JAVAFORUM 2006 stuttgart

Monitoring und Profiling von Enterprise-Anwendungen – ein Technologievergleich

Gerhard Wanner (info@wanner-it.de)

Agenda

JAVAFORUM 2006 stuttgart

- Definition/Motivation
- Verfügbare Technologien
  - Performance Profiling
  - Ressourcen Profiling
  - Profiling/Monitoring mit Bytecode-Modifikation
  - Monitoring mit JMX
- Fallbeispiele
- Werkzeugeinsatz im Projekt
- Fazit.

Monitoring, Profiling – Definition

JAVAFORUM 2006 stuttgart

- **Profiling**

Als Profiling wird der Einsatz von Programmwerkzeugen bezeichnet, die das Laufzeitverhalten von Software analysieren.
Es gibt unterschiedliche Problembereiche in der Softwareentwicklung, die durch ineffiziente Programmierung ausgelöst werden. Profiling hilft dem Entwickler durch Analyse und Vergleich von laufenden Programmen die Problembereiche aufzudecken.

- **Monitoring**

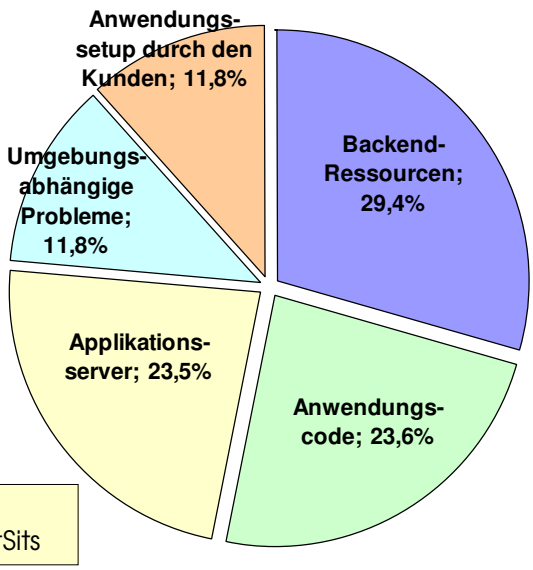
Monitoring beschreibt die Erfassung von Zuständen, Beobachtung, Überwachung oder Steuerung eines Vorgangs oder Prozesses mittels technischer Hilfsmittel.
Ein Monitoringsystem ermöglicht zum Teil auch Eingriffe in die betreffenden Prozesse, sofern sich abzeichnet, dass der Prozess nicht den gewünschten Verlauf nimmt.

Angelehnt an: www.wikipedia.org

Monitoring, Profiling – Motivation

JAVAFORUM 2006 stuttgart

- **Bei Einführung und Betrieb einer Enterprise-Anwendung treten verschiedenartige Probleme auf**
  - Dauerhafte Performance-Probleme
  - Über die Zeit anwachsende Performance-Probleme
  - Sporadische Probleme
  - Dauerhafte Fehler
  - Überlast
- **Ursache der Probleme ist vielschichtig**
- **Monitoring/Profiling dient dazu, die genannten Probleme aufzufinden.**



Quelle: IBM Case Studies of WebSphere Performance CritSits

Monitoring, Profiling – Einführung

JAVAFORUM 2006 stuttgart

Mögliche Performance-Probleme…

- und Werkzeuge für ihre Untersuchung.

Angelehnt an:
Tim Vanderham, Raleigh WebSphere User Group, 2003

	Lasttest-Werkzeug	Profiling-Werkzeug (Performance)	Profiling-Werkzeug (Ressourcen)	Monitoring Werkzeug	System-Monitor
Anwendungsfehler	X	X	X	X	
HTTP-Session Missmanagement	X	X	X	X	
Schlecht definierte Pool-Größen	X		X	X	X
Ressourcen der Backend-Systeme	X			X	X
Timeout-Werte schlecht definiert	X			X	X
Überlastung des Netzwerks	X			X	
…					

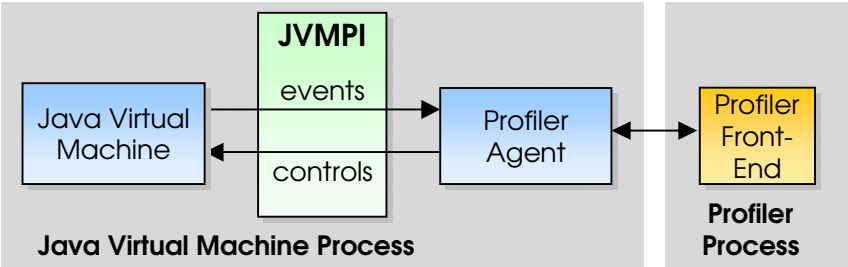
Technologien – Performance Profiling (1)

JAVAFORUM 2006 stuttgart

Performance-Profiling erfolgt üblicherweise …

- mit der Arme-Leute Variante: `System.currentTimeMillis()`
  - Geeignet um mal schnell was zu messen
- etwas komfortabler: z.B. mit JAMon (Java Application Monitor)
  - Nutzt intern `System.currentTimeMillis()`
  - API erlaubt eine komfortable Benutzung
    - Mehrere Hierarchieebenen
    - Umfangreiches Reporting
  - Open Source (http://jamonapi.sourceforge.net)

- optimal Integriert: JVMPI (Java Virtual Machine Profiler Interface)
  - Ermöglicht die Sammlung von Informationen die deutlich über die reine Zeitmessung hinausgehen.

