Verteiltes Debugging

Gemeinsames Debuggen in Saros

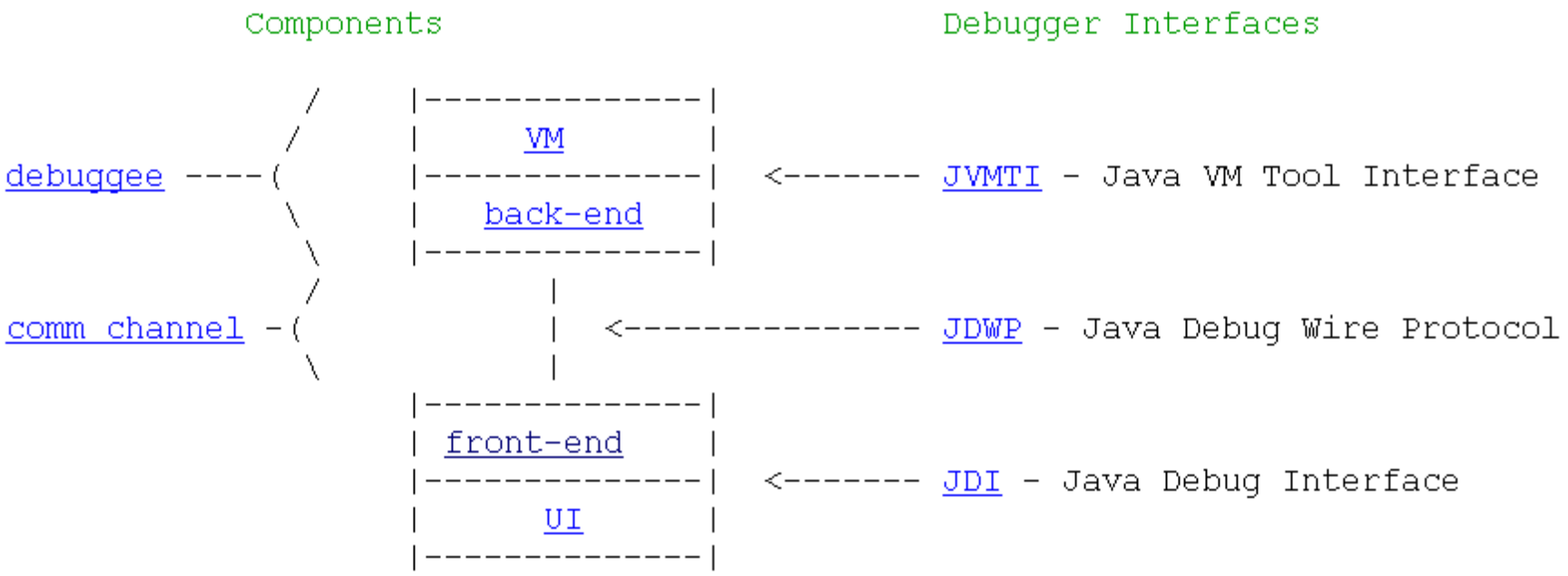
Motivation

- Saros unterstützt bislang nur das gemeinsame editieren von Quelltext
  -> Support auf Compile-Time Ebene
- Softwaredesign-Fehler (Anw-Logik) erst zur Laufzeit ersichtlich -> Logging, Debugging
- Debugging nur für Single-User Modus konzipiert: Debugger erlaubt nur ein ihn steuerndes UI.
- Gewünscht: Multi-UI, dass jedem Benutzer einen selektiven Einblick auf den Debugger erlaubt

Ziel:Debugger-Multi-UI

- Individuelles Abfragen von Debugging-Strukturen soll Einblick in Programmzustand ermöglichen
- Breakpoints und Watchpoints erstellt und ausgetauscht werden können
- Beobachtung und Steuerung von Threads durch verschiedene Benutzer (abwechselnd oder simultan) erlaubt RunTime-Brainstorming
- Parallele Abläufe durch Threadzuweisung an verschiedene Benutzer analysierbar auf Sonderfälle, Engpässe, Deadlocks

