Management und Monitoring von Java/J2EE Applikationen

Daniel Adelhardt

Consultant

Sun Microsystems GmbH

java.com

Sun microsystems

We make the net work.

Agenda

- Einführung
  - Definition von Management/Monitoring
  - System Level Monitoring von Java/J2EE
- Application Management mit den Java Management Extensions (JMX)
  - Einführung in JMX
  - JMX zum Management von J2EE Servern
- Management “out of the Box” in neuen Java Plattformen
  - J2sE 1.5 Management Technologien
  - JSR-77: J2EE Management API's für J2EE 1.4
- Zusammenfassung

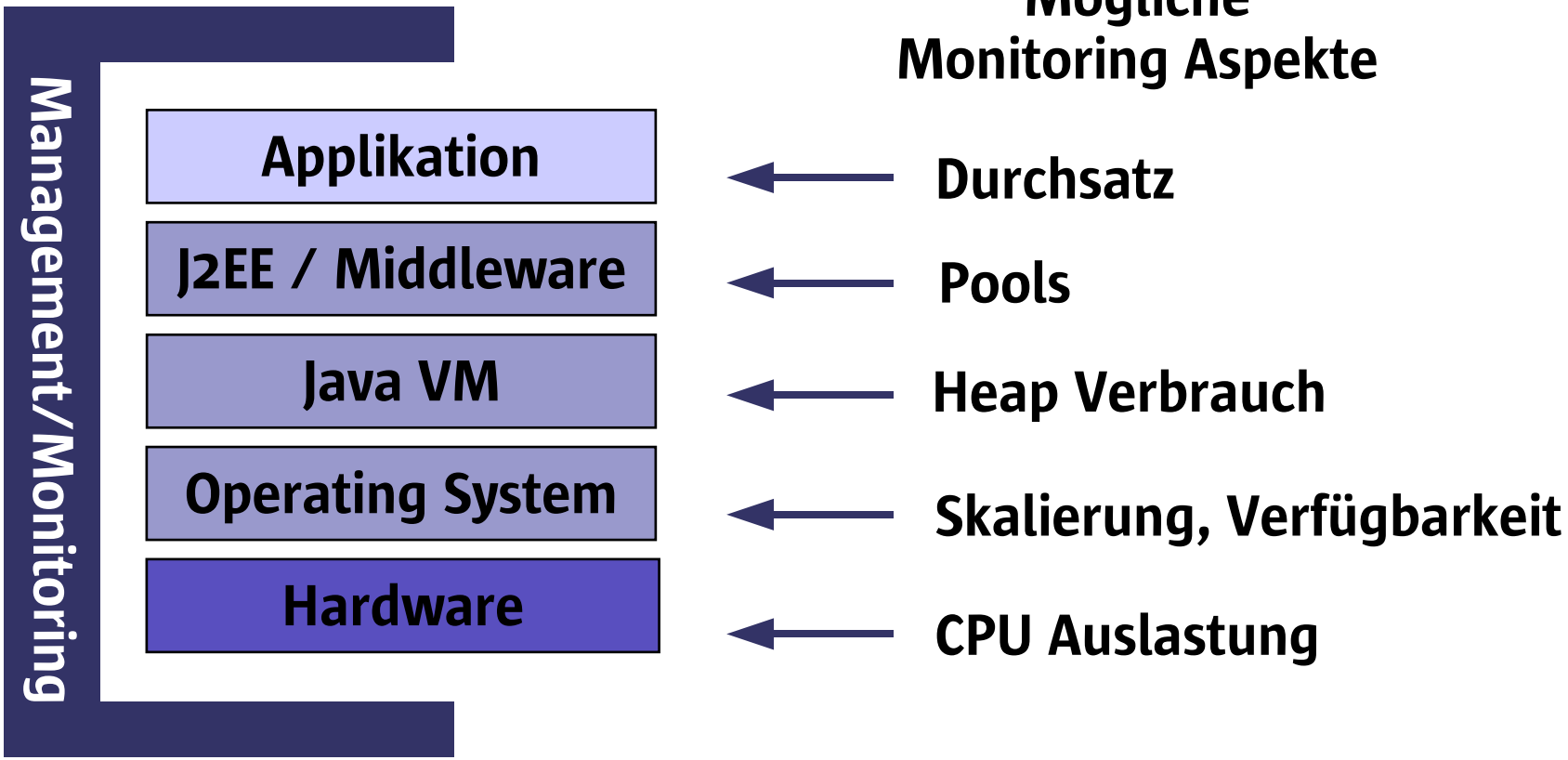
Was ist Monitoring/Management?
(im Kontext dieses Vortrags)

- Überwachen eines Systems bzw. einzelner Ressourcen im laufenden Betrieb z.B. für
  - Status Monitoring
  - Performance Analyse
  - Kapazitätsplanung
  - Diagnose
  - Debugging
  - Ausgangspunkt für Performance Tuning
  - Betriebsführung/Operating der Applikation
- Konfiguration des Systems bzw. von Ressourcen zur Laufzeit
  - z.B. Logging/Tracing Optionen etc.