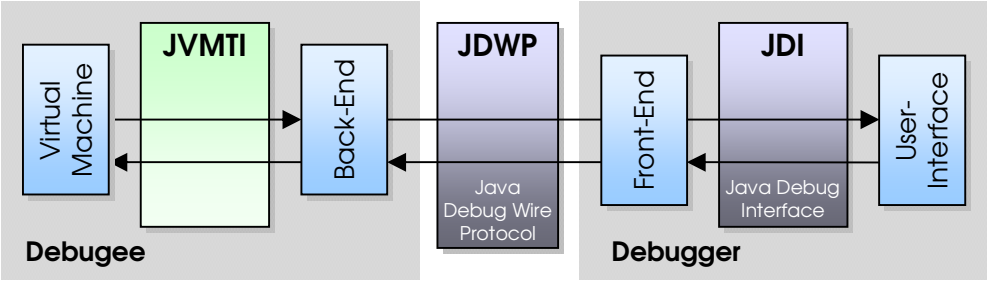


Angelehnt an:
JavaTM Virtual Machine Profiler Interface (JVMPI), v1.2, Sun, 2002

Technologien – Performance Profiling (2)

JAVAFORUM 2006 stuttgart

- **JVMPI**
  - Eingeführt mit JDK 1.1
  - Seither als "experimentell" gekennzeichnet
    - Nie wirklich stabil mit den "HotSpot"-VMs
    - Nicht lauffähig mit bestimmten GC-Implementierungen
  - Seit Java 5 *deprecated*
    - Im nächsten Major Release (Java 6) nicht mehr enthalten
- **Daher: JVMTI (Java Virtual Machine Tool Interface)**
  - Löst JVMPI (und auch JVMDI - Java Virtual Machine Debug Interface) ab
  - Neben Profiling insbesondere auch Einsatz im Bereich Debugging.

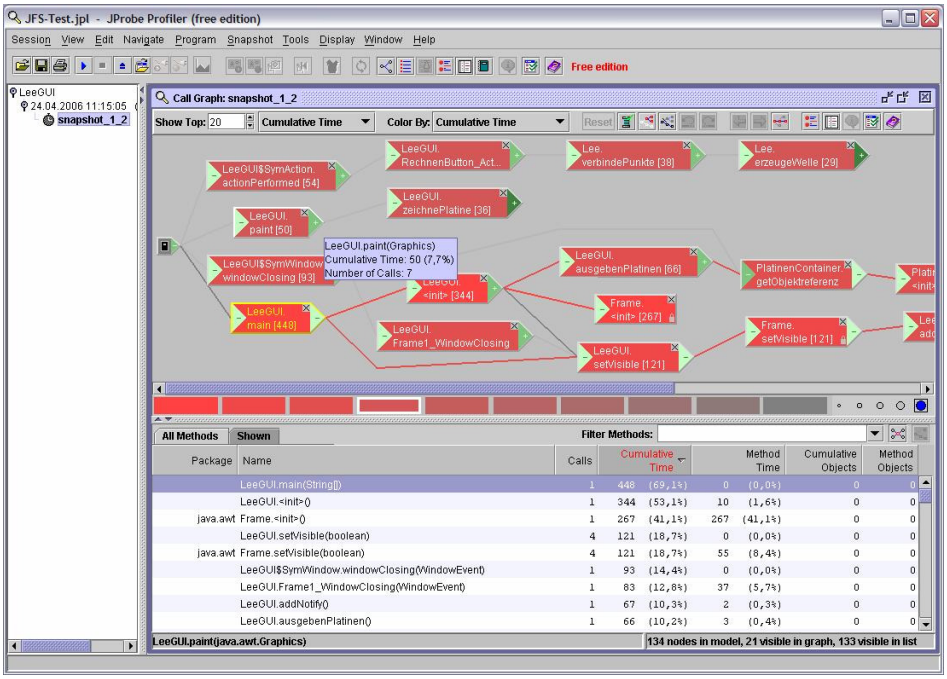


Angelehnt an: Java™ Platform Debugger Architecture, Sun, 2004

Technologien – Performance Profiling (3)

JAVAFORUM 2006 stuttgart

- **Beispiel: Quest JProbe**
  - Genaue Aufschlüsselung der Aufrufe über Call Graph
    - Wer ruft wen wie oft auf?
    - Laufzeiten
  - Top-Down-Analyse
  - Bildung von Vergleichen über Snapshots
  - …

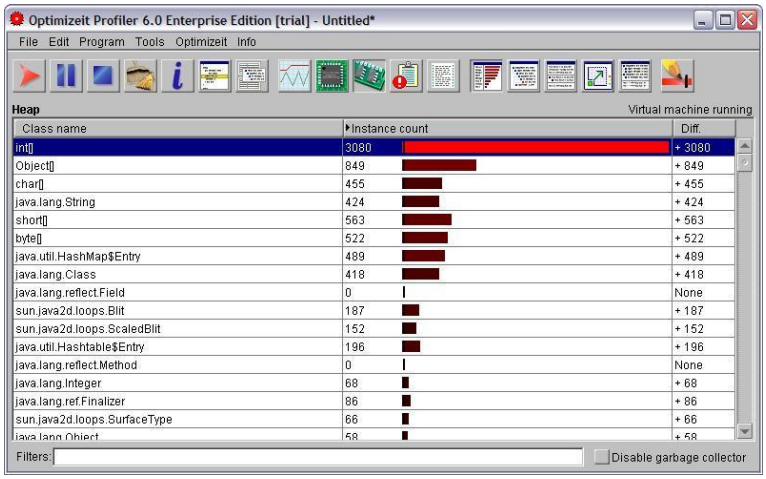


Beispiel: Quest JProbe

Technologien – Ressourcen Profiling (1)

JAVAFORUM 2006 stuttgart

- **Ressourcen-Profiling…**
  - erfolgt meist mittels JVMPI/JVMTI
    - Liefert Informationen über die Objektinstanzierung (welche Objekte in welcher Methode?)
    - Möglichkeit zur Inspektion des Heaps
    - Auch als "Memory-Debugging" bezeichnet
- **Beispiel: Borland Optimizeit**
  - Aufschlüsselung der Anzahl erzeugter Instanzen
  - Top-Down-Analyse
  - Bildung von Vergleichen über Snapshots
  - …