JPDA –
Java Platform Debugger Architecture

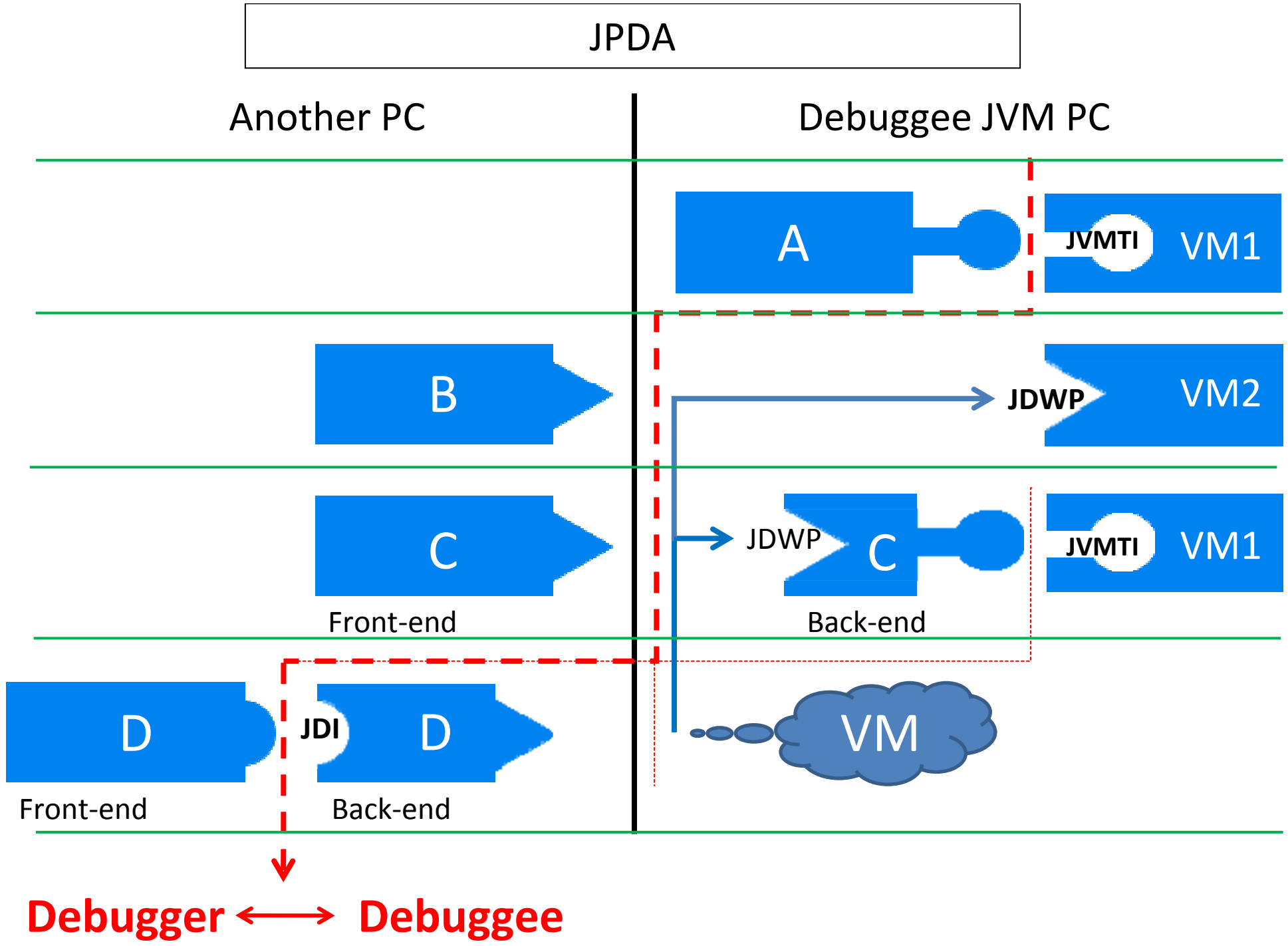


http://java.sun.com/j2se/1.5.0/docs/guide/jpda/architecture.html

JPDA

- Drei-Schichten-Architektur von Interfaces, zur Definition von Debuggee-Capabilities auf verschiedenen Abstraktionsniveaus
- Dabei kann man die Schichten JDI,JDWP und JVMTI unterscheiden, welche ähnlich wie C, Assembler und Maschinencode zueinander stehen
- Debugger steuert Debuggee über dessen implementiertes Interface (Capabilities)

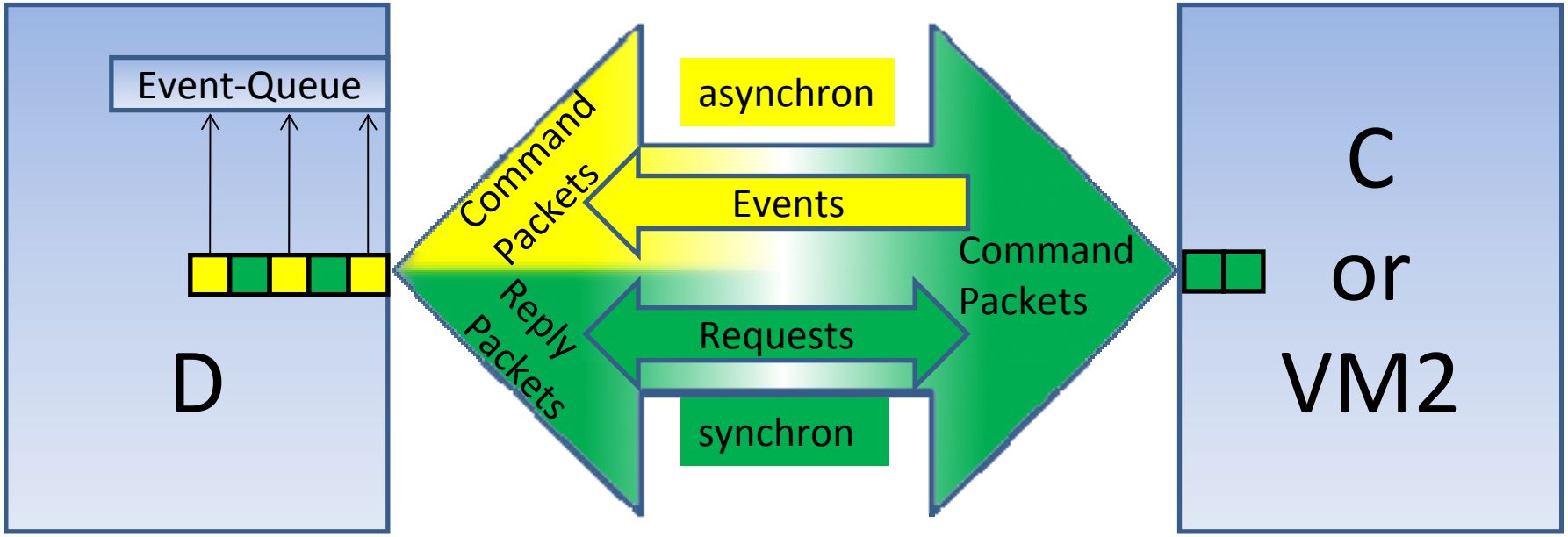

Zeichnung.pdf


JPDA
Another PC
Debuggee JVM PC
A
JVMTI
VM1
B
JDWP
VM2
C
JDWP
C
JVMTI
VM1
Front-end
Back-end
D
JDI
D
VM
Front-end
Back-end
Debugger
Debuggee

JPDA

- JVMTI – Bibliotheksschnittstelle (native) implementiert von JVM.
- JDWP – Datenverbindungsprotokoll (native).
  Es gibt die Rollen Debuggee und Debugger.
  In der Rolle Debuggee ist entweder ein vermittelnder back-end (JVMTI) oder direkt die Debuggee VM (selten). Im ersteren Fall wird die Debugger Rolle vom sog. front-end eingenommen, sonst identisch mit dem Rollennamen.
- JDI – pure JavaInterface; implementierbar vom Besitzer der Debugger Rolle in JDWP.
  Dient dann als back-end für rein in Java geschriebene Debugger; Konsolen oder GUI basiert.
- Benutzen von JVMTI und JDWP -> Debugger sprachneutral

