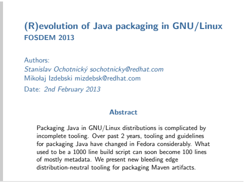
(R)evolution of Java packaging in GNU/Linux



(R)evolution of Java packaging in GNU/Linux
FOSDEM 2013

Authors:
Stanislav Ochotnický sochotnicky@redhat.com
Mikołaj Izdebski mizdebsk@redhat.com

Date: *2nd February 2013*

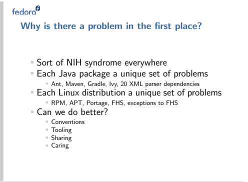
Abstract

Packaging Java in GNU/Linux distributions is complicated by incomplete tooling. Over past 2 years, tooling and guidelines for packaging Java have changed in Fedora considerably. What used to be a 1000 line build script can soon become 100 lines of mostly metadata. We present new bleeding edge distribution-neutral tooling for packaging Maven artifacts.

Just our introduction. We have been working in Java/Maven packaging for Fedora and Red Hat Enterprise Linux for several years. We like to make things simple(r).

(R)evolution of Java packaging in GNU/Linux
└─Overview

└─Why is there a problem in the first place?



fedora

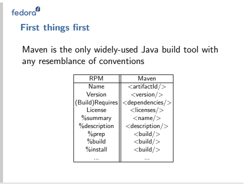
Why is there a problem in the first place?

- Sort of NIH syndrome everywhere
- Each Java package a unique set of problems
  - Ant, Maven, Gradle, Ivy, 20 XML parser dependencies
- Each Linux distribution a unique set of problems
  - RPM, APT, Portage, FHS, exceptions to FHS
- Can we do better?
  - Conventions
  - Tooling
  - Sharing
  - Caring

NIH - distributions, java developers, everyone is guilty. If we manage to provide a proper tooling a lot of things will get better as a result eventually.

(R)evolution of Java packaging in GNU/Linux
└─Overview

└─First things first



fedora

First things first

Maven is the only widely-used Java build tool with any resemblance of conventions

RPM	Maven
Name	<artifactId/>
Version	<version/>
(Build)Requires	<dependencies/>
License	<licenses/>
%summary	<name/>
%description	<description/>
%prep	<build/>
%build	<build/>
%install	<build/>
...	...

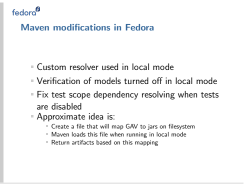
Most things are very similar. But there are exceptions:

- Not all metadata is 1:1
- No exclusions in RPMs
- No equivalent of Maven scope in RPMs
- Parent pom inheritance missing
- Optional dependencies

The problem with other build systems is that there is no way to standardize parsing and handling of their metadata. They can be spread in many places.

(R)evolution of Java packaging in GNU/Linux
└─History lessons

└─Maven modifications in Fedora

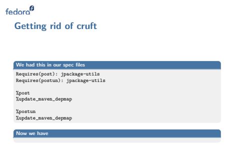


fedora

Maven modifications in Fedora

- Custom resolver used in local mode
- Verification of models turned off in local mode
- Fix test scope dependency resolving when tests are disabled
- Approximate idea is:
  - Create a file that will map GAV to jars on filesystem
  - Maven loads this file when running in local mode
  - Return artifacts based on this mapping

Maven and Java stack largely based on JPP. It is our heritage, but we are changing it bit by bit. Our patches are not very welcome by the upstream, but we are getting rid of them.



fedora

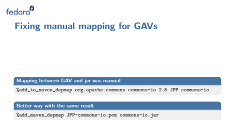
Getting rid of cruft

We had this in our spec files

```
Requires(post): jpackage-utils
Requires(postun): jpackage-utils

%post
%update_maven_depmap

%postun
%update_maven_depmap
```

Now we have

These snippets used to produce one big mapping file out of small xml-like files created by every Maven package. Maven then read this one big file. We moved to reading those small files and with this we didn't have to create the big file any more. Performance hit is negligible.



fedora

Fixing manual mapping for GAVs

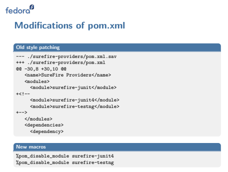
Mapping between GAV and jar was manual

```
%add_to_maven_depmap org.apache.commons commons-io 2.5 JPP commons-io
```