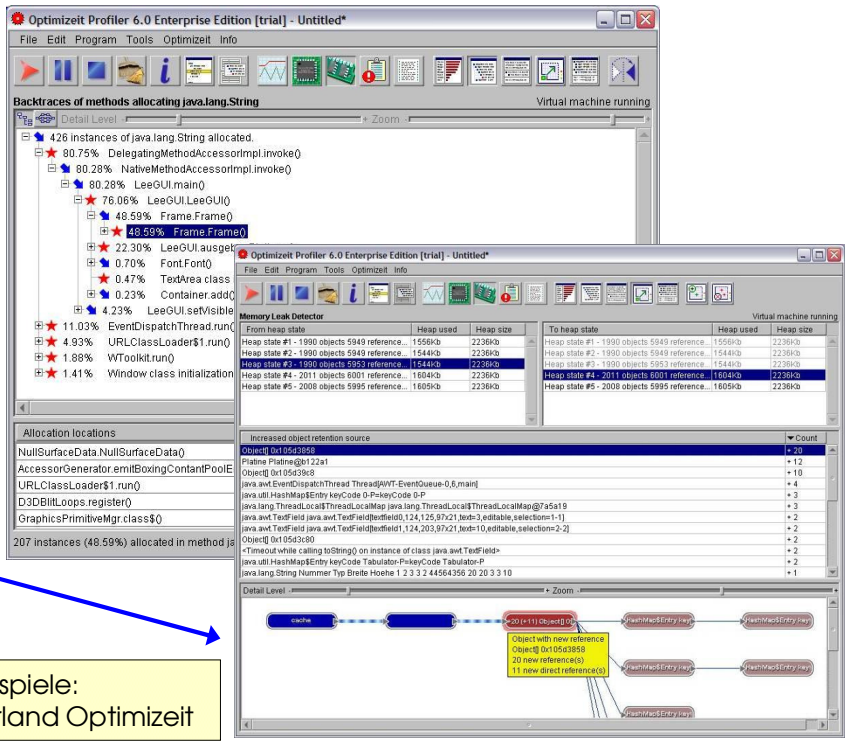


Beispiel: Borland Optimizeit

Technologien – Ressourcen Profiling (2)

JAVAFORUM 2006 stuttgart

- **Beispiel (Forts.): Borland Optimizeit**
  - Präzise Lokalisierung des Orts der Instanzierung
    - Unterschiedliche Analyse-möglichkeiten (z.B. Drill-/Top-Down)
  - Bilden von Heap-Deltas während der Programm-ausführung
    - "Halbautomatische" Suche nach Speicherlecks.

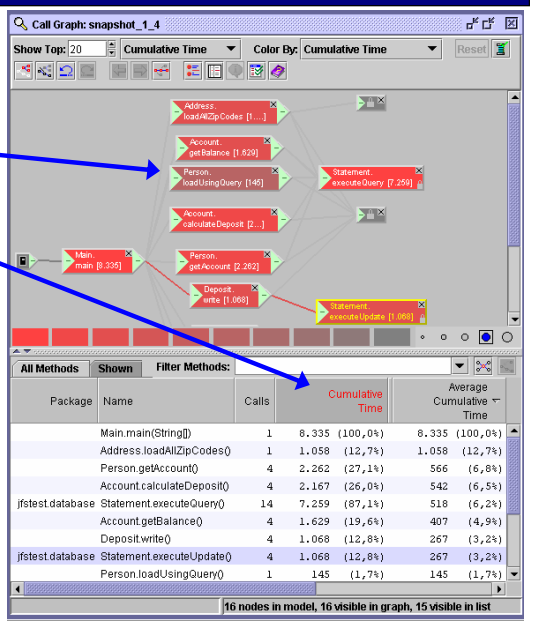


Beispiele: Borland Optimizeit

Technologien – Profiling/Monitoring mit Bytecode-Modifikation (1)

JAVAFORUM 2006 stuttgart

- **Schwächen herkömmlicher Profiler**
  - Lange Laufzeiten
  - Interpretation z.T. schwierig
  - Nur Ausgabe der Laufzeiten
  - Manche Fragestellungen sind nicht bzw. kaum beantwortbar
- **Daher: Bytecode-Instrumentierung**
  - Umsetzung aspektorientierter Programmierung
    - Anwendung frei von Anweisungen für Monitoring/Profiling
    - **Point Cuts** erlauben die schnelle Instrumentierung vieler Methoden
    - Für detaillierteres Monitoring/Profiling ist keine Quellcode-Änderung notwendig
    - Fremdsoftware kann instrumentiert und damit einem Profiling/Monitoring unterzogen werden.



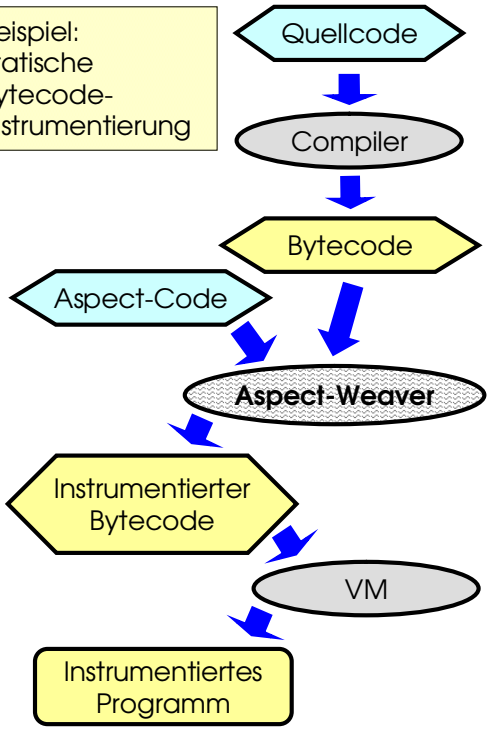
Beispiel: Quest JProbe

Technologien – Profiling/Monitoring mit Bytecode-Modifikation (2)

JAVAFORUM 2006 stuttgart

- **Durchführung der Bytecode-Instrumentierung**
  - ***statisch***: *.class* Dateien werden geladen, verändert und in veränderter Form wiederum als *.class* Dateien gespeichert
  - ***load-time***: Klassen werden beim Laden in die VM verändert
    - Es existieren zu keinem Zeitpunkt persistent veränderte *.class* Dateien
  - ***dynamisch***: Klassen werden nach dem Laden in die VM verändert (nur Java 5)
    - Änderungen können mehrfach ausgeführt und auch wieder rückgängig gemacht werden
    - Änderungen können auch durchgeführt werden, wenn bereits Instanzen existieren
    - Es existieren zu keinem Zeitpunkt persistent veränderte *.class* Dateien.

