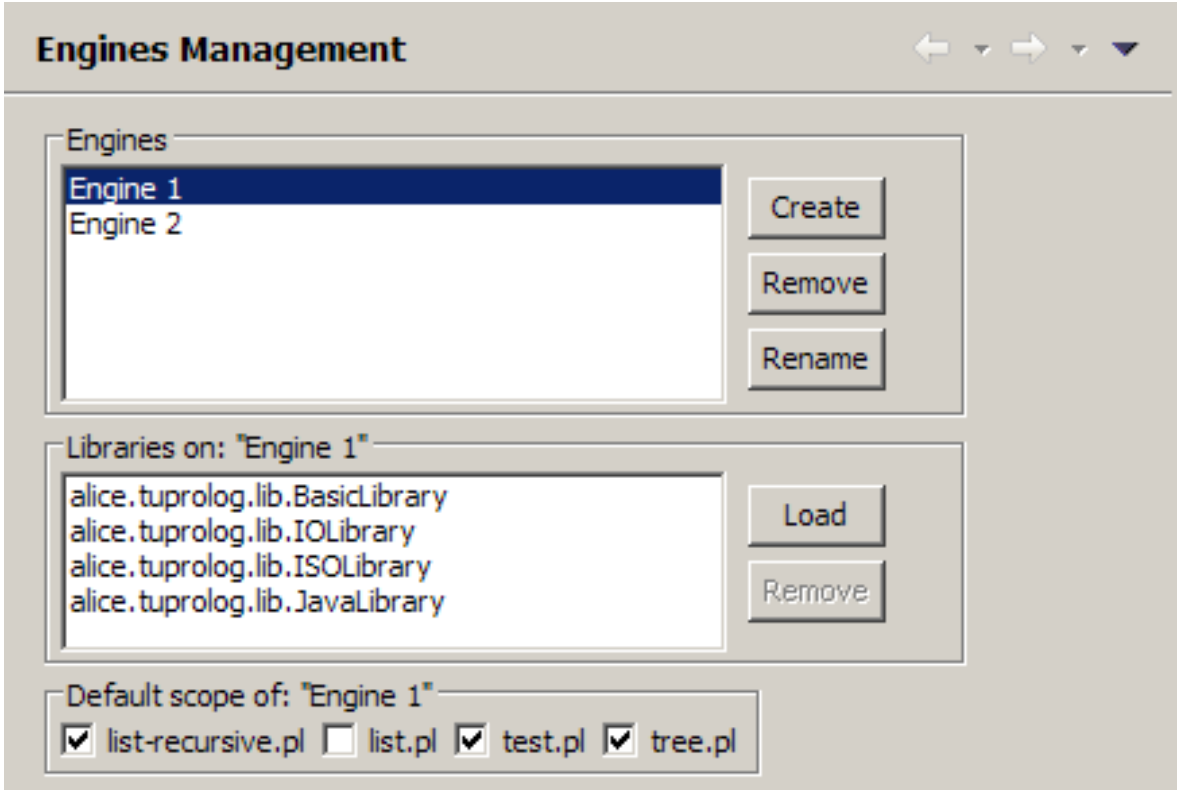
- the **Clause** view accesses the AST of the Prolog program in the selected editor buffer to show its outline in terms of clauses, each identified by its head
- the **Theory** view displays at a glance the content of all the tuProlog engines associated with the current project in terms of logic theories
- the **Query** view provides the history of the logic queries asked within the context of the current project
- the **tuProlog Console** view allows to select one of the possibly many tuProlog engines that have been asked a query and see the solution, output text, and tracing information derived by standard predicates such as `spy/0`

The first engine configuration occurs in the Prolog project wizard



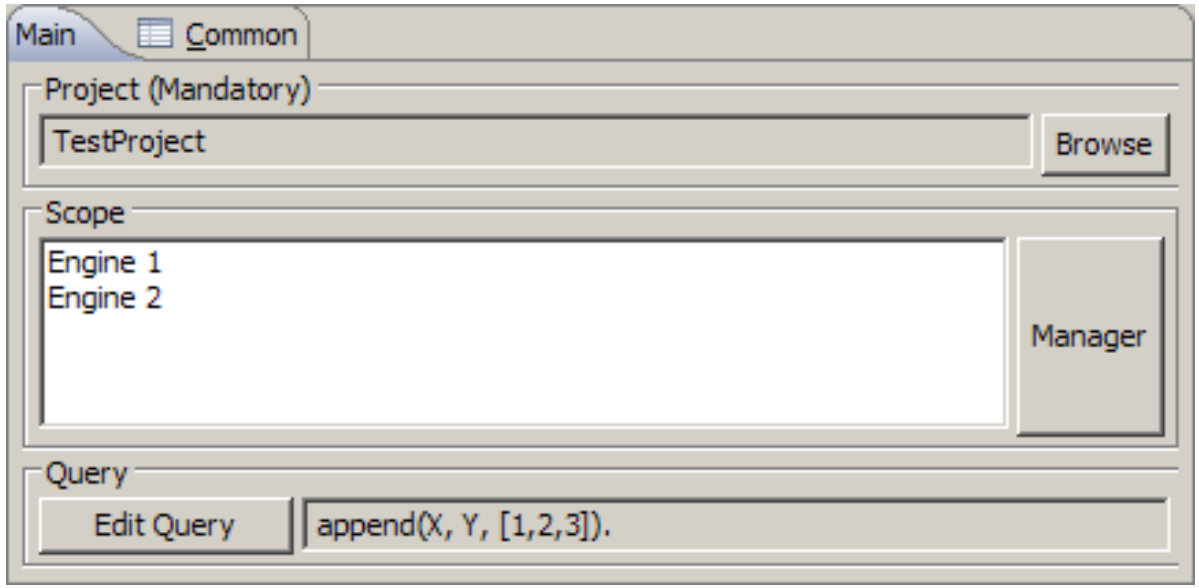
Any Prolog project comes with an associated tuProlog engine that the wizard offers an interface to configure in terms of libraries and theories