JPDA: JDWP



Debugger: demux input by packet type

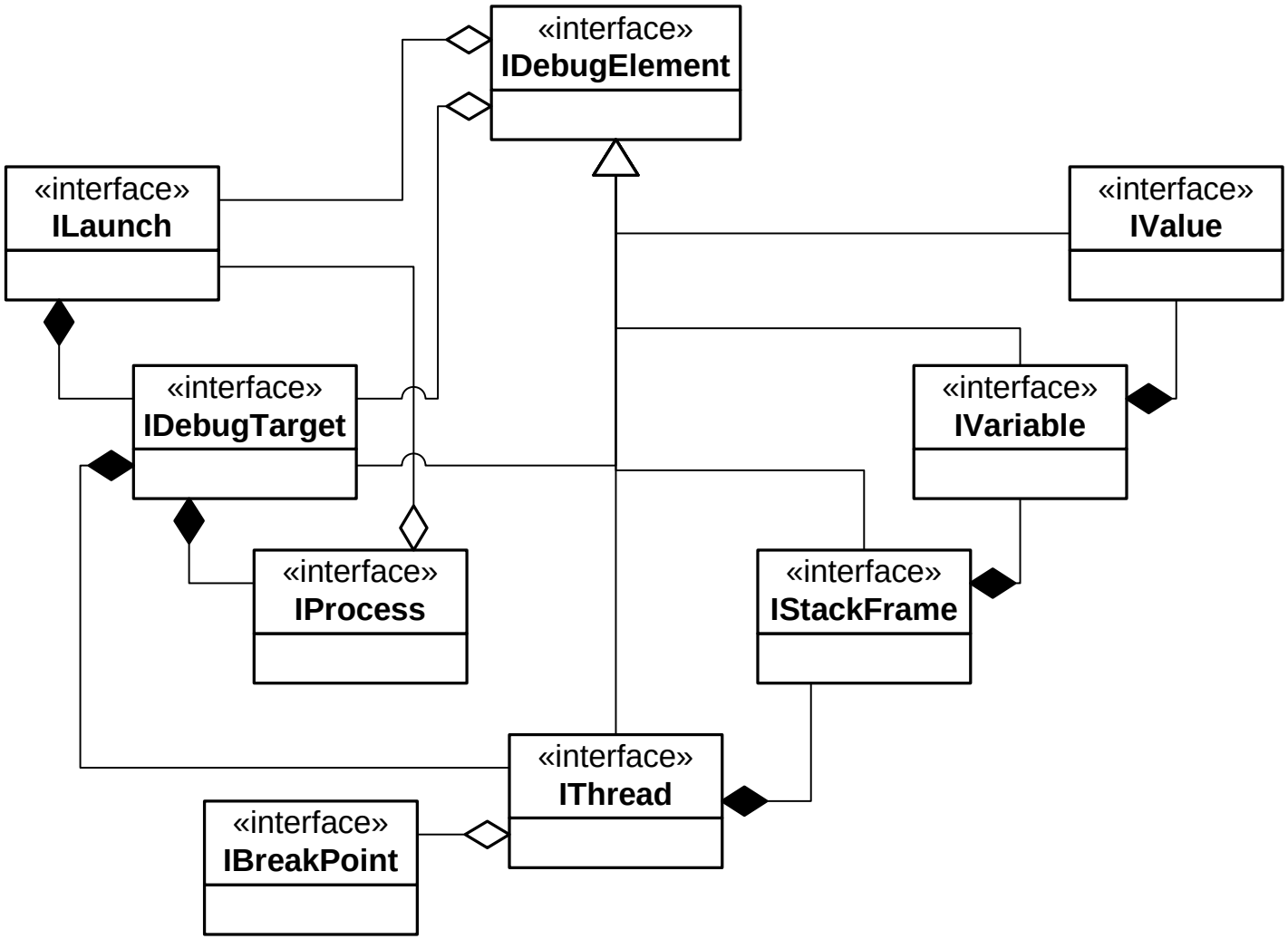
JDWP Socket Connection= Socket-Pair

Debuggee: concurrently writes to socket

Eclipse Debug Model

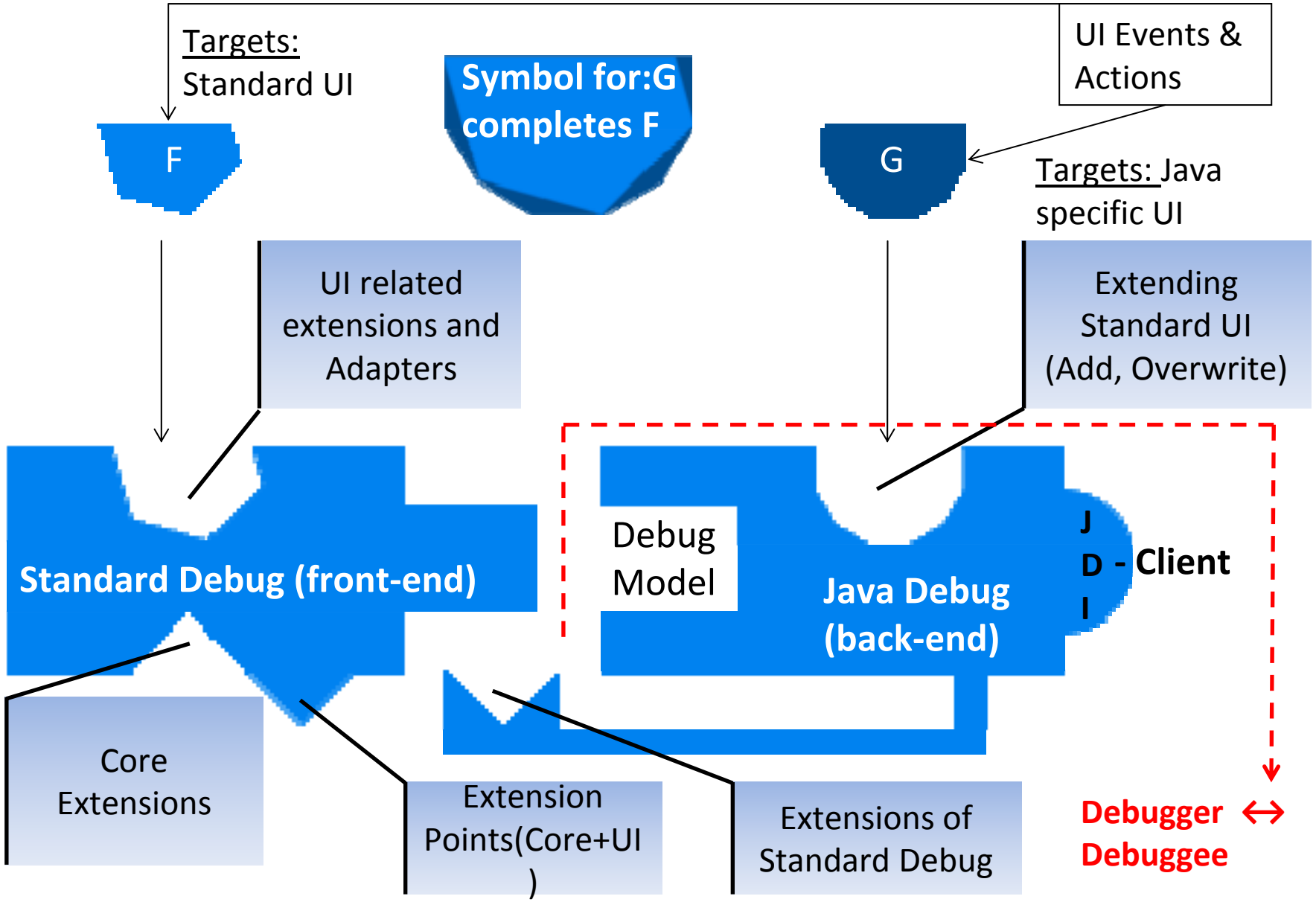
- Abstraktes Datenmodell eines Debuggees einer imperativen Sprache
- Breakpoints als Aggregat von Threads, wenn diese Thread-Suspend ausgelöst haben
- Eclipse-Debug-Perspektive kommuniziert mit diesem abstrakten Datenmodell
  ->Generische GUI
- Konkreter Debugger implementiert Interfaces des Datenmodells und erhält dadurch GUI

