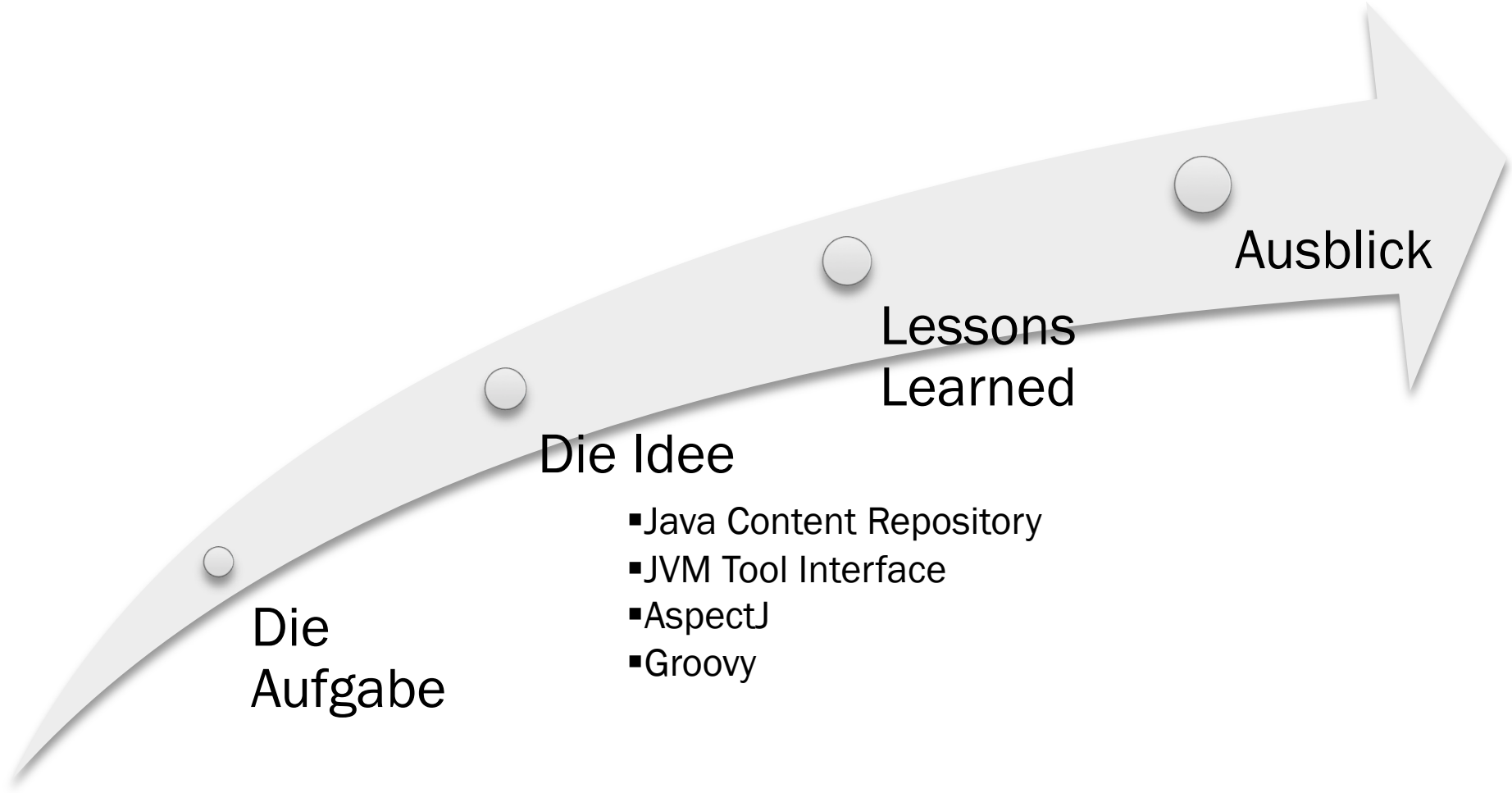
JUG – Saxony 16.07.2009

Christian Wurbs

INTEGRATION EINER IDE FÜR GROOVY

- IN EINER PRODUKTIONS-UMGEBUNG -

AGENDA


Ausblick
Lessons Learned
Die Idee
Java Content Repository
JVM Tool Interface
AspectJ
Groovy
Die Aufgabe

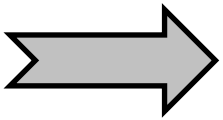
EINFÜHRUNG

- CHRISTIAN WURBS
  - seit 9 Jahren Entwicklung von Software auf Basis von Java
  - Senior Developer bei der itemic AG
    - www.itemic.de
  - E-Mail:
    - Christian.Wurbs@itemic.com