After creating a Prolog project and logic theories, developers may add other tuProlog engines to the project and modify their configurations

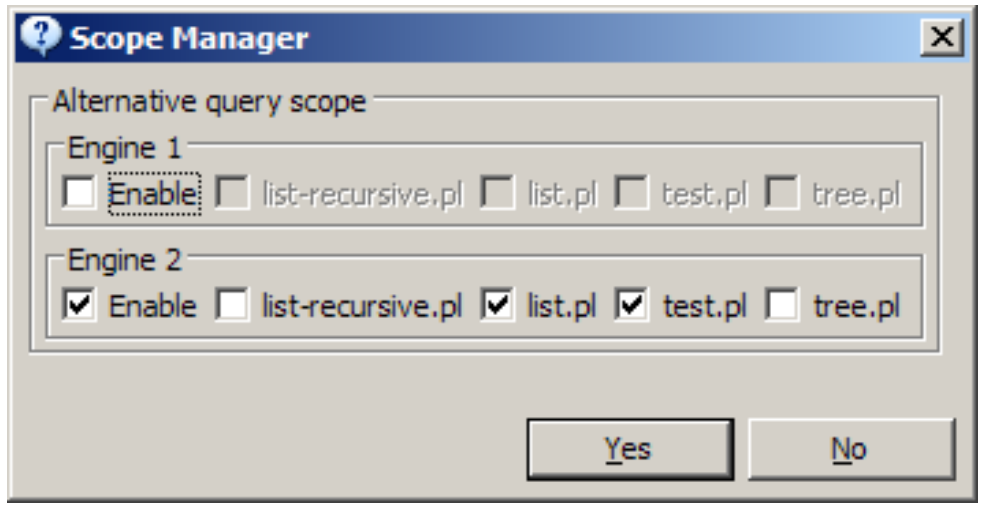


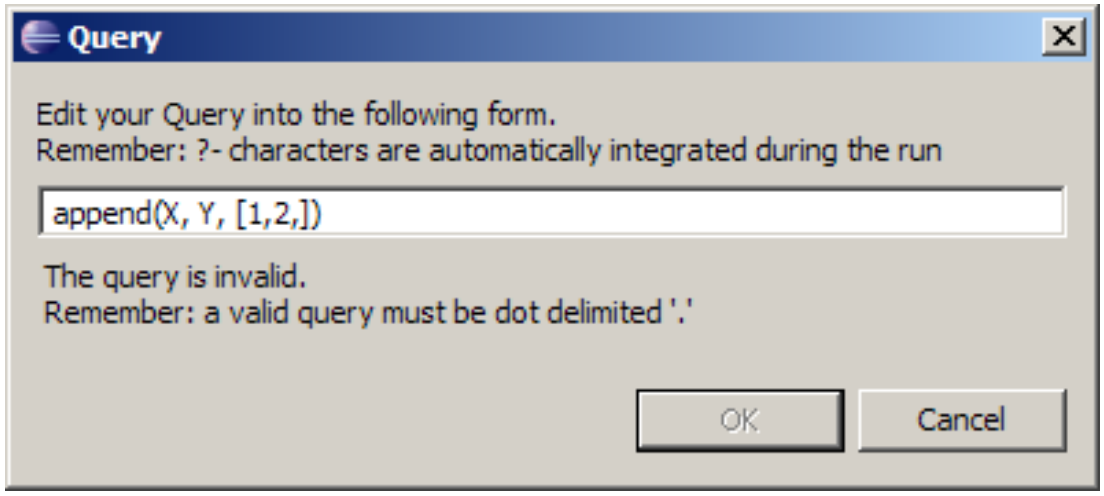
In particular, it is possible to finely set the *default scope* of an engine in terms of the logic theories contained in the project

Queries are entered through the launch configuration dialog provided by Eclipse, that the tuProlog plug-in extends to introduce its own launch configuration type



The scope of the query for each available engine and the query itself are separately set in their own dialog window


Scope Manager
Alternative query scope
Engine 1
Enable
list-recursive.pl
list.pl
test.pl
tree.pl
Engine 2
Enable
list-recursive.pl
list.pl
test.pl
tree.pl
Yes
No


Query
Edit your Query into the following form.
Remember: ?- characters are automatically integrated during the run
append(X, Y, [1,2,])
The query is invalid.
Remember: a valid query must be dot delimited '.'
OK
Cancel

A minimal example in a teaching setting

Show the effect of different implementations for the `member/2` predicate

File `list-1.pl`

```
member(E, [E | _]).
member(E, [_ | L]) :- member(E, L).
```

File `list-2.pl`

```
member(E, [H | _]) :- E == H, !.
member(E, [_ | T]) :- member(E, T).
```

Given `engine-1` with theory `list-1.pl` and `engine-2` with theory `list-2.pl`, executing the query

```
?- member(1, [1, 0, 0, 1, 0, 1]).
```

will yield three solutions for `engine-1` but just one solution for `engine-2`

Conclusions and future work

We have used Eclipse and tuProlog to start promoting innovation in the field of logic programming development environments

- introduced interface standards and workflow guidelines
- allowed for a flexible and effective interaction with a multiplicity of independently configured Prolog interpreters

In the immediate future we will pursue

- a deeper integration of the existing tuProlog plug-in tools
  - better mechanisms to configure tuProlog engines and browsing their knowledge bases
  - platform-wide access to the Prolog AST
- extension towards the multi-paradigm side of the environment