Probleme des Applikations Monitorings

- Ein Gesamtsystem besteht aus einer Vielzahl von Einzelkomponenten
  - Hardware, Storage, OS, Middleware, Applikation
  - Einheitliches Management über alle Layer ist kaum zu erreichen
- Unterschiedliche Standards für unterschiedliche Aspekte des Managements
  - SNMP, WBEM/CIM,...
- Management Produkte mit jeweils eigenen Protokollen und Schwerpunkten
  - IBM Tivoli, HP OpenView, Sun Solstice EM,...
- Monitoring und Management wird nicht von Entwicklern ausgeübt!

Typische Java/J2EE System Architektur



Monitoring auf System Ebene

- Operating System
  - Virtual Memory Benutzung (Swapping/Paging,...)
  - I/O Statistiken (KB RW/Subsystem)
  - CPU Daten (vol/invol. Context Switches, Usr/Sys load, Lastverteilung auf CPUen in SMP Systemen)
  - Netzwerkinterfaces (Retransmissions, KB/Sec, Listen Drops,...)
- ➔ Wie monitored man das OS?
  - ➔ Solaris/Linux/Unix:

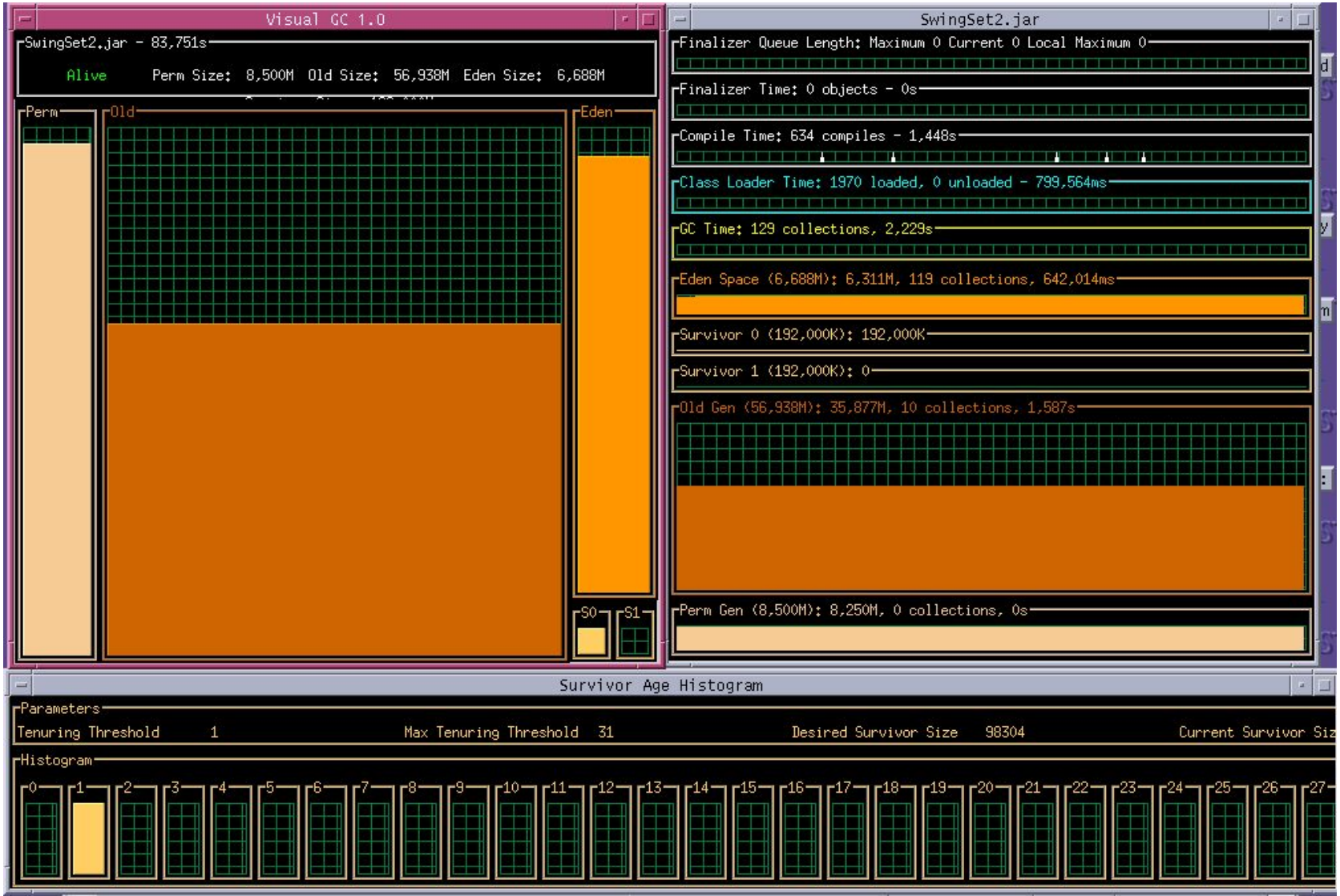
    Virtual Memory mit vmstat
    I/O mit iostat
    CPU Daten mit mpstat
    Netzwerk mit netstat
    Weitere Tools: prstat, top, dimstat, truss, sar, SE-Tools,kstat,...