DIE AUFGABE

- Software für die Halbleiterindustrie muss eine schnelle Integration neuer Mess- und Analysewerkzeuge bzw. –methoden ermöglichen.
- Einer der Schlüsselfaktoren für das Bestehen auf einem globalen Markt

- In der Vergangenheit haben die Kunden hierbei erfolgreich mit Spreadsheet-Lösungen gearbeitet
- für heutige Datenmengen und Zeitanforderungen nicht mehr anwendbar

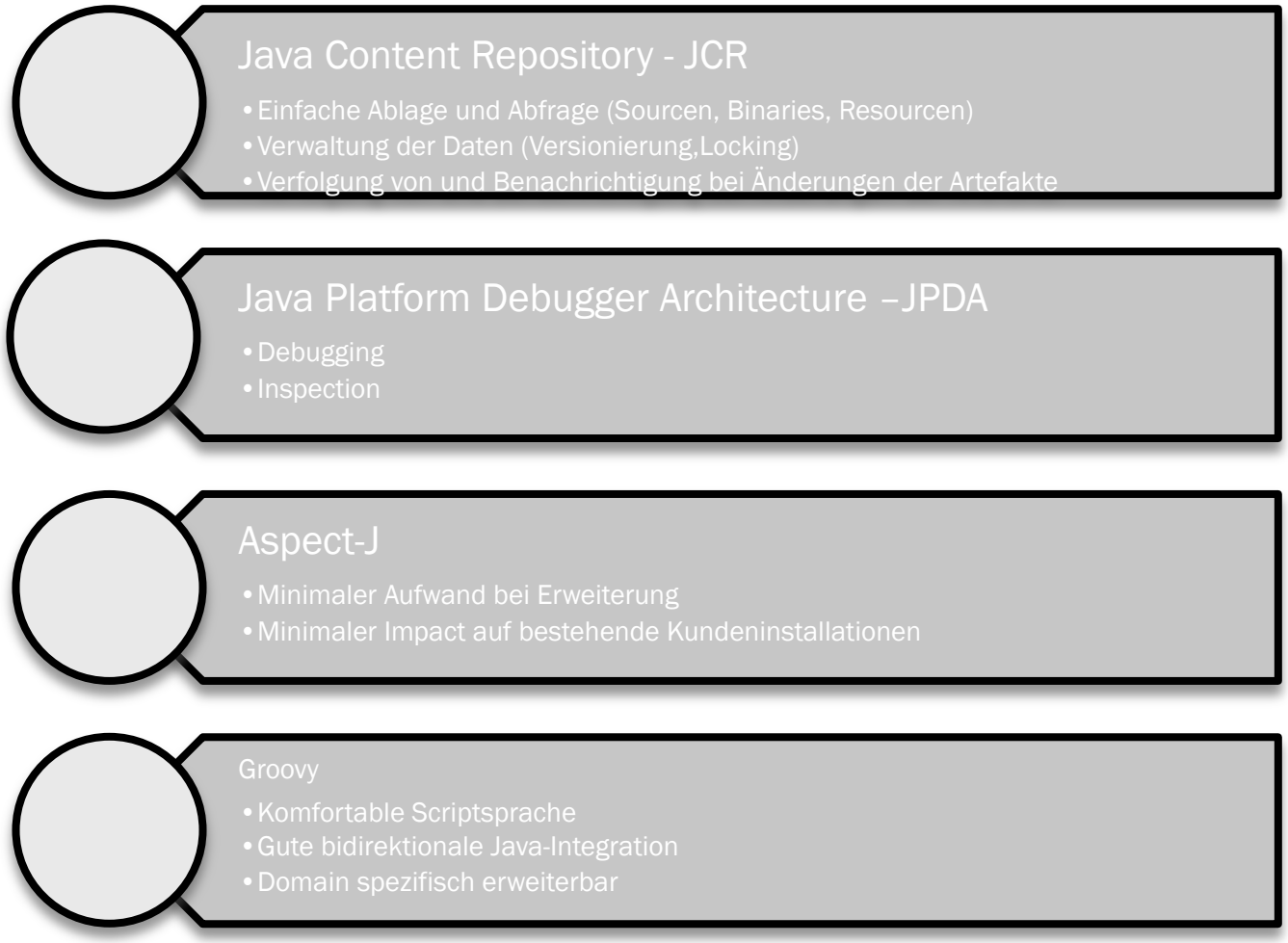


Diese Herausforderungen müssen kurzfristig für Evaluierung und Forschung – auch prototypisch oder experimentell – mit der bereits vorhandenen Software und einer bereits verfügbaren Datenbasis gelöst werden können, ohne dabei die Flexibilität der Spreadsheet Anwendungen aufgeben zu müssen.