Eclipse Debug Model



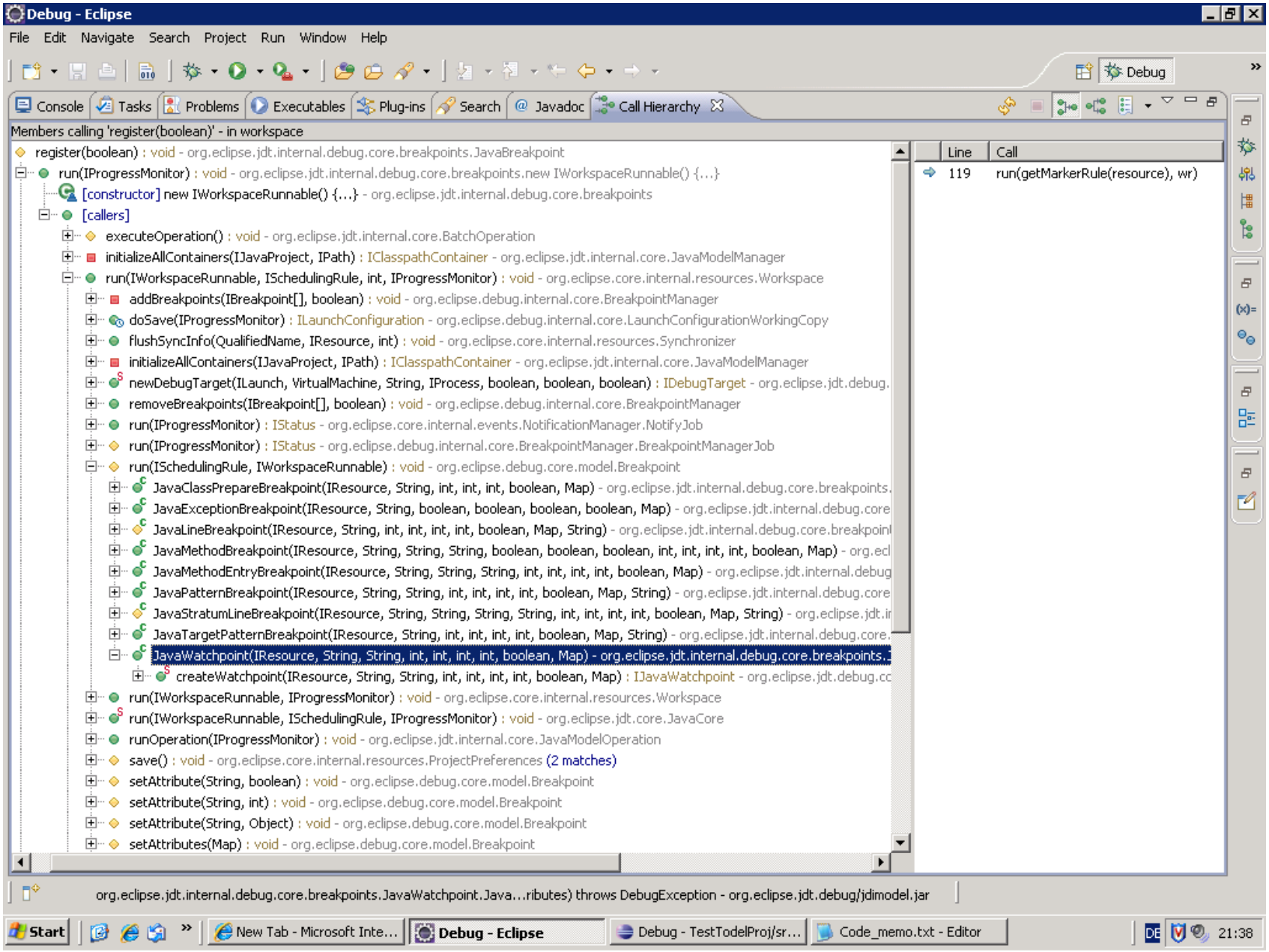
http://www.eclipse.org/eclipse/debug/documents/ec_2008/Debug_Tutorial_Basic_2008.ppt

