JNode, an operating system based on Java

JNode.org

Fabien DUMINY
Contents

- Introduction
- History
- Characteristics
- Architecture
- Plugin framework
- Driver framework
- Challenges
- Child projects
- Future
- Java benefits
Introduction

- Simple to use & install operating system for personal use: written for and in Java
- Targets:
  - Modern devices
  - Desktop
  - Small servers
- Only actively developed pure Java OS in the open source world
  - 5 active developers
  - Release 0.2.8 : 22K downloads
History

- Original idea started in 1995
- First attempt: JBS (Java Bootable System)
  - Contained C code, did not work at all
- Second attempt: JBS2
  - Still did not work well, but was better
- Then: JNode
  - No C code anymore, Classpath class libraries (and now OpenJDK and IcedTea)
- Went public in May ’03
Characteristics

• All Java, minimal assembler, no C
• All java build system (almost)
• Extensible architecture
• Single flat memory address space, no virtual memory
• JVM written in Java
• All Java code is compiled on the fly, no interpreter
• Security is always on
• LGPL license
Status (1)

• Release 0.2.8
  - Support for isolate
  - Filesystems EXT2, FAT, NTFS, ISO9660, HFS+
  - Java 6 support
  - Simple heap managers & GC
    • MMTk based heap manager & GC in development
  - IA32 & AMD64 platform support
  - Bjorne shell
  - Graphical console and console in graphical mode
Status (2)

Playing Tetris on JNode
Status (3)

JFreeChart on JNode
Status (4)

JEdit on JNode

Release Name: 0.2.8

Notes:
The JNode team is glad to announce the release of JNode 0.2.8.

JNode is a free, open source Java technology based operating system implemented in the Java language with a very small assembler nano-kernel.

The main focus of this release is general stability improvements and bug fixes and also reflects the current status of several major features in active development such as: isolates, Bjarne shell and complete support for HPSX.

Significant progress has been made with the integration of the remaining parts of OpenVIX and the JNode class library.
Architecture (1)
Architecture (2)

Java Virtual Machine

Classpath/OpenJDK/IcedTea runtime library

Runtime library support

Class manager

Security manager

Native code compilers

Scheduler & threading manager

Heap managers

Isolation manager
Plugin framework (1)

• Everything is contained in a plugin
  – code, resources
  – even JVM & the plugin framework itself

• Plugins can:
  – be loaded, unloaded & reloaded (at runtime)
  – depend on other plugins
  – provide well known extension points
  – connect to well known extension points
Plugin framework (2)

• Plugins are:
  – described by a descriptor
    • descriptor also contains license info
  – JAR files
  – inspired by Eclipse plugins (before use of OSGi)
Plugin framework (3)

```xml
<plugin id="org.jnode.driver" name="JNode Driver Framework" version="@VERSION@"
    provider-name="JNode.org" license-name="lgpl" class="org.jnode.driver.DriverPlugin">
    <requires>
        <import plugin="org.jnode.work"/>
    </requires>

    <runtime>
        <library name="jnode-core.jar">
            <export name="org.jnode.driver.*"/>
            <export name="org.jnode.driver.util.*"/>
        </library>
    </runtime>

    <extension-point id="finders" name="System device finders"/>
    <extension-point id="mappers" name="Device to Driver mappers"/>

    <extension point="org.jnode.security.permissions">
        <permission class="java.util.PropertyPermission" name="jnode.cmdline"/>
    </extension>

</plugin>
```

Copyright (c) JNode.org 2010
Driver framework (1)

Bus → contains → Device
    represents hardware

Device to driver mapper

Device finder

Driver → drives → Device
Driver framework (2)

- Device "user" calls DeviceAPI, which implements DeviceAPI.
- DeviceAPI registers API.
- Driver represents hardware and queries for API.
Driver framework (3)

Example: *Radeon Graphics card driver*

Diagram:

- **finders** find PCI devices
- **Driver plugin**
- **mappers** map PCI id to driver
- **PCI driver plugin**
- **Radeon driver plugin**
Driver framework (4)

Device manager  Device finder  Device 2 driver mapper  Driver  Device

- findDevices
- registerDevice
- findDriver
- startDevice
- registerDriver
- registerAPI
Challenges (1/2)

- Make the people come
- Let the people stay
- Implement openjdk ↔ VM bindings
  - JNode VM : bindings in pure java!
  - Issues :
    - what is that native method supposed to do?
    - Do not modify source code → easier maintenance
    - Add annotations for isolates
Challenges (2/2)

- Implement native methods in pure java
  - Needed by:
    - openjdk ↔ VM bindings
    - any application / library with native methods
  - Solution:
    - JarFile.class:
      - private native String[] getMetaInfEntryNames();
    - Replaced by NativeJarFile.class:
      - private static String[] getMetaInfEntryNames(JarFile instance)
Child Projects (1/4)

- Student projects
- Migration to maven
- JTestPlatform
Child Projects (2/4)
Student projects

- Goal: involve students in JNode development
- Source control: http://gitorious.org/jnode
- Inspired by Eric Bachard's conference at LSM 2009
- Results of 2009/2010 year:
  - Many (french) school were contacted
    - Only 3 replied positively
      - 2 went further and involved 5 students
    - 1 student from Latvia also involved
Child Projects (3/4)
Migration to Maven

- **Goal**: Use Maven instead of Ant
- **Source control**: 
  http://gitorious.org/~fduminy/jnode/maven
- **Expected benefits**:
  - Standard layout instead of home made one => easier to learn for new developers
  - **Architecture**:
    - No cycle in dependencies between plugins
    - Promote modularity
    - Discover plugin dependency issues at build time
    - Easier to use existing tools (esp. QA ones)
Child Projects (4/4)

JTestPlatform

- Goal: Test any JVM implementation
- Source control: http://gitorious.org/jtestplatform
- Not only targeted at JNode!
- Supported test frameworks:
  - Actually: JUnit, Mauve
  - Future: jtreg (partially ?), your test framework?
- Tests run in a cloud (libvirt + java binding)
- Tests run on a set of platforms (x86, x86_64 …)
- Generate reports: xml, text, html
- Future: use generated reports in QA tools
Future

- **Short term:**
  - Improved JVM performance (mm, compilers)
  - Deployment framework
  - Improved graphics

- **Long term:**
  - Simple to use desktop environment
    - Fully document oriented instead of app. oriented
  - Java powered servers
    - e.g. Cooperation with ApacheDS
Java benefits

- Dynamic linking
- Type safe language (even more since Java 5)
- Security
  - Security manager
  - No uncontrolled memory access
- Great development tools:
  - Eclipse, Ant
We need your help!

- Don't be scared by the codebase
  - Most of it is openjdk and classpath libraries
- Ask questions

- Visit http://www.jnode.org
- Contact me: fduminy@jnode.org
Demo
Questions