DIE IDEE – DIE ANFORDERUNGEN

- Erweiterung der itemic Plattform mit einer IDE
  - \+ Komfortable Scriptsprache
  - \+ Gute bidirektionale Java-Integration
  - \+ Domain spezifisch erweiterbar
  - \+ Einfache Ablage und Abfrage (Sourcen, Binaries, Resourcen)
  - \+ Verwaltung der Daten (Versionierung,Locking)
  - \+ Verfolgung *von* und Benachrichtigung *bei* Änderungen der Artefakte
  - \+ Debugging
  - \+ Inspektion
  - \+ Minimaler Aufwand bei Erweiterung
  - \+ Minimaler Impact auf bestehende Kundeninstallationen

DIE IDEE – DIE TECHNOLOGIEN

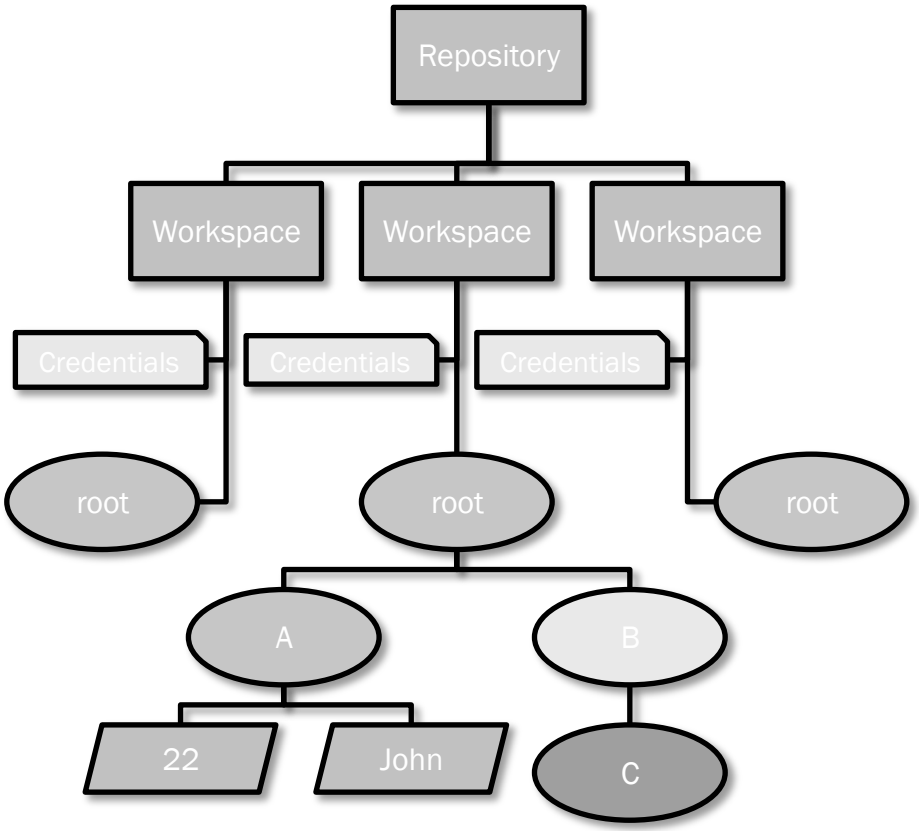


WAS IST JCR ? WOFÜR KANN MAN ES VERWENDEN?

- spezifiziert im JCP
  - JSR-170 (Version 1 – Final Relese 17. Juni 2005, Überarbeitet 22. März 2006)
  - JSR-283 (Version 2 – Proposed Final Draft vom 31 März 2009)
    - Erweiterungen wie: Federation, Remoting, mehr Standard Knotentypen, sowie bessere Zugriffskontrolle
- JCR ist keine Content Repository Implementierung sondern ein API
- Einheitlicher Weg für den Zugriff auf “Content” in einem Repository
  - Was JDBC für RDBMS ist, ist JCR für Content Repositories
- David Nüscheler von Day Software ist Specification Lead bei beiden
  - Kommerzielle JCR Implementierung: Content Repository Extreme (CRX)
  - Im ASF Open Source Projekt “Jackrabbit” engagiert (1.0 April 2006)
    - Aktuell: JSR 170 - 1.5.6 / JSR 283 - 2.0 alpha3