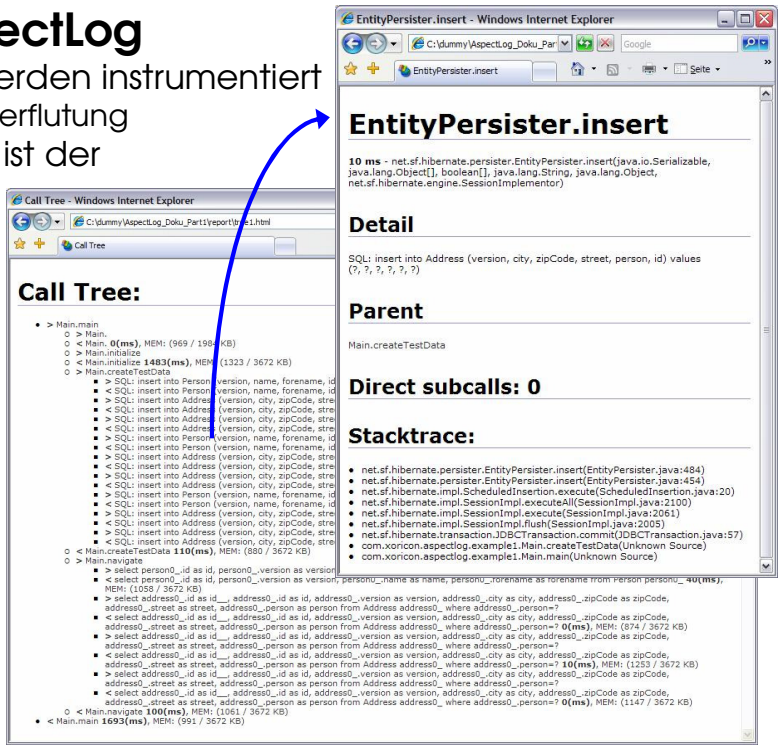
Beispiel: Statische Bytecode-Instrumentierung



Technologien – Profiling mit Bytecode-Modifikation (1)

JAVAFORUM 2006 stuttgart

- **Beispiel: XORICON AspectLog**
  - Nur relevante Methoden werden instrumentiert
    - Verhindert Informations-Überflutung
  - Für jeden Methoden-Aufruf ist der Stacktrace abfragbar
    - Ermöglicht Aufruf-Rückverfolgung
    - Beinhaltet auch nicht instrumentierte Methoden
  - Java-API
    - Für eigene, anwendungs-spezifische Aggregationen
  - Schwellwert-Definitionen
    - machen auf hohen Ressourcen-Verbrauch aufmerksam.

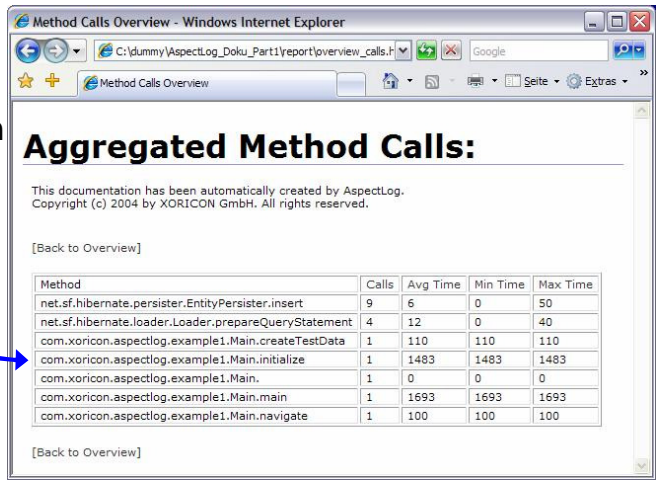


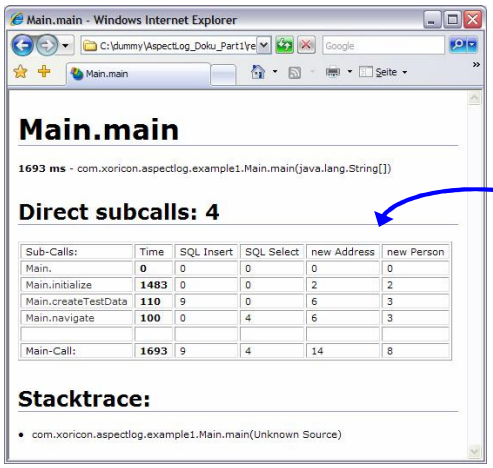
Beispiele:
XORICON AspectLog

Technologien – Profiling mit Bytecode-Modifikation (2)

JAVAFORUM 2006 stuttgart

- **Beispiel (Forts.): XORICON AspectLog**
  - Erweiterung um aggregierte Daten
    - Neben der Laufzeit können aggregierte Daten die Lösungsfindung sehr vereinfachen





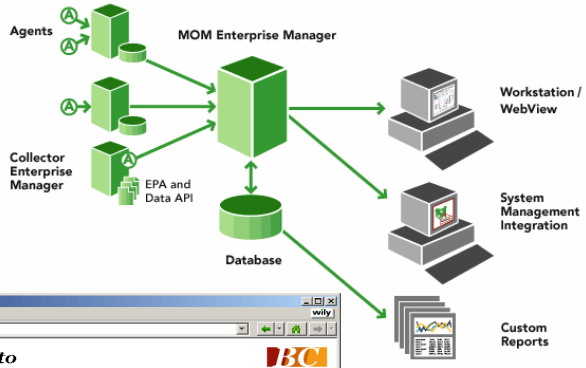
- z.B. Anzahl der …
  - Remote-Calls
  - Zugriffe auf Backend-System (DB / Host etc.)
  - IO-Zugriffe mit Anzahl der übertragenen Bytes
    Objekterzeugungen
  - Ausnahmen die geworfen/erzeugt wurden.

