The Build System of Commercial RCP Application: A Case Study

Christian Kurzke

Gustavo de Paula

Daniel Moura

Agenda

- MOTODEV for JavaME Tools
- Build Requirements
- Build Levels
- Build Steps
  - Documentation
  - External Software
  - Features and Plug-ins to be used on RCP
  - Features and Plug-ins to be used on Update Site
  - Site.xml generation
  - Tool installer
- Conclusions

MOTODEV

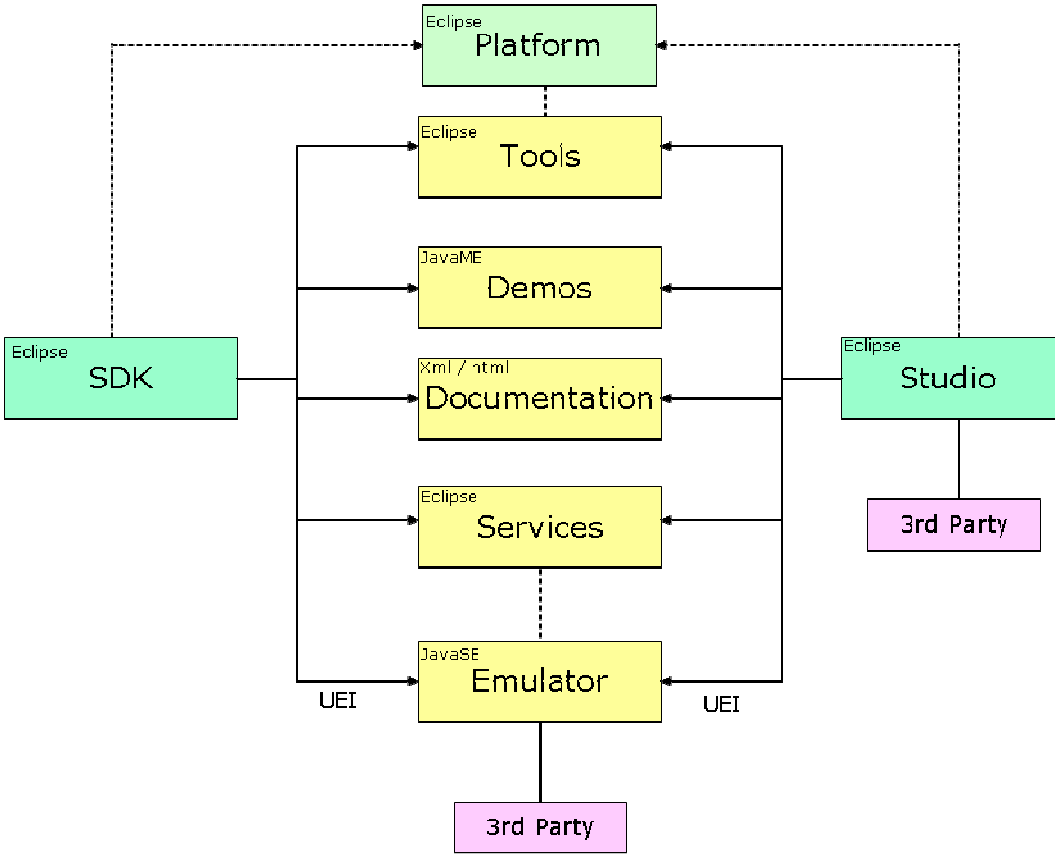
- MOTODEV is the Motorola □developer network
- Provides tools, documentation and support for developers
- Targets all different Motorola devices, from mobile devices to set-top boxes
- One of the main platforms supported by MOTODEV is mobile Java (JavaME)

MOTODEV for JavaME Tools

- Full JavaME IDE
- Based on Eclipse Platform+ JDT + EclipseME
- Includes an UEI JavaME Emulator
- Support most of MIDP 2.0 Motorola devices
- Includes complete documentation about the Devices and the APIs
- Include external tools that support the development
- Includes demo of JavaME APIs