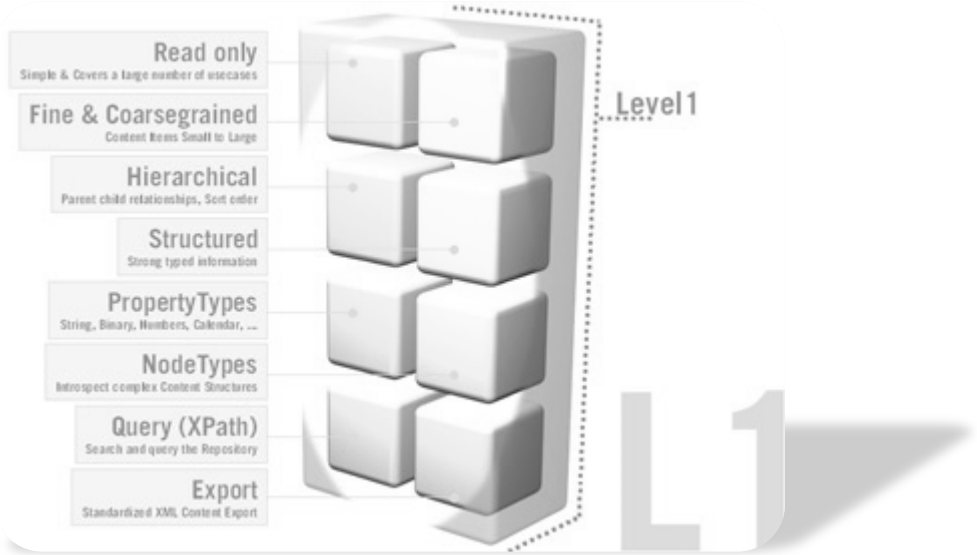
JCR – BASICS

- *Repository*:
  - besteht aus ***Workspaces***
- *Session*:
  - Sicht für den Zugriff auf einen Workspace im Repository
- Workspace:
  - Hierarchie/Baum von ***Items***
- Item:
  - *Node*:
    - kann eine Knotentyp-Definition haben
  - *Property*:
    - immer nur ein Blatt
    - Property-Values beinhalten die Daten
    - Datentypen:
      - STRING BINARY LONG DOUBLE BOOLEAN DATE PATH NAME REFERENCE UNDEFINED



JSR 170 – LEVEL 1

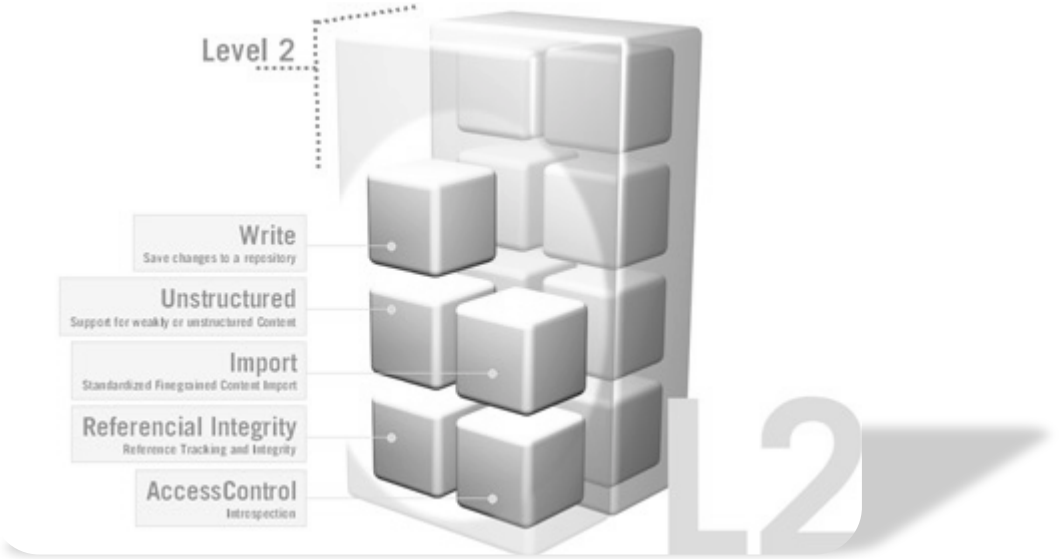
- spezifiziert eine “Read-Only-API“
- Einfache Anwendbarkeit
- Abdeckung vieler (wenig komplexer) Anwendungsfälle:
  - + Suche nach Inhalten
  - + Lesen von Inhalten
- Beispiele:
  - + Portlets
  - + CMS-Templates
  - + Reports
  - + Exports



Quelle: ASF - http://jackrabbit.apache.org/jcr-api.html

JSR 170 – LEVEL 2

- spezifiziert die “Writing-API”
- Level 1 +:
  - + Hinzufügen, Entfernen, Modifizieren von Nodes und Properties
  - + Import von Repository-Inhalten
  - + Zuweisen von Knotentypen (Namespace-Aware)
  - + Hinzufügen, Entfernen, Modifizieren von Namespaces
  - + Hinzufügen, Entfernen, Modifizieren von Referenzen