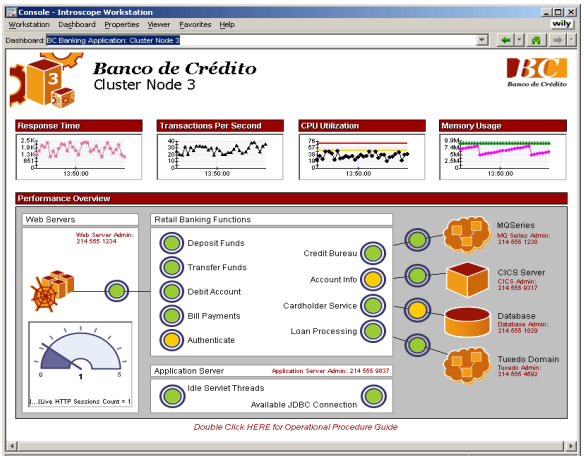
Beispiele:
XORICON AspectLog

Technologien – Monitoring mit Bytecode-Modifikation

JAVAFORUM 2006 stuttgart

- **Beispiel: wily Introscope**
  - Nutzung verschiedener Informationsquellen für das Monitoring
    - Insbesondere Bytecode-Instrumentierung
    - Aber auch z.B. SNMP, JMX
  - Instrumentierung des Bytecodes
    - Ermöglicht den Agents das Einsammeln von Daten vom überwachten Knoten
    - Umfang der Instrumentierung kann festgelegt werden
    - Im Produktionsbetrieb einsetzbar.





Abbildungen: wily technology

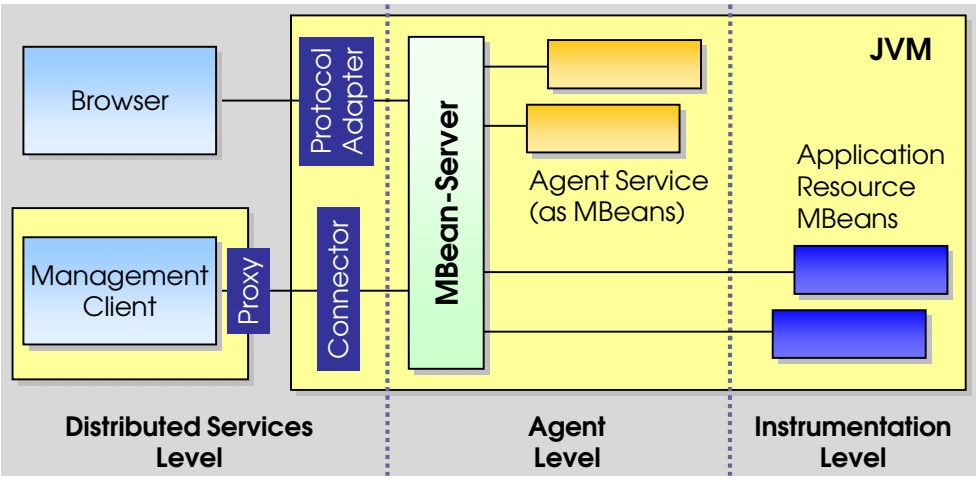
Technologien – Monitoring mit JMX (1)

JAVAFORUM 2006 stuttgart

- **Java Management Extensions (JMX) sind …**
  - eine Architektur
  - Design Patterns
  - APIs und
  - Services für Applikations- und Netzwerk-Management und Monitoring

Angelehnt an: Java Management Extensions Instrumentation and Agent Specification, v1.2, Sun, 2002

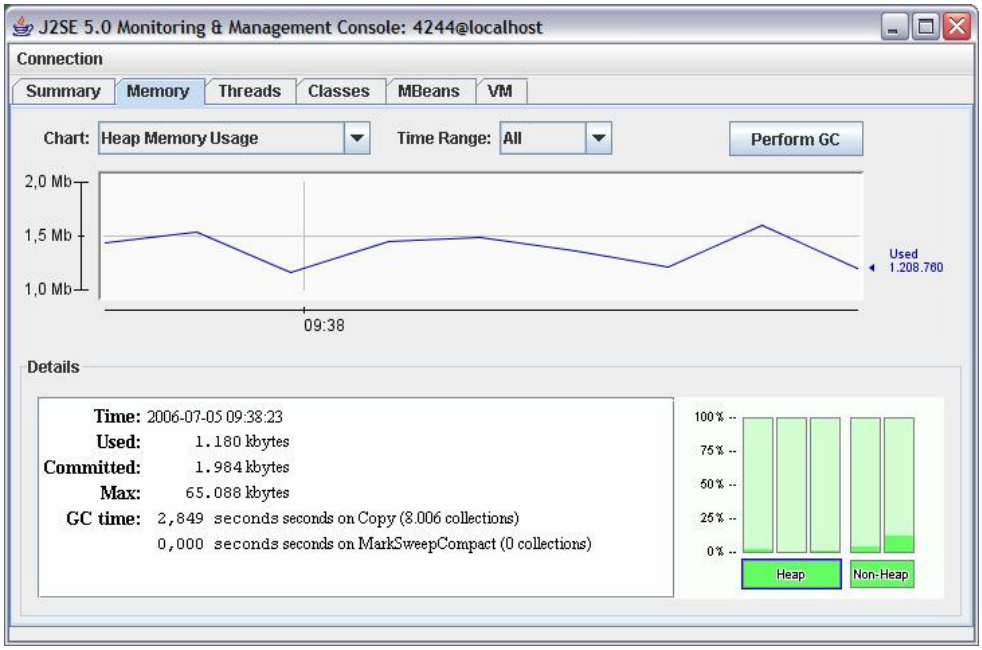
- **JMX …**
  - ermöglichen das Management von Java-Anwendungen mit geringen Eingriffen.