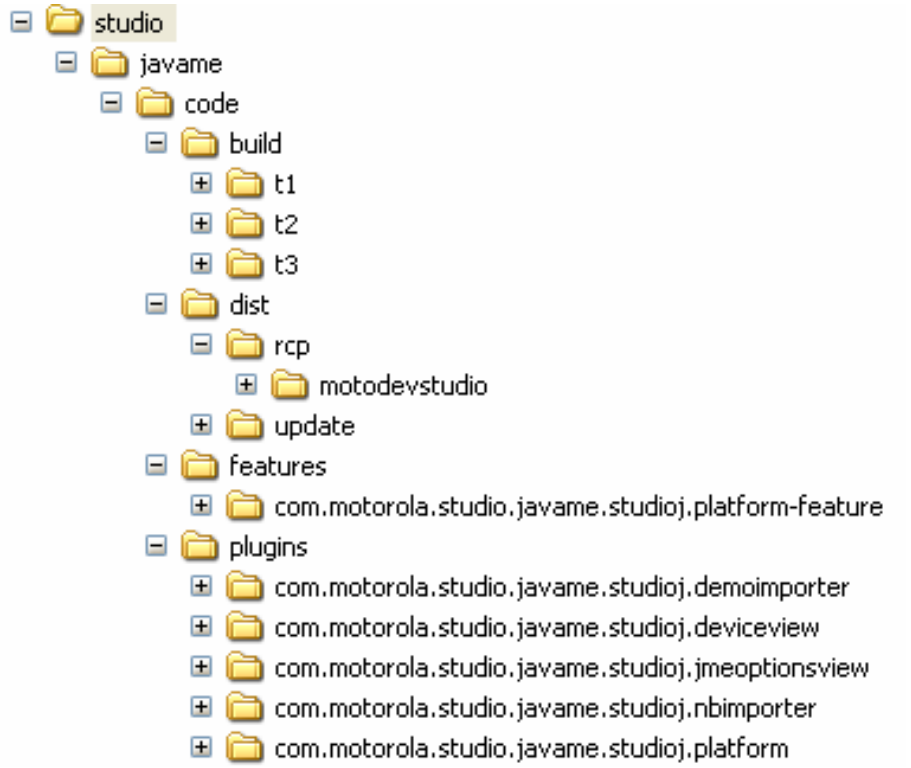
- SDK that can be integrated on any UEI compliant IDE
- Based on Eclipse RCP
- Includes an UEI JavaME Emulator
- Support most of MIDP 2.0 Motorola devices
- Includes complete documentation about the Devices and the APIs
- Include external tools that support the development
- Includes demo of JavaME APIs

MOTODEV Studio for JavaME High Level Architecture



- There are 8 sub-systems
  - Each sub-system is organized as an Eclipse feature
- 2 Eclipse products
  - SDK
  - Studio
- 3 of the sub-systems have non plug-in data
  - Emulator
  - Demos
  - Documentation
- Non plug-in data is deployed as plug-in, but there is an install handler to “install” the data
- Approximately 60 plug-ins for all features
  - 80% are shared between both products