Quelle: ASF - http://jackrabbit.apache.org/jcr-api.html

JSR 170 – OPTIONALE FEATURES

- Versionierung
  - Checkin, Checkout
- (JTA) Transaktionen
  - Container und User-managed
- Content-Observation
  - Registrierung von Listenern für konkrete Änderungen im Repository:
    - NODE_ADDED, NODE_REMOVED, PROPERTY_ADDED, PROPERTY_CHANGED, PROPERTY_REMOVED
- SQL als weitere Abfrage-Sprache neben Xpath
- Explizites Locking von Nodes um konkurrierende Änderungen zu verhindern

Quelle: ASF - http://jackrabbit.apache.org/jcr-api.html

JACKRABBIT 1.5.6

- 100% JSR-170 konform
  - Level 1, Level 2, alle optionalen Features
  - Erweiterungen:
    - ObjectContentMapping
    - Clustering
- weitere Projekte:
  - Apache Sling (Web-Framework)

➔ JCR Example

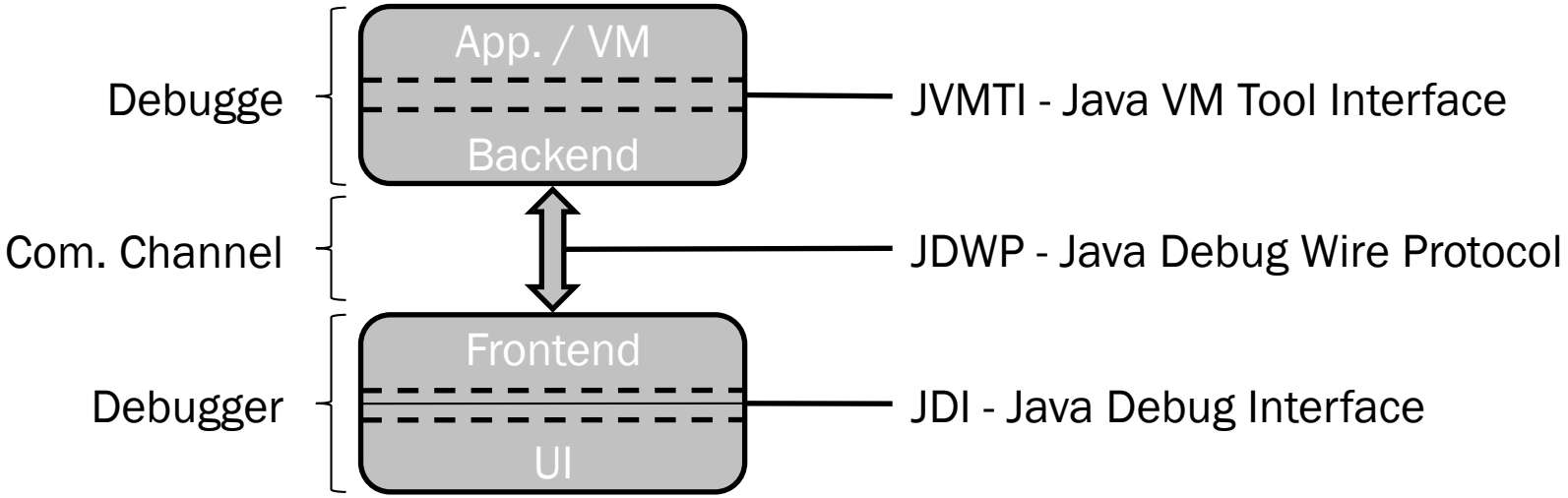
JCR –„LESSONS LEARNED“

- Solange es geht mit dem Knotentyp nt:unstructed arbeiten ➔ Migration von Knotentypen
- OCM nur bei klar definiertem Domänen-Content
- OCM erschwert Migrationen
- David‘s Model folgen – 7 einfache Regeln

http://wiki.apache.org/jackrabbit/DavidsModel

JPDA – BASICS

- Multi-Tiered Debug Architektur
- Unabhängig bzgl.: Platform, VM Implementierungen und JDK Version



JVMTI

- mit J2SE 5.0 neu eingeführt
- ersetzt JVMDI und JVMPI

JDWP

- Protokollspezifikation zur Kommunikation zwischen Debuggee und Debugger

JDI

- High-Level Java API
- einfache Nutzung zur Entwicklung von Debuggern

JPDA – AKTIVITÄTEN

Requests

- werden vom Debugger:
  - gestellt
  - aktiviert bzw. deaktiviert
  - konfiguriert
- beinhaltet:
  - Abfrage von Information
  - Setzten von Änderungen in der “remote Applikation / VM”
  - Setzten von Debugging Status wie VM oder Thread-Suspension

Events

- werden auf “Debuggee-Seite” erzeugt
- zeigen Änderungen in der “remote Applikation / VM” an