Technologien – Monitoring mit JMX (2)

JAVAFORUM 2006 stuttgart

- **Seit Java 5 kann der Status der JVM über JMX abgefragt werden**
  - Bereitstellung von Platform MBeans (MXBeans)
    - Klassen
    - Speicher
    - Threads
    - Runtime-Inform.
    - GC
    - …

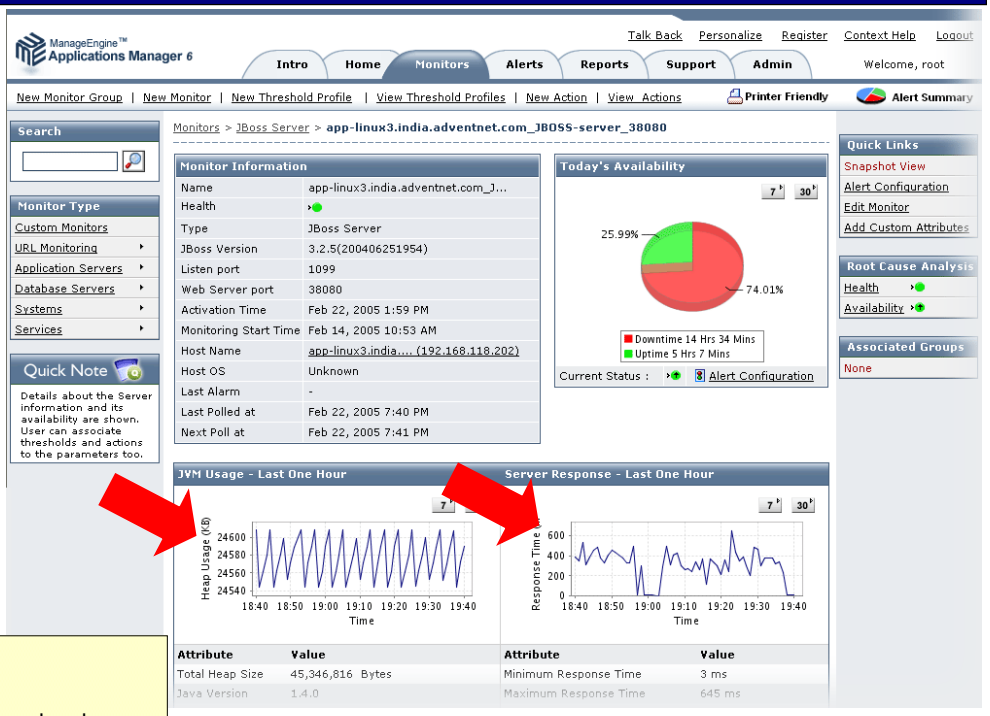


Beispiel:
Sun JSE - JConsole

Technologien – Monitoring mit JMX (3)

JAVAFORUM 2006 stuttgart

- **JMX …**
  - ist in die meisten Applikations-Server bereits integriert
    - z.B. JBoss
  - liefert die Grundlage für Hersteller von Monitoring-Werkzeugen.



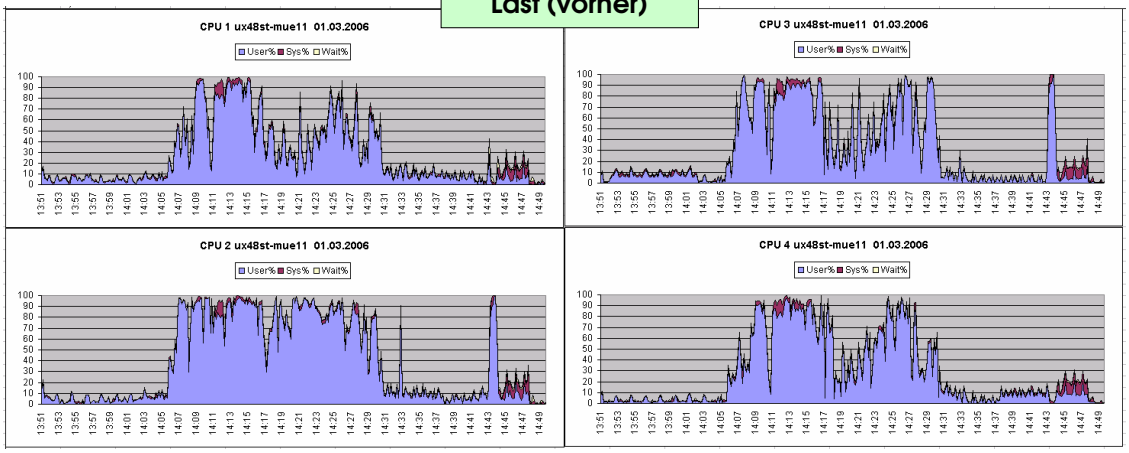
Quelle:
ManageEngine
http://manageengine.adventnet.com

Fallbeispiel 1 (1)

JAVAFORUM 2006 stuttgart

- **Durchführung eines "Interaktiven Lasttests"**
- **Problem**
  - Test führt bereits bei geringer Benutzeranzahl (<10% der tatsächlichen Endbenutzer) zu voller Systemauslastung auf dem Application-Server
  - Die in den nichtfunktionalen Anforderungen definierten Antwortzeiten werden weit überschritten.