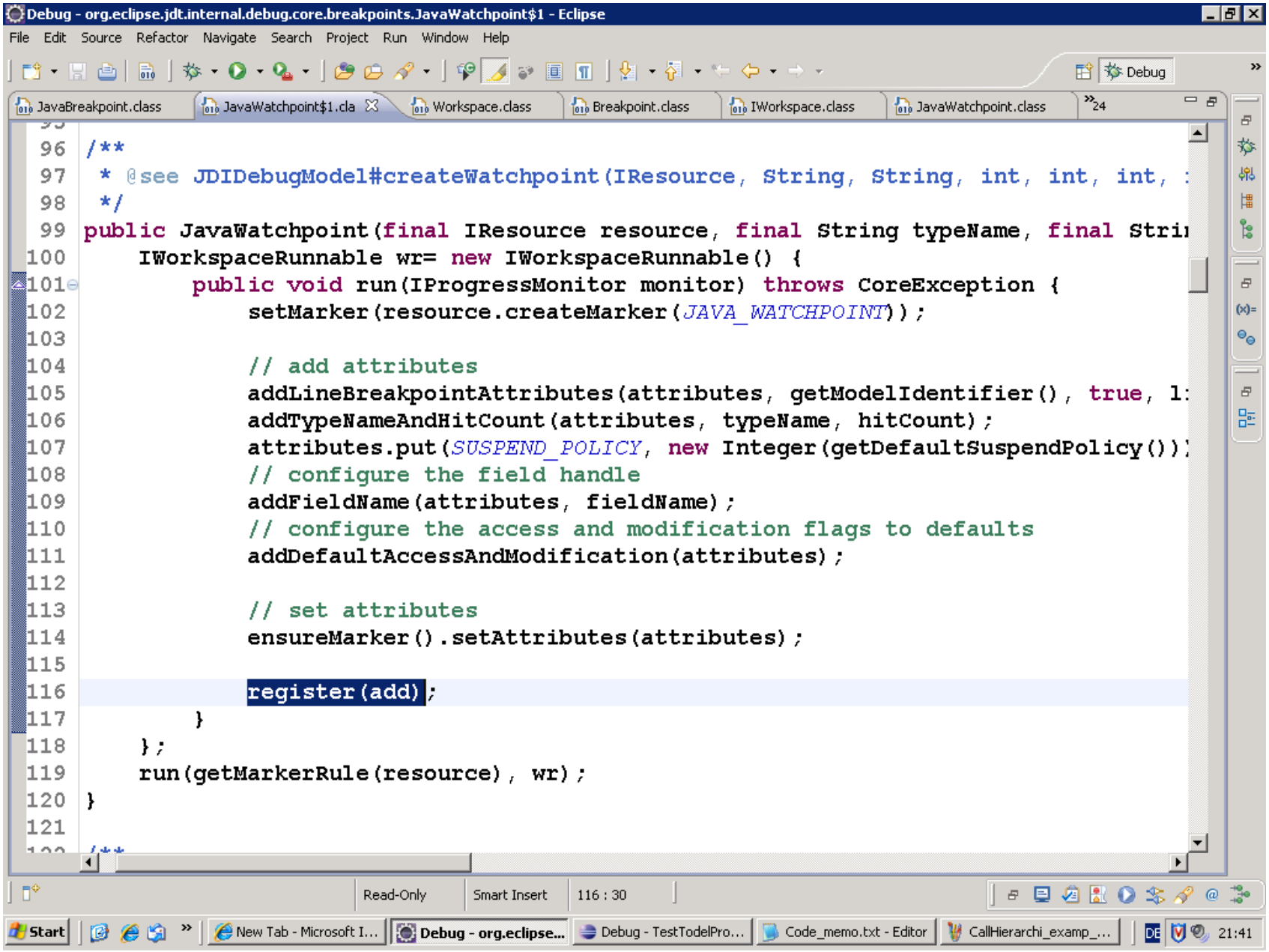
JPDA Integration in Eclipse

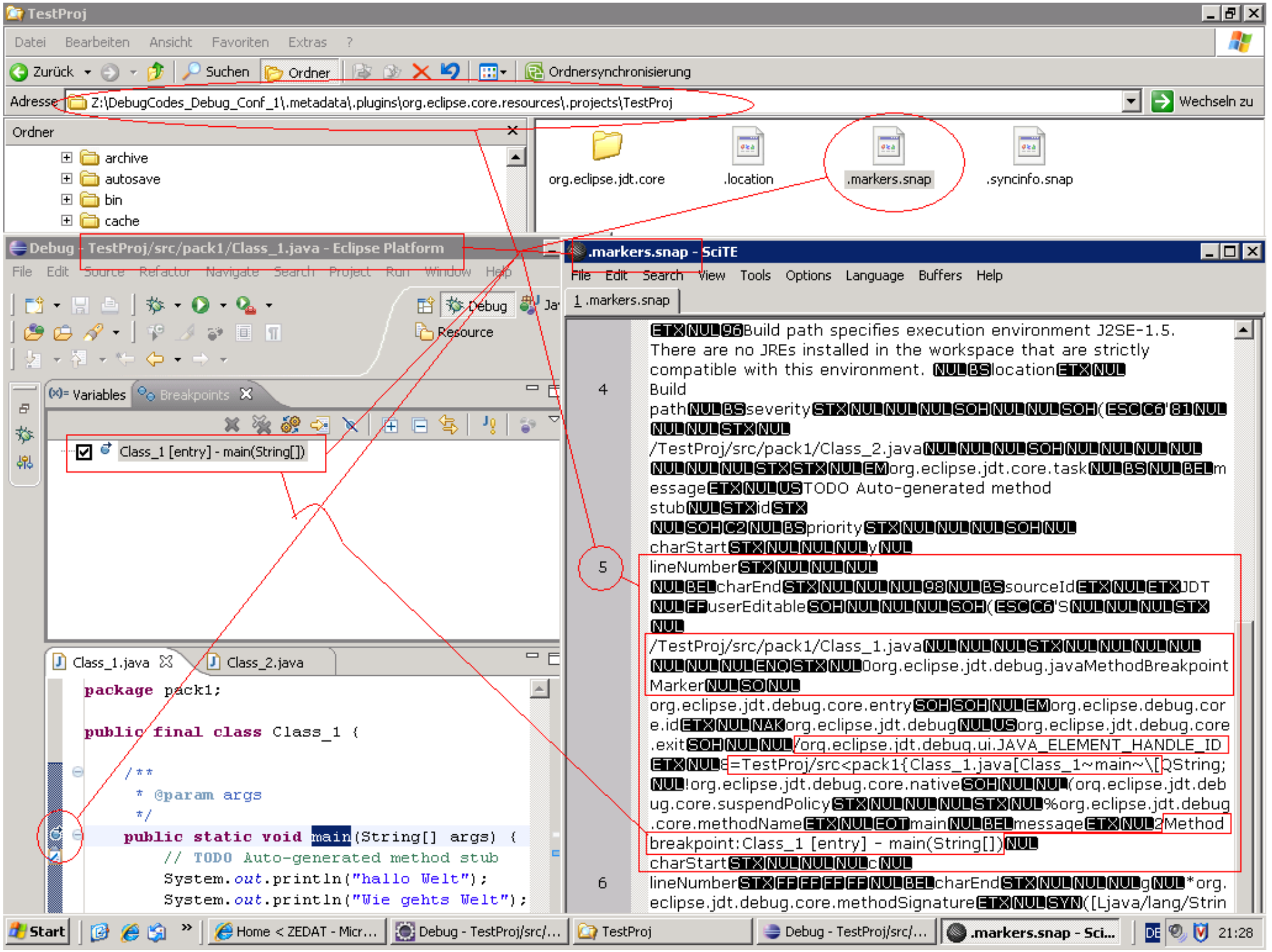


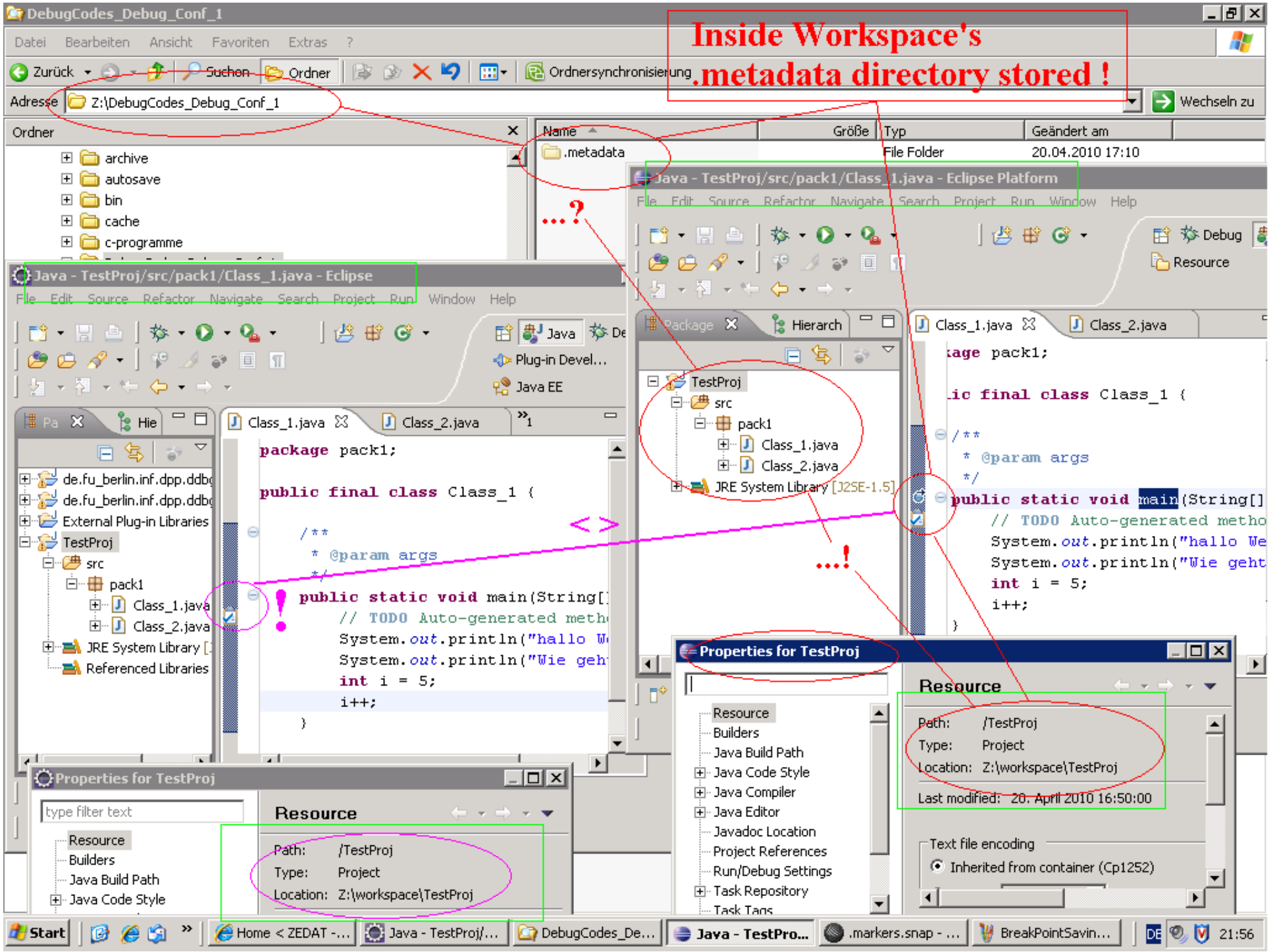
Eclipse Debug View

- Pre-Launch
  - Breakpoints werden gesetzt und mittels Marker persistent gespeichert ; Debug-Plugin verwaltet diese mittels BreakPointManager
  - ToggleBreakpointsAdapter sorgen für UI EventHandling
- Launch
  - Gesammelte Breakpoints werden auf dem Debug-Target (Debuggee) installiert und dabei gleichzeitig als enstprechende EventListener bei diesem registriert


Debug - Eclipse
File Edit Navigate Search Project Run Window Help
Debug
Console Tasks Problems Executables Plug-ins Search Javadoc Call Hierarchy
Members calling 'register(boolean)' - in workspace
register(boolean) : void - org.eclipse.jdt.internal.debug.core.breakpoints.JavaBreakpoint
run(IProgressMonitor) : void - org.eclipse.jdt.internal.debug.core.breakpoints.new IWorkspaceRunnable() {...}
[constructor] new IWorkspaceRunnable() {...} - org.eclipse.jdt.internal.debug.core.breakpoints
[callers]
executeOperation() : void - org.eclipse.jdt.internal.core.BatchOperation
initializeAllContainers(IJavaProject, IPath) : IClasspathContainer - org.eclipse.jdt.internal.core.JavaModelManager
run(IWorkspaceRunnable, ISchedulingRule, int, IProgressMonitor) : void - org.eclipse.core.internal.resources.Workspace
addBreakpoints(IBreakpoint[], boolean) : void - org.eclipse.debug.internal.core.BreakpointManager
doSave(IProgressMonitor) : ILaunchConfiguration - org.eclipse.debug.internal.core.LaunchConfigurationWorkingCopy
flushSyncInfo(QualifiedName, IResource, int) : void - org.eclipse.core.internal.resources.Synchronizer
initializeAllContainers(IJavaProject, IPath) : IClasspathContainer - org.eclipse.jdt.internal.core.JavaModelManager
newDebugTarget(ILaunch, VirtualMachine, String, IProcess, boolean, boolean, boolean) : IDebugTarget - org.eclipse.jdt.debug.
removeBreakpoints(IBreakpoint[], boolean) : void - org.eclipse.debug.internal.core.BreakpointManager
run(IProgressMonitor) : IStatus - org.eclipse.core.internal.events.NotificationManager.NotifyJob
run(IProgressMonitor) : IStatus - org.eclipse.debug.internal.core.BreakpointManager.BreakpointManagerJob
run(ISchedulingRule, IWorkspaceRunnable) : void - org.eclipse.debug.core.model.Breakpoint