➔ JDI Example

ASPECT-J – BASICS

- Aspektorientierte Java-Spracherweiterung
- Erlaubt eine Modularisierung sog. „Crosscutting Concerns“
  - Beispiele: Error Checking und Handling, Synchronisation, Kontext-Sensitives Verhalten, Performance Optimirungen, Monitoring und Logging, Debugging

Interzeptoren

- unterbrechen den vorher definierten Programmfluss
- „klinken“ sich in dessen Ausführungspfad ein
- An definierten Stellen kommen sogenannte Einschubmethoden zur Ausführung.
- Unterscheidung nach:
  - before
  - after
  - around

Joinpoints

- Die Stellen, an denen Sie sich mit Interzeptoren in ein bestehendes Programm einklinken können
- können Methodenausführungen sein
- Zugriff auf Attribute eines Objekts
- Aufruf der Operationen einer Klasse
- die Erstellung von Objekten
- das Auslösen einer Exception

Pointcut

- definiert, an welchen konkreten Joinpoints Interzeptoren aufgerufen werden sollen
- Während Joinpoints nur mögliche Einstiegspunkte für Interzeptoren definieren, legen Pointcuts fest, welche Joinpoints in einem konkreten Fall genutzt werden sollen
- legt noch nicht fest, welche konkreten Interzeptoren verwendet werden

Advice

- beschreibt, was wann an den Joinpoints eines Pointcuts passiert
- definiert welche Interzeptoren an den durch einen Pointcut selektierten Joinpoints in den Programmfluss eingefügt werden und welcher Code an diesen Stellen ausgeführt wird

Aspekt

- fasst Pointcuts und Advices zusammen
- zum Beispiel:
  - Sicherheitsaspekt besteht aus mehreren konkreten Advices, die sich auf verschiedene Pointcuts beziehen
  - Modifizierung existierender Klassen durch hinzufügen von Membern und Methoden

ASPECT-J – EVENTDISPATCHASPECT

```
3  import org.apache.log4j.Logger;
9
10 public aspect AlgoStepEventDispatchAspect
11 {
12     private static final String _IDENT = "AlgoStepEventDispatchAspect";
13
14     pointcut callsToAlgoStep_RunStep(AlgoStep step, AlgoData data):
15         execution(boolean AlgoStep+.runStep(AlgoData))
16         && !within(AlgoStep)
17         && target(step)
18         && args(data);
19
20     before(AlgoStep step, AlgoData data) : callsToAlgoStep_RunStep(step,data)
21     {
22         final String className = step.getClass().getName();
23         Logger.getLogger(_IDENT).debug("before calling " + className + ".runStep()");
24         triggerEvent(className  + "#OnBefore",data);
25     }
26
27     after(AlgoStep step, AlgoData data) returning: callsToAlgoStep_RunStep(step,data)
28     {
29         final String className = step.getClass().getName();
30         Logger.getLogger(_IDENT).debug("after returning succesfully from " + className + ".runStep()");
31         triggerEvent(className  + "#OnAfter",data);
32     }
33
34     after(AlgoStep step, AlgoData data) throwing: callsToAlgoStep_RunStep(step,data)
35     {
36         final String className = step.getClass().getName();
37         Logger.getLogger(_IDENT).debug("after returning with exception from " + className + ".runStep()");
38         triggerEvent(className  +  "#OnException",data);
39     }
40
41     private static final void triggerEvent(String eventName,AlgoData data)
42     {
43         EventDispatcher.getInstance().triggerEventSilently(eventName, false, new AlgoDataContext(data), _IDENT);
44     }
45 }
```

ASPECT-J – SCRIPTING EVENT LIBRARY

```
<library name="Overlay">
    <element name="Selection" >
        <element name="DirectSelection" >
            <events>
               <event name="prepareContextMenu" user="$current_user$" event-id="DirectSelectionOV#prepareContextMenu" >
                    <parameter name="popUpMenu" type="javax.swing.JPopupMenu" />
                </event>
            </events>
        </element>
    </element>
    <element name="ShowAnalysis" >
        <events>
            <event name="prepareViews" user="$current_user$" event-id="ShowAnalysisComponent#prepareViews" >
                <parameter name="viewList" type="sapi.overlay.ui.AnalysisResultViewList" />
            </event>
        </events>
    </element>
    <element name="OVAnalysis" >
        <events>
            <event name="afterAnalysis" user="$static_user$" event-id="OvLAlgoMgr#OnAfter" >
                <parameter name="currentAnalysis" type="analysis.sapi.Analysis" />
            </event>
            <event name="afterSaving" user="$static_user$" event-id="datahandling.SaveAnalyseAlgoStep#OnAfter" >
                <parameter name="currentAnalysis" type="analysis.sapi.Analysis" />
            </event>
            <event name="beforeSaving" user="$static_user$" event-id="datahandling.SaveAnalyseAlgoStep#OnBefore" >
                <parameter name="currentAnalysis" type="analysis.sapi.Analysis" />
            </event>
            <event name="afterLoading" user="$static_user$" event-id="datahandling.LoadAnalyseAlgoStep#OnAfter" >
                <parameter name="currentAnalysis" type="analysis.sapi.Analysis" />
            </event>
            <event name="replaceDefaultAnalysis" user="$static_user$" event-id="optimization.OvLOptimMgr#OnReplace" >
                <parameter name="currentAnalysis" type="analysis.sapi.Analysis" />
            </event>
        </events>
    </element>
    <element name="Measurement" >
        <events>
            <event name="afterLoading" user="$static_user$" event-id="datahandling.LoadMeasurementAlgoStep#OnAfter" />
        </events>
    </element>
</library>
```