Last (vorher)



Quelle: BG Phoenics

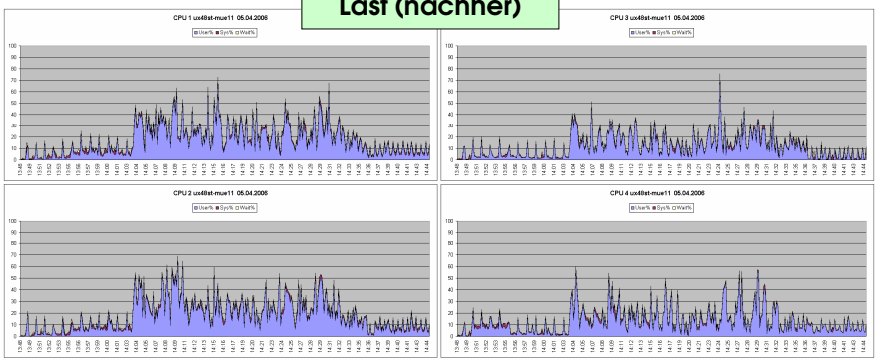
Fallbeispiel 1 (2)

JAVAFORUM 2006 stuttgart

- **Vorgehensweise**
  - Benutzung herkömmliches Profiling-Werkzeug (z.B. JProbe) nicht möglich
    - Startzeit des Servers steigt von 1-2 Min. auf >45 Min. wegen Vorabladen statischer Objekte (ca. 300 MB) aus der Datenbank
  - **Monitoring** (Application Response Measurement) erkennt den Server-Call, der sehr lange dauert
  - Über **Profiling** (AspectLog) kann Problem gefunden werden
    - Aggregation (wegen Vielzahl der Server-Calls) und Schwellwerte
- **Ursache / Lösung**
  - Ursache war das sequenzielle Durchsuchen einer Vielzahl von Objekten auf dem Server bei der Objekt-Navigation
  - Vorab-Auflösen der Referenzierung.

Quelle: BG Phoenics

Last (nachher)



Fallbeispiel 2 (1)

JAVAFORUM 2006 stuttgart

- **Signalisierung von Schwellwerten bei Durchführung von Entwicklertests**
  - Entwickler erkennt oft nicht die Tragweite seiner Implementierung
  - Werkzeugeinsatz zur Dokumentation "auffälliger" Codestellen
    - Report wird automatisch während Testausführung erzeugt
    - Muss nicht zwangsläufig ein Problem sein, ist aber ein starker Indikator.

Quelle: BG Phoenics

Overview

This documentation has been automatically created by AspectLog.
Copyright (c) 2004-2005 by XORICON GmbH. All rights reserved.

Aggregated Method Calls

Overview aggregated method calls

Root Calls

BootstrapSessionPOAInvokeHandler.getUserAssignmentData **211 ms** -- Free Mem: (512411) (KB)
BootstrapSessionPOAInvokeHandler.createServiceSession **440 ms** -- Free Mem: (511803) (KB)
GdaServiceBean.ejbCreate **0 ms** -- Free Mem: (511387) (KB)
GdaServicePOAInvokeHandler.find **61 ms** -- Free Mem: (511933) (KB)
WorkflowServicePOAInvokeHandler.wfSearchStartableProcessDefinitionTypes **60 ms** -- Free Mem: (511943) (KB)
WorkflowServicePOAInvokeHandler.create **80 ms** -- Free Mem: (510949) (KB)
UamSessionPOAInvokeHandler.getUserData **90 ms** -- Free Mem: (507668) (KB)
AfPOAInvokeHandler.getContextObject **0 ms** -- Free Mem: (508929) (KB)
UamSessionPOAInvokeHandler.hasCompetence **111 ms** -- Free Mem: (508758
AfPOAInvokeHandler.executeAgent **39358 ms** -- Free Mem: (505950) (KB)
WorkflowServicePOAInvokeHandler.wfCreateWorkItemData **20 ms** -- Free Mem: (505920) (KB)
SearchPOAInvokeHandler.execute **110 ms** -- Free Mem: (504471) (KB)
SearchPOAInvokeHandler.getNextRows **0 ms** -- Free Mem: (505942) (KB)
AfPOAInvokeHandler.executeAgent **110 ms** -- Free Mem: (504642) (KB)
AfPOAInvokeHandler.checkConstraints **10 ms** -- Free Mem: (505506) (KB)
AfPOAInvokeHandler.completeVg **16516 ms** -- Free Mem: (505277) (KB)
AfPOAInvokeHandler.complete **0 ms** -- Free Mem: (503550) (KB)
AfPOAInvokeHandler.getGPContext **0 ms** -- Free Mem: (505322) (KB)

Servercall dauert ca. 40 s

Servercall dauert ca. 16 s

Fallbeispiel 2 (2)

JAVAFORUM 2006 stuttgart

- **Zoom in die kritischen Stellen klärt Auffälligkeit**
  - Drill-Down ermöglicht Lokalisierung eines möglichen Problems
  - Im Beispiel werden viel zu viele SQLs abgesetzt.

Quelle: BG Phoenics

Hauptbuchungsgruppe.erstelleBuchungsgruppen

36135 ms - de.bgnet.zw.abs.auftrag.Hauptbuchungsgruppe.erstelleBuchungsgruppen(de.bgnet.zw.bss.zwdatentyp.DtBelegartNummerkreis, de.bgnet.fwk.ekomp.komm.dt.DtDatum)

Detail

AGO: Hauptbuchungsgruppe.erstelleBuchungsgruppen

Parent

AGENT: FiwiUmbuchungenFeststellenAnordnenVgAgentSkel.erstelleBuchungsgruppeUndBestimmeBuchungsdatum

Direct subcalls: 2

Sub-Calls:	Time	Ago	GDA Api	Mat.Ego	SQL Select	new Dt
Hauptbuchungsgruppe.createBuchungssatzeZuKonntierungen	**35014**	5056	572	668	667	13690
Hauptbuchungsgruppe.createBuchungsgruppenZuBuchungssaetze	**1121**	273	46	1	15	495
Main-Call:	**36135**	5330	618	669	682	14185