JavaClassPrepareBreakpoint(IResource, String, int, int, int, boolean, Map) - org.eclipse.jdt.internal.debug.core.breakpoints.
JavaExceptionBreakpoint(IResource, String, boolean, boolean, boolean, boolean, Map) - org.eclipse.jdt.internal.debug.core
JavaLineBreakpoint(IResource, String, int, int, int, int, boolean, Map, String) - org.eclipse.jdt.internal.debug.core.breakpoint
JavaMethodBreakpoint(IResource, String, String, String, boolean, boolean, boolean, int, int, int, int, boolean, Map) - org.ecl
JavaMethodEntryBreakpoint(IResource, String, String, String, int, int, int, int, boolean, Map) - org.eclipse.jdt.internal.debug
JavaPatternBreakpoint(IResource, String, String, int, int, int, int, boolean, Map, String) - org.eclipse.jdt.internal.debug.core
JavaStratumLineBreakpoint(IResource, String, String, String, String, int, int, int, int, boolean, Map, String) - org.eclipse.jdt.i
JavaTargetPatternBreakpoint(IResource, String, int, int, int, int, boolean, Map, String) - org.eclipse.jdt.internal.debug.core.
JavaWatchpoint(IResource, String, String, int, int, int, int, boolean, Map) - org.eclipse.jdt.internal.debug.core.breakpoints.
createWatchpoint(IResource, String, String, int, int, int, int, boolean, Map) : IJavaWatchpoint - org.eclipse.jdt.debug.cc
run(IWorkspaceRunnable, IProgressMonitor) : void - org.eclipse.core.internal.resources.Workspace
run(IWorkspaceRunnable, ISchedulingRule, IProgressMonitor) : void - org.eclipse.jdt.core.JavaCore
runOperation(IProgressMonitor) : void - org.eclipse.jdt.internal.core.JavaModelOperation
save() : void - org.eclipse.core.internal.resources.ProjectPreferences (2 matches)
setAttribute(String, boolean) : void - org.eclipse.debug.core.model.Breakpoint
setAttribute(String, int) : void - org.eclipse.debug.core.model.Breakpoint
setAttribute(String, Object) : void - org.eclipse.debug.core.model.Breakpoint
setAttributes(Map) : void - org.eclipse.debug.core.model.Breakpoint
Line
Call
119
run(getMarkerRule(resource), wr)
org.eclipse.jdt.internal.debug.core.breakpoints.JavaWatchpoint.Java...ributes) throws DebugException - org.eclipse.jdt.debug/jdimodel.jar
Start
New Tab - Microsoft Inte...
Debug - Eclipse
Debug - TestTodelProj/sr...
Code_memo.txt - Editor
21:38