ASPECT-J – „LESSONS LEARNED“

- Akzeptanz im Entwicklungsprozess
  - + Refactorings, Call-Hierarchy
- CompileTimeWeaving:
  - + bei großen und mittel-großen Multi-Projekt-Builds Zeit- und Ressourcenaufwendig
- RuntimeWeaving:
  - + Akzeptanz bei QA
  - + erhöht den Testaufwand
  - + Frage nach Anwendbarkeit (Verwaltungsaufwand) in Verbindung mit Obfuscation

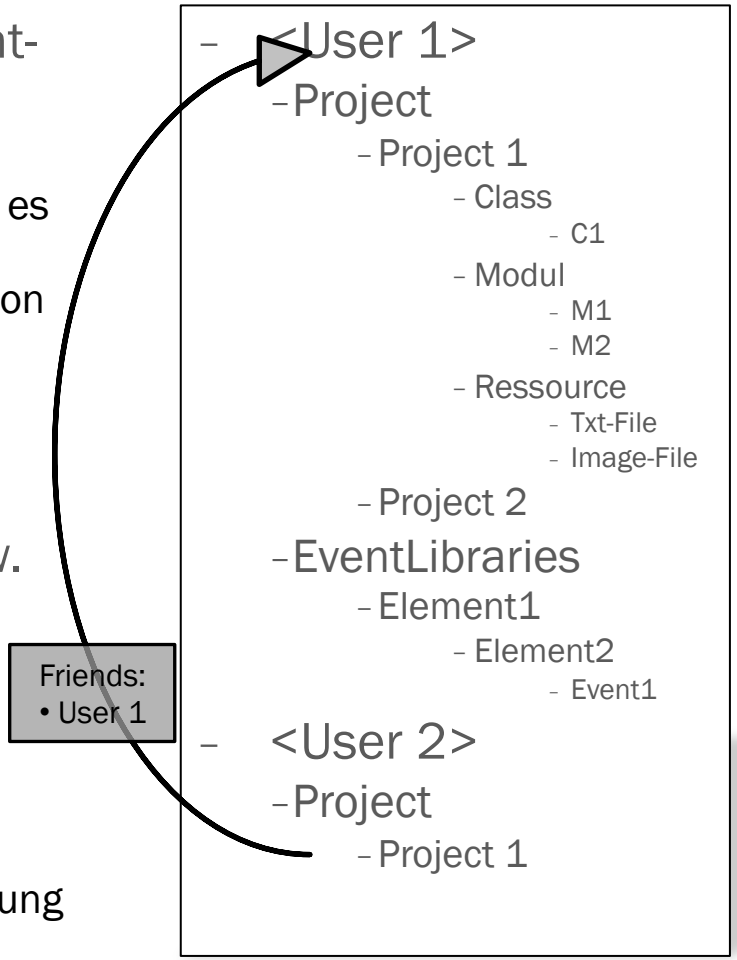
GROOVY – BASICS

- is an agile and **dynamic language** for the **Java Virtual Machine**
- builds upon the strengths of Java but has **additional power features** inspired by languages like Python, Ruby and Smalltalk
- makes **modern programming features** available to Java developers with **almost-zero learning curve**
- supports **Domain-Specific Languages** and other compact syntax so your code becomes **easy to read and maintain**
- makes writing shell and build scripts easy with its **powerful processing primitives**, OO abilities and an Ant DSL
- increases developer productivity by **reducing scaffolding code** when developing web, GUI, database or console applications
- **simplifies testing** by supporting unit testing and mocking out-of-the-box
- seamlessly **integrates with all existing Java objects and libraries**
- compiles straight to Java bytecode so you can use it anywhere you can use Java

