



# Scientific (Java) Programming 2.0

Simon Gregor Ebner [simon.ebner@psi.ch]

Efficient scientific Java development need a set of well defined processes, tools and standards/technologies. This poster summarizes the Java development procedures and environment at the Paul Scherrer Institute / Swiss Light Source (SLS)

## Process

- Agile Development Process
- Modeling via UML



## Development Process

- Very short iterative cycles
- Daily/weekly "development meetings" with product owner
- Identification and prioritization of work packages with product owner



## Modelling (UML)

- *Visual Paradigm*
- Visual modeling of software
- Improves code structure, quality
- Visual documentation
- No use of code generation/roundtrip so far

## Infrastructure / Tools

- Well established and maintained development environment to support development



## Application Server

- *Glassfish v3*
- Used to host centralized (web) applications
- Standardized authentication and authorization through server
- Standardized database access through server



## Issue Tracking

- *Jira*
- Centralized storage and documentation of requests, bugs and issues



## Dependency Management

- *Apache Ivy*
- Lightweight dependency manager
- Can easily make use of Maven artifact repositories
- Good IDE support and ANT integration
- Low learning curve, easy to install/use
- Own repository (see Artifact Repository) to publish own artifact



## IDE

- *Eclipse/Netbeans*
- Use of advanced Java development environments



## Artifact Repository

- *Artifactory*
- Caching/mirroring of the global Maven repository <http://mvr.repository.com>
- Centralized storage of versioned development artifact (i.e. Jar files)



## Continuous Integration

- *Hudson*
- Automated build and test environment
- Low learning curve, easy to install/use



## Build System

- *Apache ANT*
- Low learning curve, easy to install/use
- Flexible and extensible
- Good IDE support



## Revision Control

- *CVS / Subversion*
- All code/modules/applications are versioned



## Database

- *Derby, Oracle*
- Test databases (Derby, Oracle)
- Embedded databases (Derby)
- Production databases (Oracle)

## Coding / Technology / Standards

- InCode documentation via Javadoc
- Keep code as simple and dependencies as low as possible
- Use of modern Java Technologies and Standards (no dependencies on a specific implementation of a standard)
- Modularization – Creation of independent lightweight libraries/services for a specific purpose
- An application is a mashup of libraries/services
- Clear separation of data model(s) and logic



## JUnit

- Creation of JUnit tests for all Java classes



## Jython

- *Java Scripting API*
- Use of Jython via Java Scripting API for scripting
- The use of the Java Scripting API facilitates the change of the scripting language to Javascript or some other language

## Logging

- *java.util.Logging*
- Comes with standard JDK/JVM
- Native logging support in Glassfish Application Server

## JAXB

- Use of JAXB to read and write XML documents
- Automatic class generation via XSD file
- Validation of XML before unmarshalling



## XML / XSD / XSL

- Use of XSD validation to validate data against an agreed schema
- Use of XSL transformation to transform XML data models

...

## JPA - Java Persistence API

- *EclipseLink*
- Facilitate database access and querying
- Additional layer of abstraction compared to JDBC



## OSGi

- *Apache Felix*
- Used to build modularized applications

## EJB 3.0

- Used to build server side applications

## Web Services

- *Metro*
- SOAP
- Rest (on some projects in future)

## JSF 2.0 - Java Server Faces

- Server side visualization

## JSP - Java Server Pages

- Server side visualization



## Netbeans Platform

- Used to build modularized GUI applications
- Use of Netbeans Platform instead of Eclipse RCP because of:
  - Platform independence because of Swing
  - Visual GUI Builder
  - Skinning support

## Findings – Scientific (Java) Programming need to ...

- Have more specialized (e.g. for plotting, control system access) and well defined libraries/services (no Frameworks) to enable mashups like in Web 2.0
- Make more use of industry standards and best practices
- Focus more on standardized interfaces and data access protocols than on the standardization of data formats