Debug - org.eclipse.jdt.internal.debug.core.breakpoints.JavaWatchpoint$1 - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
JavaBreakpoint.class
JavaWatchpoint$1.cla
Workspace.class
Breakpoint.class
IWorkspace.class
JavaWatchpoint.class
24
/**
* @see JDIDebugModel#createWatchpoint(IResource, String, String, int, int, int,
*/
public JavaWatchpoint(final IResource resource, final String typeName, final Stri
IWorkspaceRunnable wr= new IWorkspaceRunnable() {
public void run(IProgressMonitor monitor) throws CoreException {
setMarker(resource.createMarker(JAVA_WATCHPOINT));
// add attributes
addLineBreakpointAttributes(attributes, getModelIdentifier(), true, l
addTypeNameAndHitCount(attributes, typeName, hitCount);
attributes.put(SUSPEND_POLICY, new Integer(getDefaultSuspendPolicy()));
// configure the field handle
addFieldName(attributes, fieldName);
// configure the access and modification flags to defaults
addDefaultAccessAndModification(attributes);
// set attributes
ensureMarker().setAttributes(attributes);
register(add);
}
};
run(getMarkerRule(resource), wr);
}
Read-Only
Smart Insert
116 : 30
Start
New Tab - Microsoft I...
Debug - org.eclipse...
Debug - TestTodelPro...
Code_memo.txt - Editor
CallHierarchi_examp_...
21:41


TestProj
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Ordner Ordnersynchronisierung
Adresse Z:\DebugCodes_Debug_Conf_1\.metadata\.plugins\org.eclipse.core.resources\.projects\TestProj
Wechseln zu
Ordner
archive
autosave
bin
cache
org.eclipse.jdt.core
.location
.markers.snap
.syncinfo.snap
Debug - TestProj/src/pack1/Class_1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Resource
Variables
Breakpoints
Class_1 [entry] - main(String[])
Class_1.java
Class_2.java
package pack1;
public final class Class_1 {
/**
* @param args
*/
public static void main(String[] args) {
// TODO Auto-generated method stub
System.out.println("hallo Welt");
System.out.println("Wie gehts Welt");
.markers.snap - SciTE
File Edit Search View Tools Options Language Buffers Help
1 .markers.snap
4
5
6
Build path specifies execution environment J2SE-1.5. There are no JREs installed in the workspace that are strictly compatible with this environment.
/TestProj/src/pack1/Class_2.java
org.eclipse.jdt.core.task
TODO Auto-generated method stub
/TestProj/src/pack1/Class_1.java
org.eclipse.jdt.debug.javaMethodBreakpointMarker
org.eclipse.jdt.debug.core.entry
org.eclipse.jdt.debug.core.id
org.eclipse.jdt.debug
org.eclipse.jdt.debug.core.exit
/org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID
=TestProj/src<pack1{Class_1.java[Class_1~main~\[QString;
org.eclipse.jdt.debug.core.nativ
(org.eclipse.jdt.debug.core.suspendPolicy
%org.eclipse.jdt.debug.core.methodName
main
message
Method breakpoint:Class_1 [entry] - main(String[])
charStart
charEnd
lineNumber
sourceId
JDT
userEditable
org.eclipse.jdt.debug.core.methodSignature
([Ljava/lang/Strin
Start
Home < ZEDAT - Micr...
Debug - TestProj/src/...
TestProj
Debug - TestProj/src/...
.markers.snap - Sci...
21:28