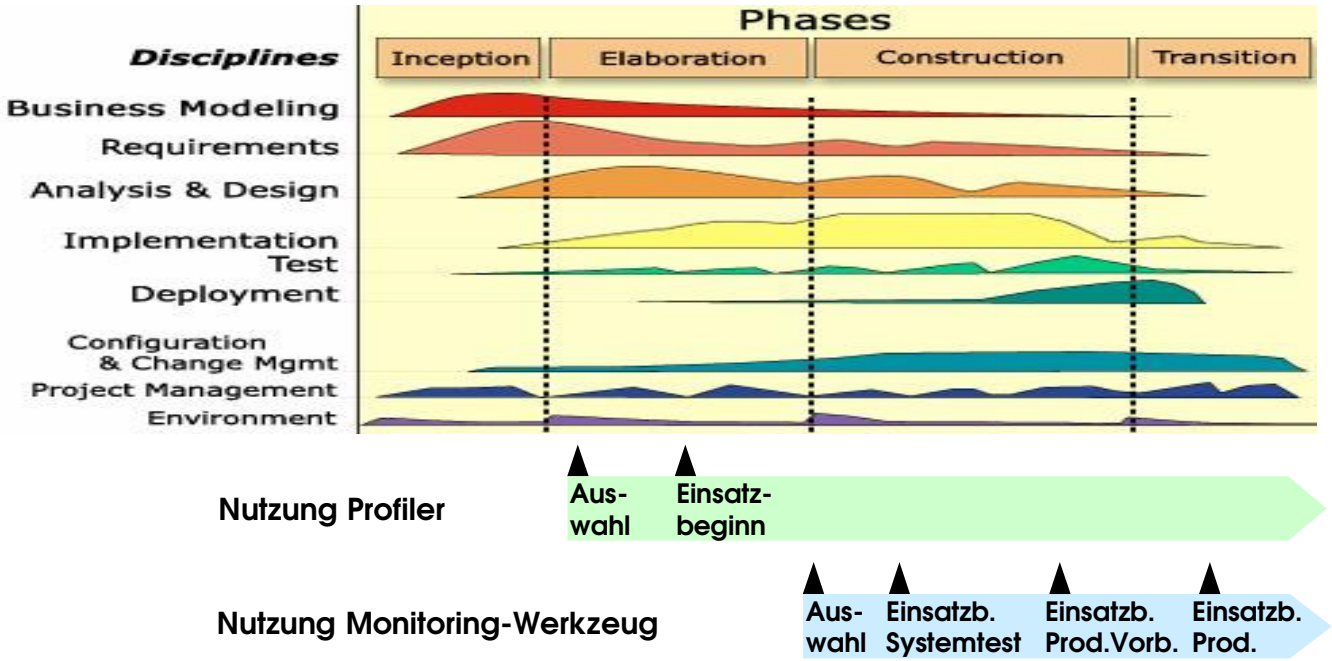
Stacktrace:

- de.bgnet.zw.abs.auftrag.Hauptbuchungsgruppe.erstelleBuchungsgruppen(Hauptbuchungsgruppe.java:1460)
- de.bgnet.zw.afss.fiwiumbuchungenfeststellenanordnen.srv.fiwiumbuchungenfeststellenanordnenvg.FiwiUmbuchungenFeststellenAnordnenVgAgentSkel.erstelleBuch
- de.bgnet.zw.afss.fiwiumbuchungenfeststellenanordnen.srv.fiwiumbuchungenfeststellenanordnenvg.FiwiUmbuchungenFeststellenAnordnenVgAgentSkel.getUmbuchu
- de.bgnet.zw.afss.fiwiumbuchungenfeststellenanordnen.vg.fiwiumbuchungenfeststellenanordnenvg.agent.FiwiUmbuchungenFeststellenAnordnenVgAgentStub.invoke
- de.bgnet.fwk.pkomp.agent.AgentStub.execute(AgentStub.java:176)
- de.bgnet.fwk.pkomp.AfBean.executeAgent(AfBean.java:1652)
- java.lang.reflect.Method.invoke(Native Method)
- com.inprise.ejb.ConcreteMethod.invoke(ConcreteMethod.java:25)
- com.inprise.ejb.EJBContext.invoke(EJBContext.java:129)
- com.inprise.ejb.Dispatcher.doInvoke(Dispatcher.java:1055)
- com.inprise.ejb.Dispatcher.invoke(Dispatcher.java:572)
- com.inprise.ejb.Dispatcher.invoke(Dispatcher.java:548)
- com.inprise.ejb.Dispatcher.invoke(Dispatcher.java:242)

Werkzeugeinsatz im Projekt

JAVAFORUM 2006 stuttgart

- **Am Beispiel des Rational Unified Process (RUP)**



Fazit (1)

JAVAFORUM 2006 stuttgart

- **Monitoring und Profiling ergänzen sich**
  - Es gibt keinen "Goldenen Hammer", der Werkzeug-Mix macht es aus
    - Monitoring "entdeckt" Probleme
    - Mit Profiling geht man ihnen auf den Grund
- **Reproduzierbare und aussagefähige Ergebnisse erfordern End-2-End-Monitoring**
  - Insbesondere bei Enterprise-Umgebungen essenziell
  - Die Addition der Ergebnisse von Tests einzelner Strecken (abschnittsweise Überprüfung) entspricht meist nicht der Gesamtzeit
  - Grund sind z.B. ...
    - Nebenläufigkeitsbedingungen,
    - Race-Conditions,
    - Last,
    - ...

Fazit (2)

JAVAFORUM 2006 stuttgart

- **Der Betrieb einer Enterprise-Anwendung ohne Monitoring ist wie das Autofahren im Regen ohne Scheibenwischer**
  - Fehler müssen vor den Benutzern erkannt werden
  - Einhalten von Service Level Agreements (SLAs) ohne Monitoring ist nicht möglich
  - Anwendung/Umgebung/Datenmenge ändert sich über die Zeit
- **Die Nutzung von Monitoring in der Produktion ist allerdings diffizil**
  - Ein Einstellungsfehler kann die Benutzbarkeit der Anwendung verhindern
  - Daher frühzeitiger Einsatz vom Monitoring im Test, damit Erfahrungen gesammelt werden können.

Abschluss

JAVAFORUM 2006 stuttgart

- **Fragen?**
- **Vielen Dank!**