Developing Protégé Plugins

Ray Fergerson

Stanford

Overview

- What is a Plugin?
- How Plugins work
- Plugin Types and Capabilities
- Development Tips
- Packaging
- Bundling
- Coming Changes

Out of Scope

- Standard Java
  - Coding
  - Packaging (jars)
  - Utilities
- Development environments
- Licensing (see FAQ)
- Non-plugin extensions

What is a Plugin?

- Extension to Protege
  - Requires no source code modifications
  - Loaded and managed by system
- Typically the implementation of a Java *interface* and an entry in a Java manifest file.
- Typically packaged as one or more jar files and installed in a subdirectory of the Protege *plugins* directory

How Plugins work

- System looks at all manifests available:
  - On the classpath
  - In directories one level down from *plugins*
    - all jars
    - File meta-inf/manifest.mf
- System looks for a manifest entry identifying a plugin and loads the referenced class
- System creates an instance of the class as needed

Types of Plugins

- TabWidget
- SlotWidget
- KnowledgeBaseFactory (“Backend”)
- ProjectPlugin
- ExportPlugin
- ImportPlugin

Plugin: TabWidget

- What is it?
  - Large piece of screen real-estate
  - Can interact with domain KB

    browse, change, delete, corrupt
- What are its limitations?
  - Difficult to supplement or even interact with other tabs
- How hard is it to create?
  - Easy (1 day)

TabWidget Example

(for all example code see

http://protege.stanford.edu/doc/pdk)

Plugin: SlotWidget

- What is it?
  - UI Control which allows the user to display and modify a slot value
  - Follows a protocol for hiding interaction KB
- What are its limitations?
  - Works best with a *single* slot
- How hard is it to create?
  - Easy (1 day)

SlotWidget Example

(for all example code see

http://protege.stanford.edu/doc/pdk)

Plugin: KnowledgeBaseFactory

- What is it?
  - Replacement for the standard storage mechanisms with
    - Database
    - External server
    - ...
  - Allows for parsing of different file formats
- What are its limitations?
  - Difficult to manipulate UI
  - Implementations tend to be buggy
- How hard is it to create?
  - Hard (>= 1 month)
  - Consider Import/Export plugin instead

KnowledgeBaseFactory Example

(for all example code see

http://protege.stanford.edu/doc/pdk)

Plugin: ProjectPlugin

- What is it?
  - Code that executes when “things happen” to a project (create, load, display, close, etc)
  - Get access to project, view, menu bar, tool bar and can modify them as you like
- How hard is it to create?
  - Easy (1 day)

ProjectPlugin Example

(for all example code see

http://protege.stanford.edu/doc/pdk)

Plugin: ExportPlugin

- What is it?
  - Code that saves (part of) a knowledge-base in any given format to *somewhere else*
    - files, servers, web, …
  - No change of the current backend
  - No guarantee of “round trip” (export->import)
  - No “live” connection
- How hard is it to create?
  - Medium (1 week)

ExportPlugin Example

(for all example code see
  http://protege.stanford.edu/doc/pdk)

Plugin: ImportPlugin

- What is it?
  - Code that creates a knowledge-base from information from *somewhere else*
    - files, servers, web, …
  - No change of the current backend
  - No guarantee of “round trip” (export->import)
  - No “live” connection
- How hard is it to create?
  - Medium (1 week)

ImportPlugin Example

(for all example code see

http://protege.stanford.edu/doc/pdk)

Development Tips

- To ease integration with a debugger
  - implement a main() method
- To avoid making a jar while debugging
  - Put your meta-inf/manifest.mf file on the classpath
- To access icons from your code:
  - use FileUtilities.loadImageIcon()
- Access the plugin’s directory for config files
  - PluginUtilites.getInstallationDirectory()
- Watch out for caching!

Packaging

- Create a directory structure like:

  edu.stanford.smi.protegex.myproject/

  ```
      myproject.jar
      myproject_doc.html
      myproject_about.html
      plugin.properties
  ```

- Zip it up

Packaging II

- Sample Plugin properties file

```
plugin.component.count=1
plugin.component.name.0=PROMPT tab
plugin.component.about.0=about_prompt.html
plugin.component.doc.0=doc/index.html
plugin.dependency.count=1
plugin.dependency.0=edu.stanford.smi.protegex.owl
```

Bundling

- Plugins can be “bundled” with the full release and made available to all users
- Advantage:
  - You may get a lot of users quickly
- Disadvantage:
  - You may get a lot of users quickly
- In order to be bundled the plugin must be:
  - *Well Formed*
  - *Well Behaved*
  - *Well Maintained*

Bundling II

- *Well Formed*
  - jar file in an appropriate, recognizable directory
    - appropriate: “edu.myschool.mygroup.myproject”, not “foo”
    - recognizable: last directory element: “mytab” not “foo”
  - About Box and Documentation entries
  - Minimal size
    - minimal documentation
      - links to more extensive documentation on web
      - no PDF, MS Word, large image files
    - no source
    - at most one small example project
    - readme.txt file if necessary
  - isSuitable implemented if appropriate
    - requires certain sorts of projects or additional installation (shared libraries, etc)

Bundling III

- *Well Behaved*
  - Must “work” (not crash on startup) with the current release
  - Minimal information (just errors) printed to the console window
    - Single startup line is ok (but certainly not required)
    - No tracing
  - Must start up and shut down smoothly
    - No time consuming code executed in static initializer
    - No long start up delays or modal dialogs that block the rest of the system
    - Must free acquired resources in “dispose()”
- *Well Maintained*
  - Developer/maintainer “responsive” to problems.

Coming Changes

- **Nothing major!**

- Additional fixes to class loader mechanism
- Allow users to disable installed plugins
- Additional optional “static interface” methods:
  - isSuitable() for other plugin types
  - buildString() for macro substitution on About Box page
- Optional localization support for plugins
- Documented procedures for bundling

Summary

Plugins provide flexible and powerful mechanisms for extending Protege in many ways.

Go do something interesting!

(Think about contributing it back to the community.)