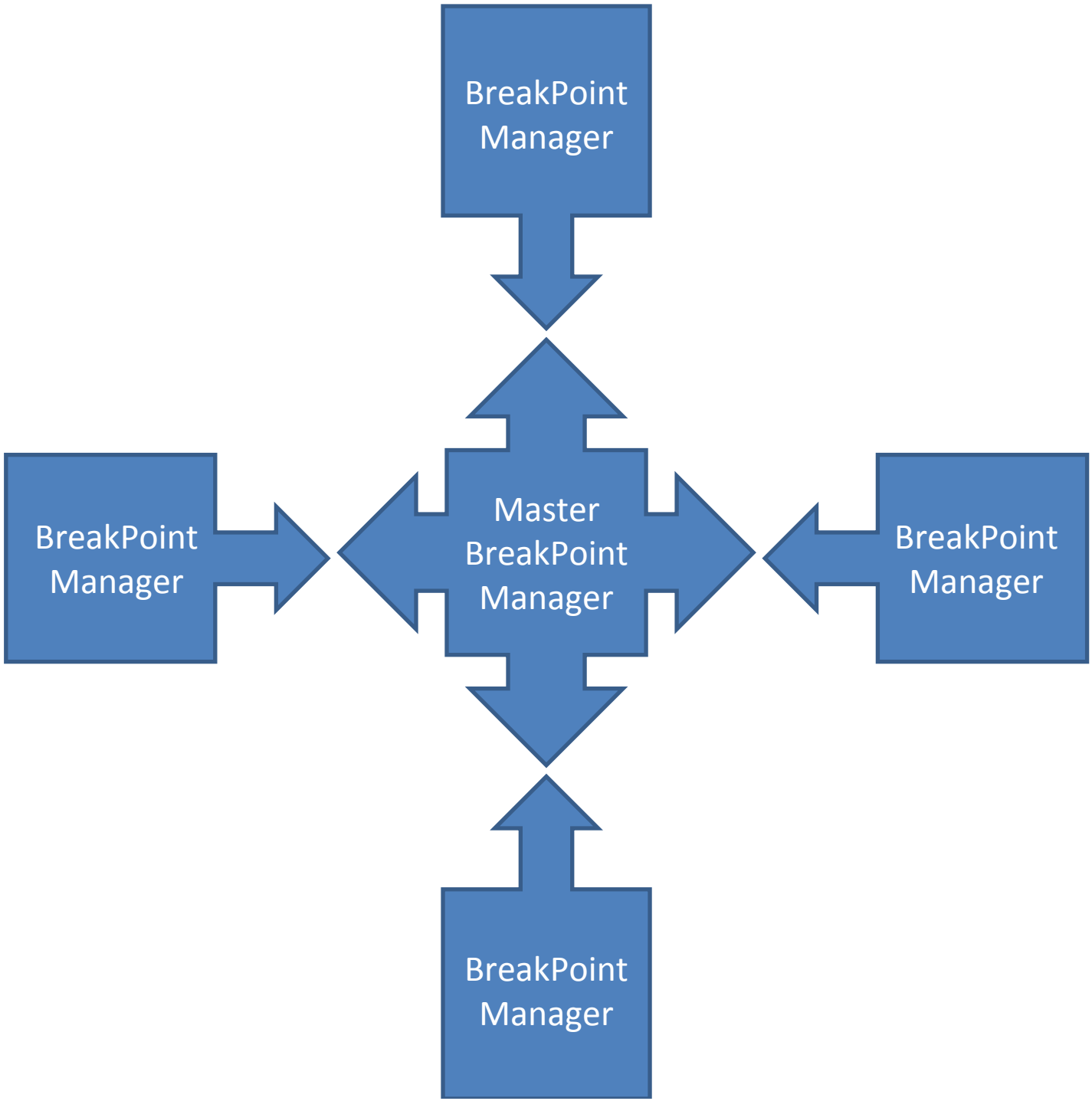

Inside Workspace's
.metadata directory stored !
DebugCodes_Debug_Conf_1
Adresse Z:\DebugCodes_Debug_Conf_1
Wechseln zu
Ordner
archive
autosave
bin
cache
c-programme
Name
Größe
Typ
Geändert am
.metadata
File Folder
20.04.2010 17:10
...?
Java - TestProj/src/pack1/Class_1.java - Eclipse
Java - TestProj/src/pack1/Class_1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Resource
Java
Plug-in Devel...
Java EE
Package
Hierarch
TestProj
src
pack1
Class_1.java
Class_2.java
JRE System Library [J2SE-1.5]
External Plug-in Libraries
Referenced Libraries
de.fu_berlin.inf.dpp.ddb
package pack1;
public final class Class_1 {
/**
* @param args
*/
public static void main(String[]
// TODO Auto-generated metho
System.out.println("hallo We
System.out.println("Wie geht
int i = 5;
i++;
}
...!
Properties for TestProj
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Task Repository
Task Tags
Path: /TestProj
Type: Project
Location: Z:\workspace\TestProj
Last modified: 20. April 2010 16:50:00
Text file encoding
Inherited from container (Cp1252)
Start
Home < ZEDAT -...
Java - TestProj/...
DebugCodes_De...
Java - TestPro...
.markers.snap - ...
BreakPointSavin...
21:56

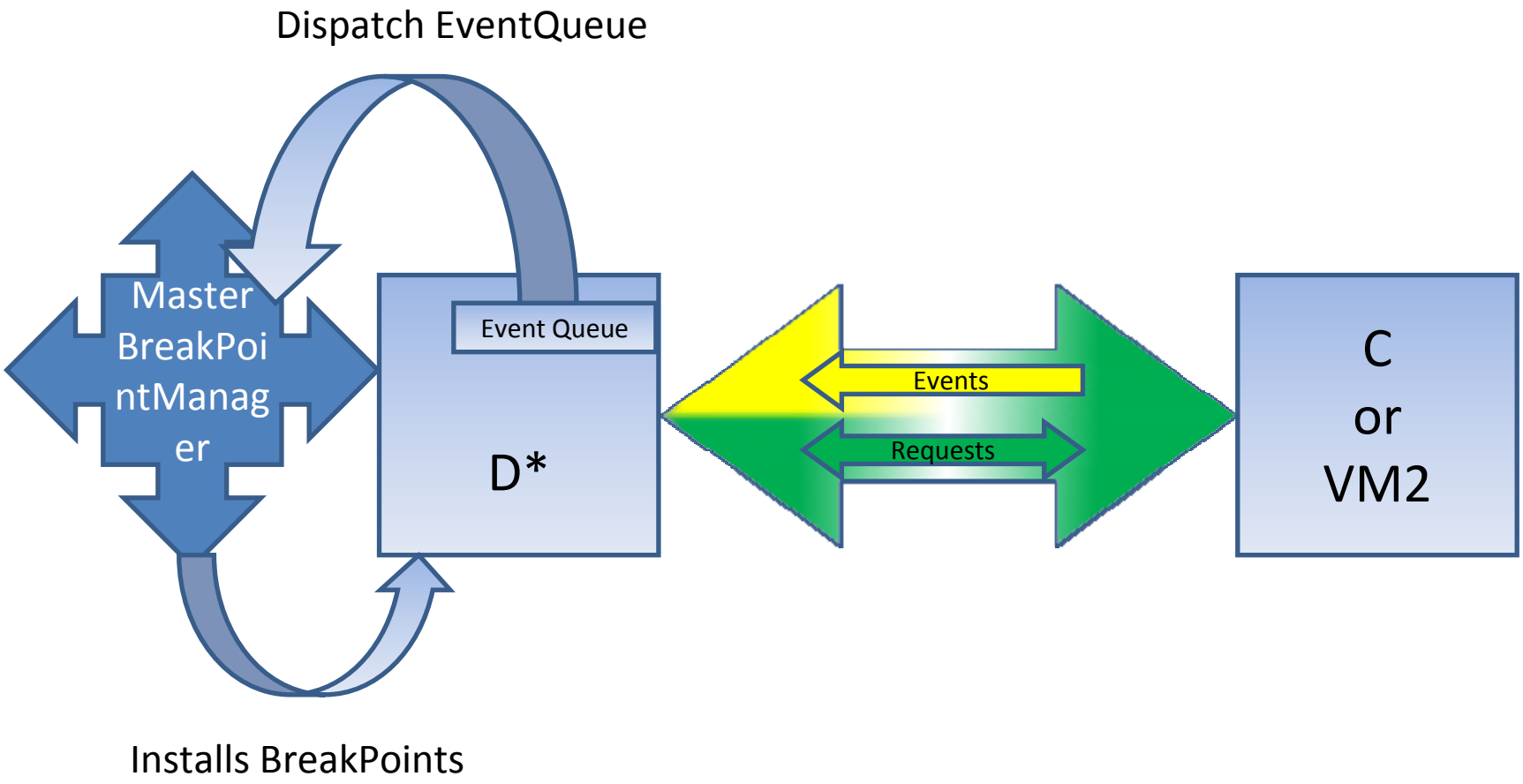
Grundlegende Idee

- Delegiere alle UI Events an zentrales Single-User Debug-Datenmodell; Input Multiplexing und Output-Forwarding (Datenabfrage)
- Repliziere alle DebugEvents der Eventqueue, so daß Sie bei jedem Benutzer einen Update des Datenmodells signalisieren

Idee

- Pre-Launch
  - Zentraler BreakPointManager als Verteiler von lokalen BreakPointManagern
- Launch
  - Zentraler BreakpointManager installiert BreakPoints im Debug-Target.


BreakPoint Manager
BreakPoint Manager
Master BreakPoint Manager
BreakPoint Manager
BreakPoint Manager


Dispatch EventQueue
Master BreakPoi ntManag er
Event Queue
D*
Events
Requests
C or VM2
Installs BreakPoints

Literatur

- A High-Level and Flexible Framework for Implementing Multi-User User-Interfaces
  PRASUN DEWAN und RAJIV CHOUDHARY
  1992 Purdue University
- Eclipse Infrastructure, Chapter 3; Eric Clayberg