MOTODEV for JavaME High Level Architecture

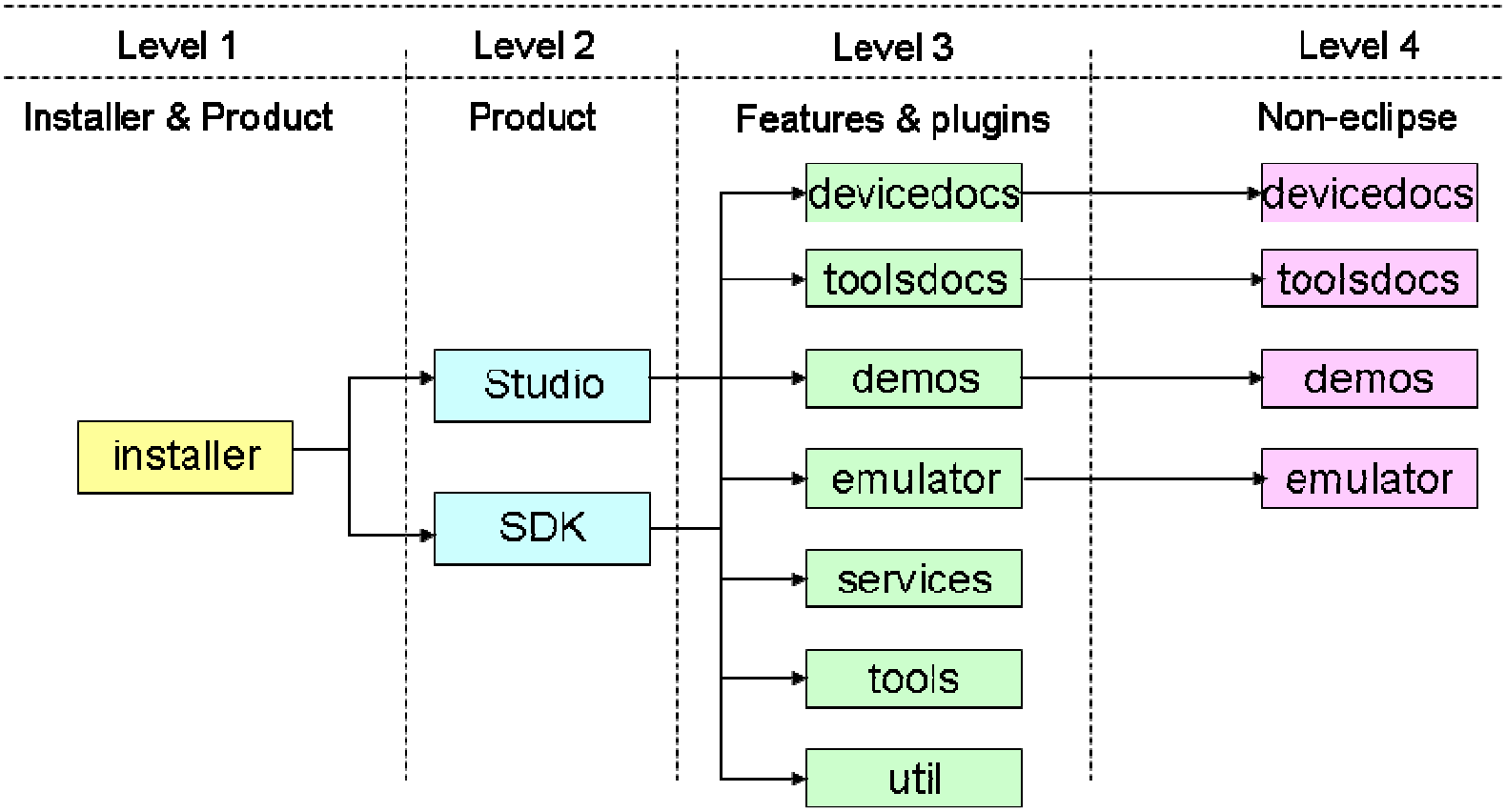


- Each sub-system has its own build.xml file
- The build.xml generates the feature/plug-ins of that sub-system
- The product build file call the dependent sub-system builds and copy the result to the base eclipse
- If the sub-system has non plug-in data, it will call the external component build

Build Requirements

- REQ1: The build must be 100% automated
- REQ2: The build must generated both products (RCP and Studio)
- REQ3: It must be possible to build only one specific part of the system and copy it to a previous compilation
- REQ4: The build must be integrated with all documentation build
- REQ4: The build must integrate the build of all non plug-in content (emulator, demos, etc.)
- REQ5: The build must generate the update site features and plug-ins
- REQ6: The build must generated the site.xml
- REQ7: The build must generated final windows installer of both products
- REQ8: The build must be fast enough that it can be used in an agile development environment

