Stalin-Core-Plugin mit JPF

Robert Jakob, Nicola Sheldrick

Fakultät für Angewandte Wissenschaften
Albert- Ludwigs- Universität Freiburg

November 28, 2006

Warum JPF und nicht RCP?

- geringere Komplexität
- bessere Wartbarkeit
- Funktionalitätsrahmen für Stalin gegeben

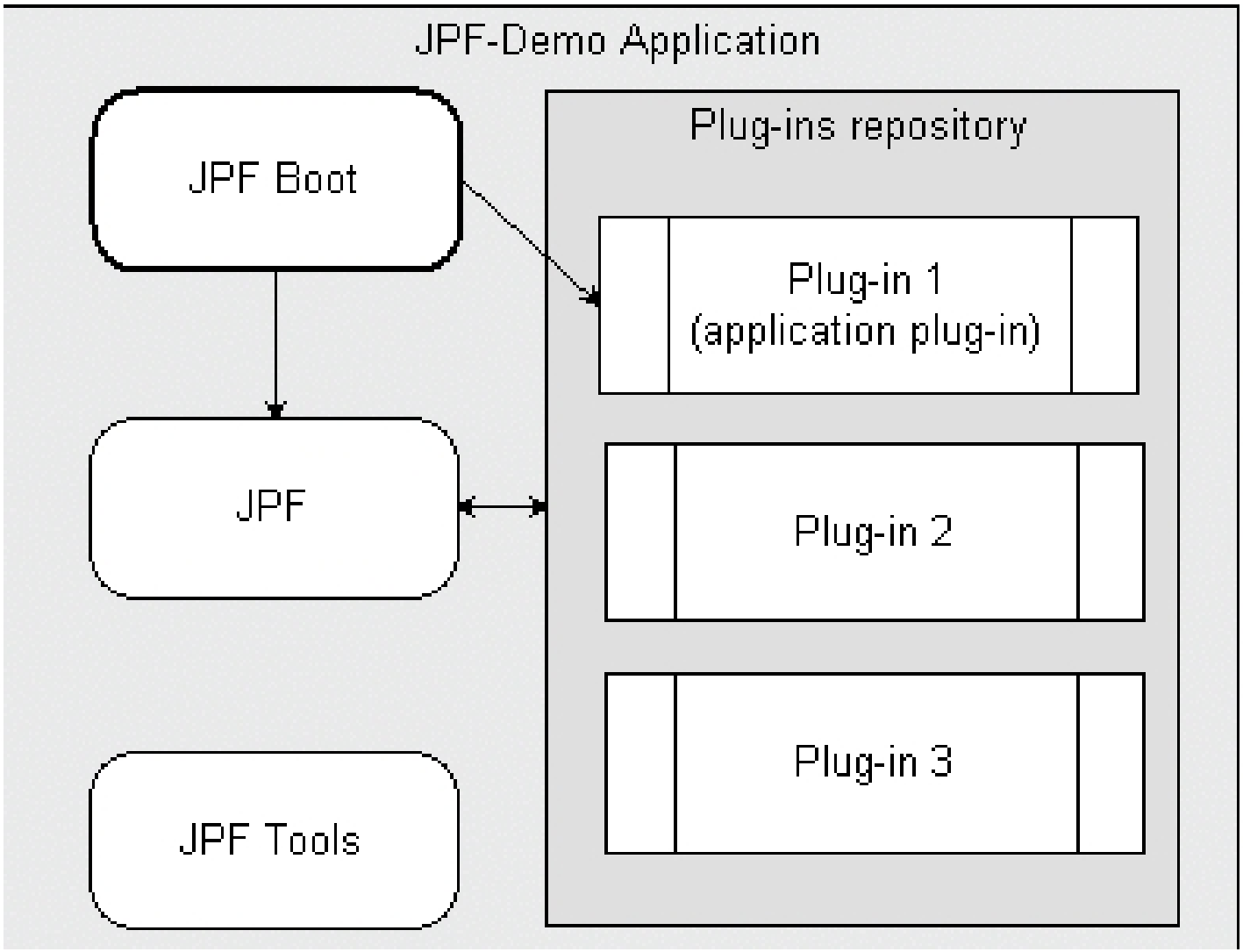
Beispielstruktur einer einer Anwendung mit JPF

StalinCoreApp/

```
+- data/
+- lib/
    +- commons-logging.jar
    +- jpf.jar
    +- jpf-boot.jar
    +- jpf-tools.jar
    +- jxp.jar
    +- log4j.jar
+- logs/
+- plugins/
+- boot.properties
+- log4j.properties
+- run.bat
+- run.sh
```