Better way with the same result

```
%add_maven_depmap JPP-commons-io.pom commons-io.jar
```

This has been achieved by reading metadata from pom.xml instead of relying on manual inspection. We managed to get rid of a lot of errors this way



fedora

Modifications of pom.xml

Old style patching

```
--- ./surefire-providers/pom.xml.sav
+++ ./surefire-providers/pom.xml
@@ -30,8 +30,10 @@
   <name>SureFire Providers</name>
   <modules>
     <module>surefire-junit</module>
+<!--
     <module>surefire-junit4</module>
     <module>surefire-testng</module>
+-->
   </modules>
   <dependencies>
     <dependency>
```

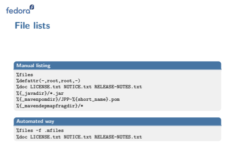
New macros

```
%pom_disable_module surefire-junit4
%pom_disable_module surefire-testng
```

There are other macros:

- adding/removing dependencies
- modifying plugins
- injecting/removing any xml parts

All in all they simplify updates and are more reliable than patches since they know about XML structure



fedora

File lists

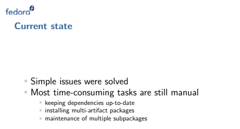
Manual listing

```
%files
%defattr(-,root,root,-)
%doc LICENSE.txt NOTICE.txt RELEASE-NOTES.txt
%{_javadir}/*.jar
%{_mavenpomdir}/JPP-%{short_name}.pom
%{_mavendepmapfragdir}/*
```

Automated way

```
%files -f .mfiles
%doc LICENSE.txt NOTICE.txt RELEASE-NOTES.txt
```

These file lists are relatively limited in that they only handle files that %add_maven_depmap macro knows about: poms, jars, mapping files. They still simplify spec files considerably


fedora

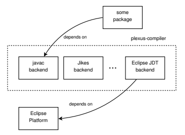
Current state

- Simple issues were solved
- Most time-consuming tasks are still manual
  - keeping dependencies up-to-date
  - installing multi-artifact packages
  - maintenance of multiple subpackages

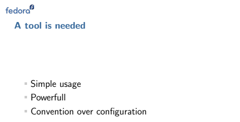
Maintenance of multiple subpackages is always a burden. Dependencies get more complicated, spec files are much longer and their updates more error-prone.

fedora

Plexus-compiler example



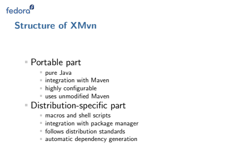
Maintenance of multiple subpackages is burdensome so it's not done usually. This causes issues when one of the artifacts pulls in big dependency tree. This is a recurring problem we need to solve. One of more recent examples was Freemind which pulled in big part of eclipse-platform in Fedora.


fedora

A tool is needed

- Simple usage
- Powerfull
- Convention over configuration

As shown in previous exmaples, currect situation of packaging of Maven artifacts in Fedora requires a new tool. A tool that would be simple to use, doing most of the tasks for users. It should be powerfull enough to allow migration of all possible spec files to the new style of packaging, not only some of them. It should also utilize the convention over configuration rule so that most simple packages have the simpliest specx files possible, but more complicated cases can be handled with customizations.

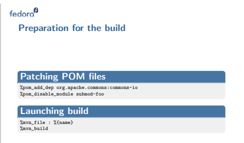

fedora

Structure of XMvn

- Portable part
  - pure Java
  - integration with Maven
  - highly configurable
  - uses unmodified Maven
- Distribution-specific part
  - macros and shell scripts
  - integration with package manager
  - follows distribution standards
  - automatic dependency generation

XMvn consists of two parts, portable part and distribution-specific part. The first one is written in pure Java. It is a set of extensions to (otherwise unmodified) Apache Maven and is licensed in consistent way with Maven (the license is Apache License version 2.0). The portable part has many configuration options and it tries not to rely on any distribution-specific characteristics, so it should be possible to use it on any GNU/Linux or Unix distribution.

