



Java Community Packaging : a JPackage Perspective

Fernando Nasser and Paul Nasrat

Agenda

- jpackage.org
- The Problem
- Our Solution
- Difficulties
- Future

jpackage.org

- Goals
 - To provide a coherent set of Java software packages for Linux
 - To establish an efficient and robust policy for Java software installation
- Who
 - Independent Volunteers
 - BEA, Red Hat, Suse, Mandriva and others
 - Installation practices adopted by Sun, Debian, IBM etc.
- URL
 - <http://jpackage.org>

The Problem

- Monolithic applications
 - Large tar balls
 - Full of binary code in the form of jar files
 - Dependencies on other software not explicit for both build and runtime
 - Building details not always well documented and requiring revisiting every time one wants to build a specific software
 - Builds not audited; binaries from unknown origin may pose security threats or include IP violations

The problem

- Many copies of the “same” jar package scattered through the system
 - different versions
 - unknown origin
 - no source code
- Many versions of the same class
 - CLASSPATH order matters
 - JVM-specific classes sometimes sneak-in
- Different JVMs with
 - different installation procedures and locations
 - non-trivial install, non-automatic updates
 - different setup and different requirements

Our Solution

- Common packaging and installation of available JVMs (BEA, IBM, Sun, ECJ/GCJ)
- Install jar files in central location and share
- Build (when possible) the jar files from source as independent “packages”
- Add version to jar file names (in addition to unversioned link)
- Use native package manager to maintain the dependencies among packages (including version requirements)
- Use alternatives or wrapper scripts to solve different JVM requirements or for version coexistence
- Provide scripts for handling references to jar files

Our Solution

<http://jpackage.org/>

Our Solution

Example: tomcat5.spec

Our Solution

■ Alternatives

- Created by Debian but available on all Linux distributions
- Allow installing in parallel different versions of a software or even software from different providers, and allow part of the differences to be accounted for
- Launching scripts combined with alternatives makes switching possible
- System-wide selection (by root), but can be overridden by `JAVA_HOME` most of the time
- Links to specific SDK versions provided

Our Solution

■ Advantages

- Easier to install and update
- Can switch JVM providers and versions
- All sources available (except for non-free packages)
- Can replace non-free dependencies with Open Source equivalents one at a time
- Central point of coordination of releases for all projects
- Uniform Java installation and use among distributions
- Improved visibility of projects and incentive for people to try them (due to easy of install/uninstall)

Our Solution

- Advantages (cont.)
 - Spec files work as a knowledge repository for building details, system specific customizations etc.
 - Allows backporting of fixes through patches applied during build, which remain documented in the spec file
 - SRPM contains everything about the software like sources, fixes, how to build and allow reproduction of build
 - Any person can rebuild as the knowledge is stored

Difficulties

- How to know the ever changing dependencies of the upstream packages at build and run time?
 - jar files have no version
 - jar files are binary (no sources)
- API changes without backward compatibility and delay in reworking the code by some upstream projects
 - forces to maintain old versions in parallel with the new one or to drop the project
- Use of modified or non-standard releases of dependencies, or dependencies that do not have releases properly tagged
- Dependencies on non-free packages

Difficulties

- Need to extend to other packaging systems besides RPM
- Dependencies of jar files cannot be handled by MANIFEST.MF Class-Path; no solution in this version but Sun is working on a proposal for Java 6
- Some packages have problems with the square-brackets syntax used for links so this is being gradually phased out

Future

- Metadata-based jar repositories
- Replacements for non-free packages from Classpath[X] and Geronimo (among others) (MOSTLY DONE)
- Open Source JVM (ECG/GCJ) (ALREADY WORKS)
- Better support from upstream projects:
 - metadata about external jar files (MAVEN)
 - sources and external jars in separate tar balls
 - properly marked releases of dependencies
 - faster replacement of deprecated APIs, bits from dead projects
 - avoidance of non-free dependencies or JVM-specific code
 - build and test jpackaged version upstream
- LSB 4?

Thank You

fnasser@redhat.com

nasrat@redhat.com

jpackage-discuss@jpackage.org