Erläuterung

data	in diesem Ordner werden die Konfigurations- und andere Dateien der Plug-ins abgelegt
lib	JPF-Bibliotheken, sowie die von JPF benötigten Bibliotheken für Logging.
logs	Ordner für Log-Dateien
plugins	Ordner für die JPF-Plugins
boot.properties	die Boot Config
run.bat	Start-Skript Windows
run.sh	Start-Skript Unix


JPF-Demo Application
JPF Boot
JPF
JPF Tools
Plug-ins repository
Plug-in 1
(application plug-in)
Plug-in 2
Plug-in 3

Starten der Anwendung

1. das run-Skript ruft die main-Methode org.java.plugin.boot.Boot.main(String[]) der JPF-Boot Bibliothek auf
2. die Main-Methode liest die Config boot.properties aus, initialisiert das JPF Framework und läd alle Plugins aus dem plugins-Ordner
3. zum Schluß ruft sie das StalinCoreApp Plug-in auf, da wir dieses in der Config-datei boot.properties angeben
4. Sobald JPF das Core Plug-in aufgerufen hat geht die gesamte Kontrolle der Anwendung an das Plug-in StalinCoreApp über.
5. Die Plug-in Klasse StalinCoreApp erbt von der speziellen abstrakten Klasse org.java.plugin.boot.ApplicationPlugin der JPF Boot-Bibliothek.

StalinCoreApp

Unser StalinCoreApp besteht, sowie jedes JPF Plug-in, aus zwei Teilen:

1. der Manifest-Datei
2. Plug-in Java Code

Wir schauen uns die beiden mal genauer an.

Plug-in manifest

- XML-Datei

Schlüssel-Tag

<plugin id="StalinCoreApp" version="0.0.1"
class="StalinCoreApp.CorePlugin">

- die Plug-in ID ist "StalinCoreApp" und die Versionsnummer ist "0.0.1" .
- Wir deklarieren, dass unser Plug-in eine "Plug-in Klasse" besitzt, nämlich StalinCoreApp.CorePlugin,

Plug-in manifest

- Die "Plug-in Klasse" ist ein optionales Element in der Plug-in Deklaration und nur nötig, wenn während des Startes etwas in der doStart()- oder der doStop()-Methode aus dem Plugin-Interface ausgeführt werden soll.
- Lassen wir diese Deklarierung weg, also wenn während der Plug-in Aktivierung/Deaktivierung nichts spezielles gemacht werden muss, benutzt JPF eine Standard Plug-in Klasse.
- Beim Core Plug-in ist dies jedoch nicht der Fall, denn dieses besondere Plug-in ist der Anwendungs-Einstiegspunkt und muss bei seiner Aktivierung die GUI aufbauen und verwalten.

Deklarierung der Bibliotheken

```
<runtime>
    <library id="core" path="classes/" type="code">
        <export prefix="*" />
    </library>
    <library type="resources" path="icons/" id="icons">
        <export prefix="*" />
    </library>
</runtime>
```

Deklarierung der Bibliotheken

Hier definieren wir :

1. Alle Java .class-Dateien dieses Plug-ins werden im Ordner "classes/" innerhalb des zugehörigen Plug-in Ordners abgelegt.
2. Alle Klassen und Pakete (*) sind sichtbar fr andere Plug-ins, und diese können unseren Code frei verwenden.
3. Es gibt einen Ressources-Ordner "icons/", welcher ebenfalls für andere Plug-ins sicht- und benutzbar ist.

Deklaration eines Extension-Points

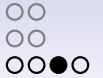
Mächtigste Eigenschaft von JPF

```
<extension-point id="Core">
   <parameter-def id="class" />
   <parameter-def id="name" />
   <parameter-def id="description" multiplicity="none-or-one" />
   <parameter-def id="icon" multiplicity="none-or-one" />
</extension-point>
```

Deklaration eines Extension-Points

Hier definieren wir :

1. Unser Core Plug-in veröffentlicht einen Punkt , an dem es durch andere Plug-ins erweitert werden kann.
2. Wir nennen diesen Punkt "Core" und erklären, dass Erweiterungen an diesem Punkt als "Tab" in der GUI dargestellt werden.
3. Außerdem sollte jedes Plug-in das an diesem Punkt festmacht verschiedene Parameter zur Verfügung stellen, welche teils in der GUI benutzt werden, teils dazu um mit dem Plug-in zu kommunizieren (bsp. eindeutiger Namen).



mögliche Parameter für einen Extension Point

class Dies ist ein benötigter Parameter vom Typ String, er sollte den kompletten Java Klassennamen enthalten.

name Der Name der App, er wird als Tabname in der GUI zu sehen sein.

description Die App-Beschreibung, welche als "Tab-Hint" in der GUI gezeigt werden wird. Dies ist ein optionaler Parameter.

icon Der Dateiname des App-Icons. Dies ist ein optionaler Parameter.