The distribution-specific part forms an interface between the way how packages are built in distributions and the portable part. For this part is implemented only for Fedora, but different layers could be created for different distributions, also those not based on RPM. Among other things this part is responsible for keeping distribution-specific defaults and automatic dependencies are generated.



fedora

Preparation for the build

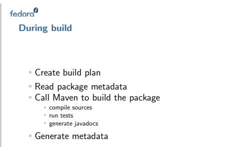
Patching POM files

```
%pom_add_dep org.apache.commons:commons-io
%pom_disable_module submod-foo
```

Launching build

```
%mvn_file : %{name}
%mvn_build
```

Before the actual build is started POM files are patched using new Macros if needed, but usually that step is not required. Next the build process is configured using simple macros and the portable part of XMvn is launched.

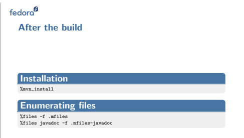


fedora

During build

- Create build plan
- Read package metadata
- Call Maven to build the package
  - compile sources
  - run tests
  - generate javadocs
- Generate metadata

During the build the configuration is read and the build plan is created. XML metadatata of all Maven packages installed in the file system is read and Maven is invoked to perform the build. After Maven completes the build metadata of currently built packages is generated.



fedora

After the build

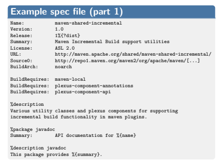
Installation

```
%mvn_install
```

Enumerating files

```
%files -f .mfiles
%files javadoc -f .mfiles-javadoc
```

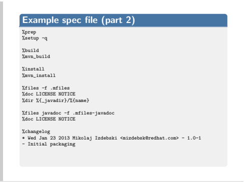
After build is finished all relevant files are installed into the build root and lists of installed files are created. These lists are then used to let RPM know which file belongs to which package.



Example spec file (part 1)

```
Name:           maven-shared-incremental
Version:        1.0
Release:        1%{?dist}
Summary:        Maven Incremental Build support utilities
License:        ASL 2.0
URL:            http://maven.apache.org/shared/maven-shared-incremental/
Source0:        http://repo1.maven.org/maven2/org/apache/maven/[...]
BuildArch:      noarch

BuildRequires:  maven-local
BuildRequires:  plexus-component-annotations
BuildRequires:  plexus-component-api

%description
Various utility classes and plexus components for supporting
incremental build functionality in maven plugins.

%package javadoc
Summary:        API documentation for %{name}

%description javadoc
This package provides %{summary}.
```

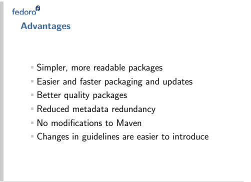
A typical RPM spec file for package built using XMvn consists of standard package metadata (name, version, etc.), there are not even requires, only build requires.



Example spec file (part 2)

```
%prep
%setup -q

%build
%mvn_build

%install
%mvn_install

%files -f .mfiles
%doc LICENSE NOTICE
%dir %{_javadir}/%{name}

%files javadoc -f .mfiles-javadoc
%doc LICENSE NOTICE

%changelog
* Wed Jan 23 2013 Mikolaj Izdebski <mizdebsk@redhat.com> - 1.0-1
- Initial packaging
```

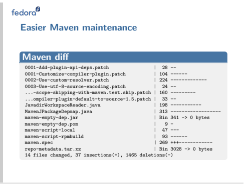
Build and install sections are usually very simple, possiblt single-line. In files sections only non-maven files (like license texts or documentation files) need to be listed, everything Macen-specific is handled automatucally.



Advantages

- Simpler, more readable packages
- Easier and faster packaging and updates
- Better quality packages
- Reduced metadata redundancy
- No modifications to Maven
- Changes in guidelines are easier to introduce

From previous examples its clear that the new way of packaging Maven artifacts has numerous advantages.

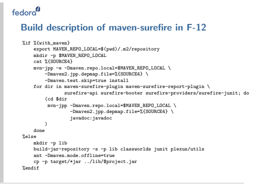


Easier Maven maintenance

Maven diff

```
0001-Add-plugin-api-deps.patch                 |  28 --
0001-Customize-compiler-plugin.patch           | 104 ------
0002-Use-custom-resolver.patch                 | 224 -------------
0003-Use-utf-8-source-encoding.patch           |  24 --
...-scope-skipping-with-maven.test.skip.patch | 160 ---------
...ompiler-plugin-default-to-source-1.5.patch |  33 --
JavadirWorkspaceReader.java                    | 198 -----------
MavenJPackageDepmap.java                       | 313 ------------------
maven-empty-dep.jar                            | Bin 341 -> 0 bytes
maven-empty-dep.pom                            |   9 -
maven-script-local                             |  47 ---
maven-script-rpmbuild                          |  93 ------
maven.spec                                     | 269 +++-----------
repo-metadata.tar.xz                           | Bin 3028 -> 0 bytes
14 files changed, 37 insertions(+), 1465 deletions(-)
```