Build Levels

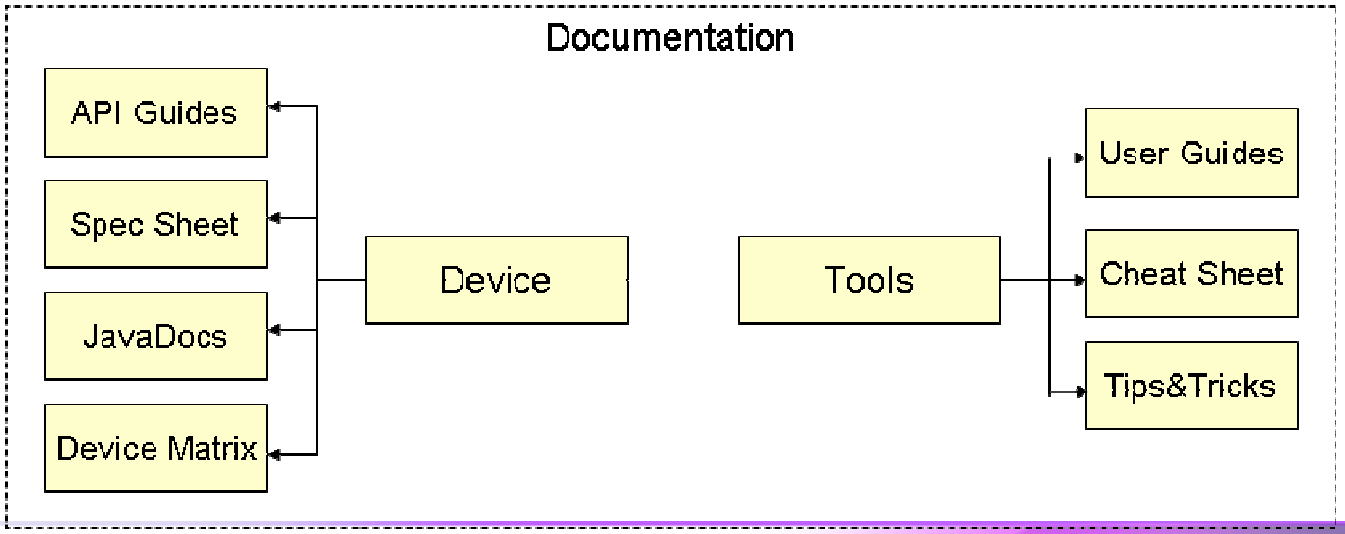


Build Steps

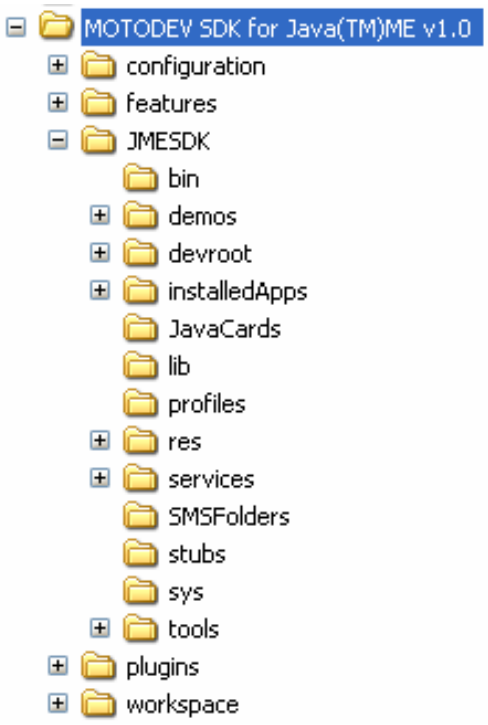
- 6 steps to generate final build
  - Documentation
  - External Software
  - Features and Plug-ins to be used on RCP
  - Features and Plug-ins to be used on Update Site
  - Site.xml generation
  - Tool installer

Build Steps → Documentation (Separated short talk - Room 203/204 on Wednesday March 19th, 2008 10:50am)

- Most complex part of the build system
- Most of our document were already written in an XML format
- The build system need to integrate the conversion of the XML into HTML
- Part of the documentation is common to both tools and part is specific



Build Steps → External Software



- Main external software is the UEI-compliant JavaME Emulator
- The emulator is a JavaSE Application that is unzipped in the root of the Eclipse Product
  - It is distributed as a set of eclipse plug-ins
- The build process is:
  - Build all emulator classes
  - Copy the content to the distribution plug-ins
  - Call Eclipse PDE Build to generate the final plug-ins
- In update manager the unzip is performed by an install handler
- During build, the only solution was to unzip the emulator plug-ins after the build process is finished
  - One alternative was generate a local update site and call the standalone update

Build Steps → Features and Plug-ins to be used on RCP

- Used basic Eclipse PDE build infra-structure
- All code is organized in features and plug-ins folders
- Base Eclipse is used to build the features and plug-ins

Build Steps → Features and Plug-ins to be used on Update Site

- After the product is generated the update site features are generated
- Eclipse PDE is not very well documented and organized to do that (there are some tips on Eclipse newsgroup)
  - The proposed solution did not worked for us
- Only solution was to zip all features and unzipped plug-ins after the Product was generated

Build Steps → Site.xml generation

- There is no direct support on PDE to do that on automated build

- Only solution was to implement a script after the update site features and plug-ins were generated

Build Steps → Tool installer

- Eclipse help suggest a way to organize the installer, but
  - It didn't mention external softwares
  - There is no suggested template that can be used
- Created a template based on "Install Jammer" (see: www.installjammer.com)
- The installer is generated for both products after the products are built

Conclusions

- Eclipse PDE provide most of the functionality that is necessary on a build system of a real Eclipse-based tool
- The are some steps that are missing, but they can be implemented separately
- There is a lot of information on how to implement the missing steps, but it is not easy to find
- The effort to set up the build infra-structure of an Eclipse-based application is quite high (but it worth doing it)