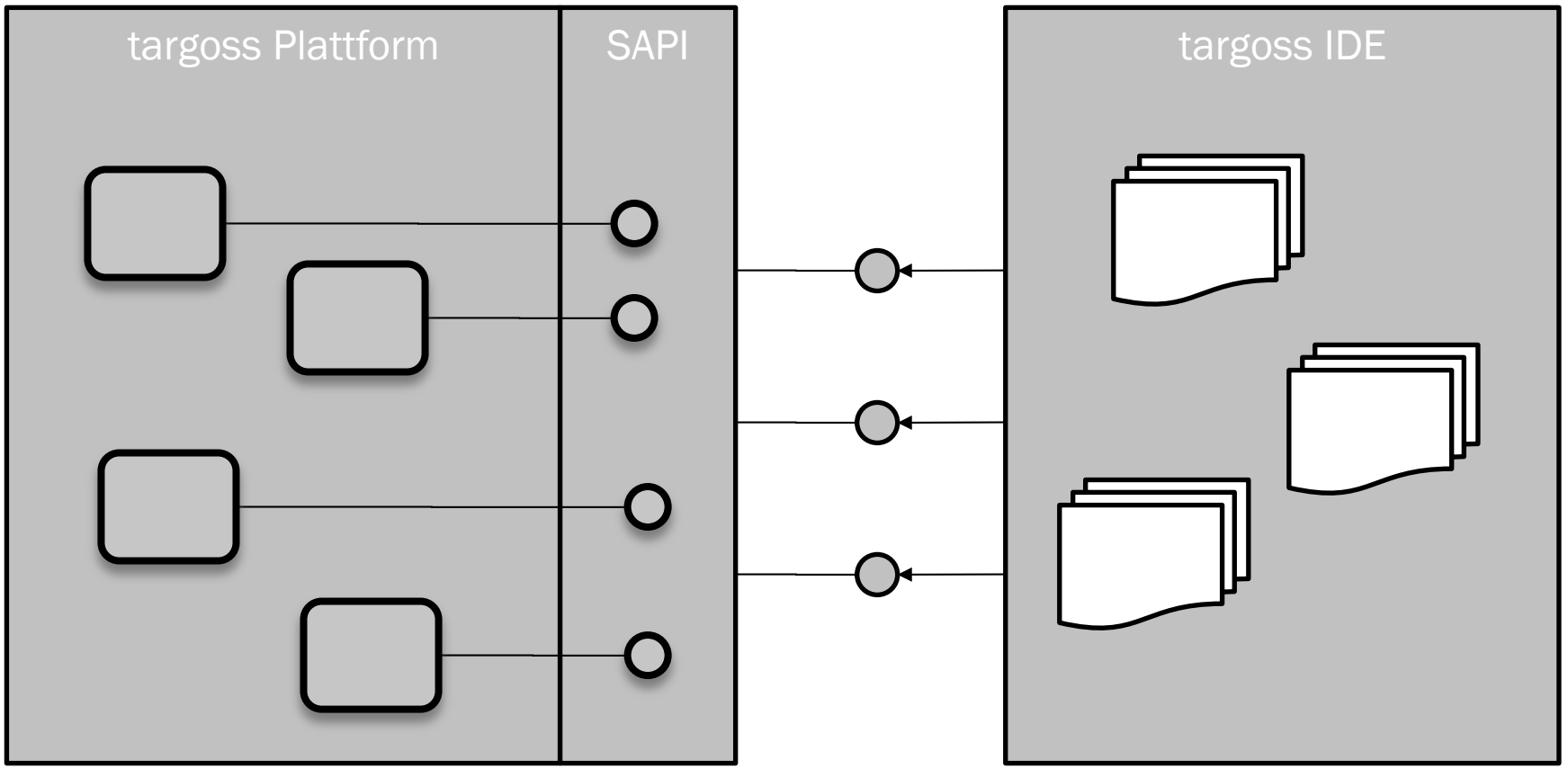
Quelle: http://groovy.codehaus.org

GROOVY – INTEGRATION

- EventDispatchAspect aktiviert die „Event-Handler“ (i.V.m. ScriptingEventLibrary)
  - + Funktionalität wird direkt implementiert oder es wird an Module delegiert
  - + „Friend“-Beziehungen erlauben das Nutzen von Projekten anderer Nutzer
  - + es dürfen nur die eigenen Projekte geändert werden
- „Dependencies“ werden auf Projekt bzw. Event-Ebene hinzugefügt
  - + Projektweite „imports“
  - + „imports“ am Ende (Debugging)
- Erweiterung GroovyClassLoader
  - + recompile von Klassen bei Abhängiger Änderung



GROOVY – INTEGRATION



AUSBLICK

- Erweiterung IDE:
  - + CodeCompletion
  - + Wizards
- SAPI – Verwaltung
- Signierung und Validierung von Scripten
- Performance Optimierung

Q&A

Vielen Dank für die Aufmerksamkeit