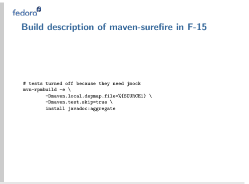
All Fedora-specific Maven customizations could be removed, which means much easier Maven maintenance in Fedora.



Build description of maven-surefire in F-12

```
%if %{with_maven}
    export MAVEN_REPO_LOCAL=$(pwd)/.m2/repository
    mkdir -p $MAVEN_REPO_LOCAL
    cat %{SOURCE4}
    mvn-jpp -e -Dmaven.repo.local=$MAVEN_REPO_LOCAL \
        -Dmaven2.jpp.depmap.file=%{SOURCE4} \
        -Dmaven.test.skip=true install
    for dir in maven-surefire-plugin maven-surefire-report-plugin \
               surefire-api surefire-booter surefire-providers/surefire-junit; do
        (cd $dir
         mvn-jpp -Dmaven.repo.local=$MAVEN_REPO_LOCAL \
                -Dmaven2.jpp.depmap.file=%{SOURCE4} \
                javadoc:javadoc
        )
    done
%else
    mkdir -p lib
    build-jar-repository -s -p lib classworlds junit plexus/utils
    ant -Dmaven.mode.offline=true
    cp -p target/*jar ../lib/$project.jar
%endif
```

That's how the build section of an example spec file used to look like in Fedora 12.

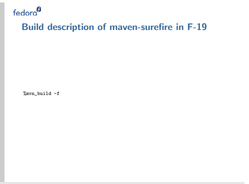


fedora

Build description of maven-surefire in F-15

```
# tests turned off because they need jmock
mvn-rpmbuild -e \
        -Dmaven.local.depmap.file=%{SOURCE1} \
        -Dmaven.test.skip=true \
        install javadoc:aggregate
```

That's how build section of the same package used to look in Fedora 15.

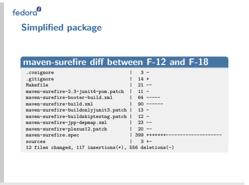


fedora

Build description of maven-surefire in F-19

```
%mvn_build -f
```

And that's Fedora 19. This example speaks for itself.



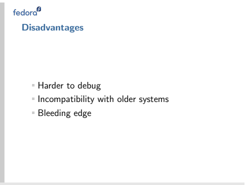
fedora

Simplified package

maven-surefire diff between F-12 and F-18

```
.cvsignore                              |   3 -
.gitignore                              |  14 +
Makefile                                |  21 --
maven-surefire-2.3-junit4-pom.patch     |  11 -
maven-surefire-booter-build.xml         |  64 -----
maven-surefire-build.xml                |  90 ------
maven-surefire-buildonlyjunit3.patch    |  13 -
maven-surefire-buildskiptestng.patch    |  12 -
maven-surefire-jpp-depmap.xml           |  23 --
maven-surefire-plexus12.patch           |  20 --
maven-surefire.spec                     | 399 +++++++-------------------------
sources                                 |   3 +-
12 files changed, 117 insertions(+), 556 deletions(-)
```

Packaging of multi-artifact packages was simplified considerably. This can be seen in an example difference between Fedora 12 and Fedora 19 package.

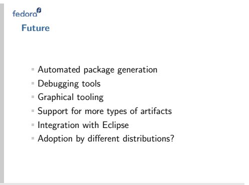


fedora

Disadvantages

- Harder to debug
- Incompatibility with older systems
- Bleeding edge

XMvn as any solution solution has its disadvantages too, but it's believed that all the advantages surpass them.



fedora

Future

- Automated package generation
- Debugging tools
- Graphical tooling
- Support for more types of artifacts
- Integration with Eclipse
- Adoption by different distributions?

Any software that is useful has to be changed, so hopefully XMvn future will bring many changes.