Monitoring der Java Virtual Machine

- Java Virtual Machine
  - Garbage Collection Verhalten (Min/Max Heap, Young Generation vs. Old Generation, Perm Space, Tenuring Distribution)
  - JIT/Compiler Aktivitäten
  - Anzahl Threads, Thread Contention
- Wie monitored man eine (Sun HotSpot) JVM?
  - Java VM Flags:
    - Garbage Collection: **`-verbose:gc, -XX:+PrintHeapAtGC, -Xloggc:<filename>,`**
    - Compile Vorgänge: **`-XX:+PrintCompilation`**
  - Serviceability Agent
  - Profiling Agent
  - VisualGC (für J2SE 1.4.1/1.4.2)

Neu: VisualGC

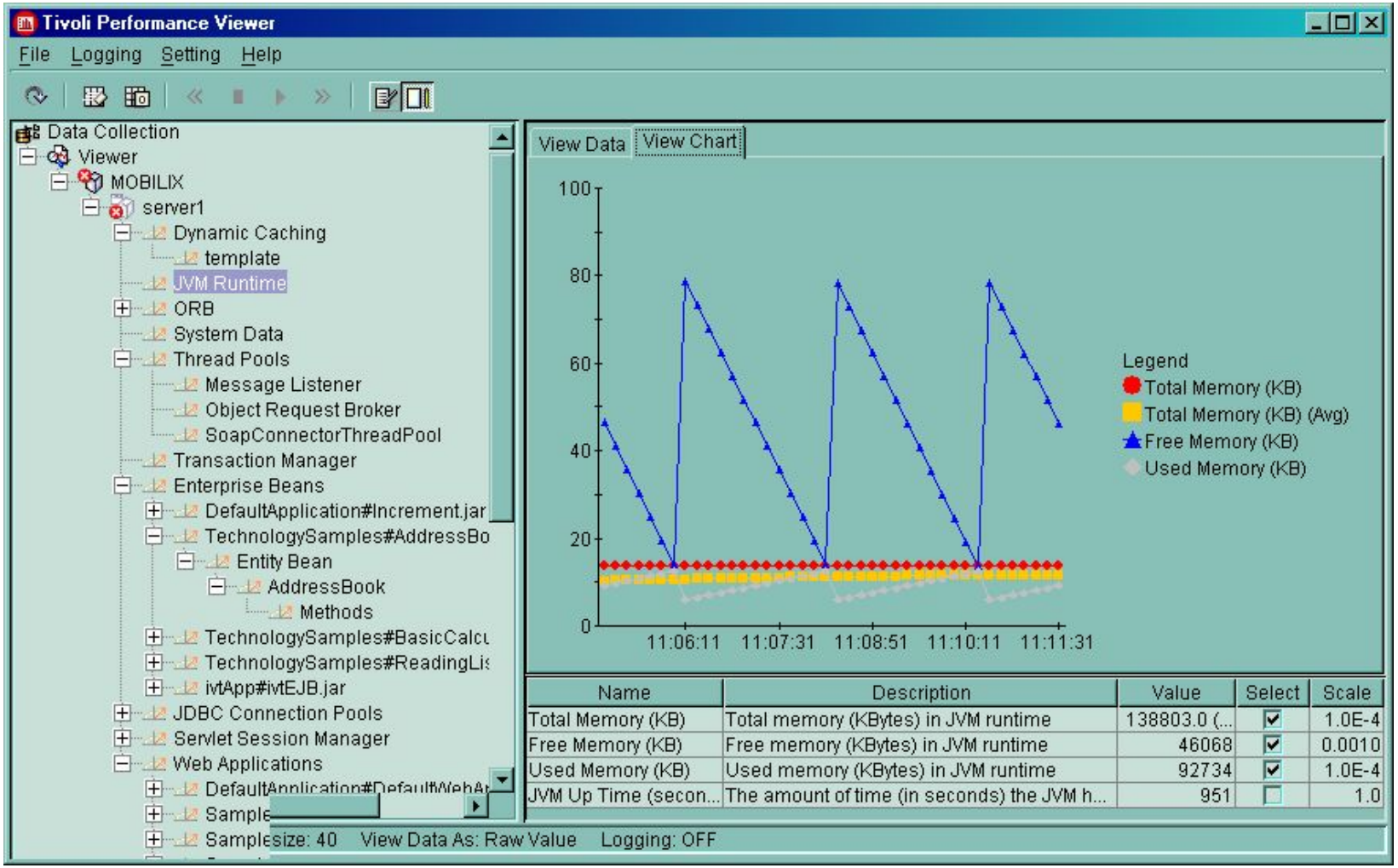
http://developers.sun.com/dev/coolstuff/jvmstat/



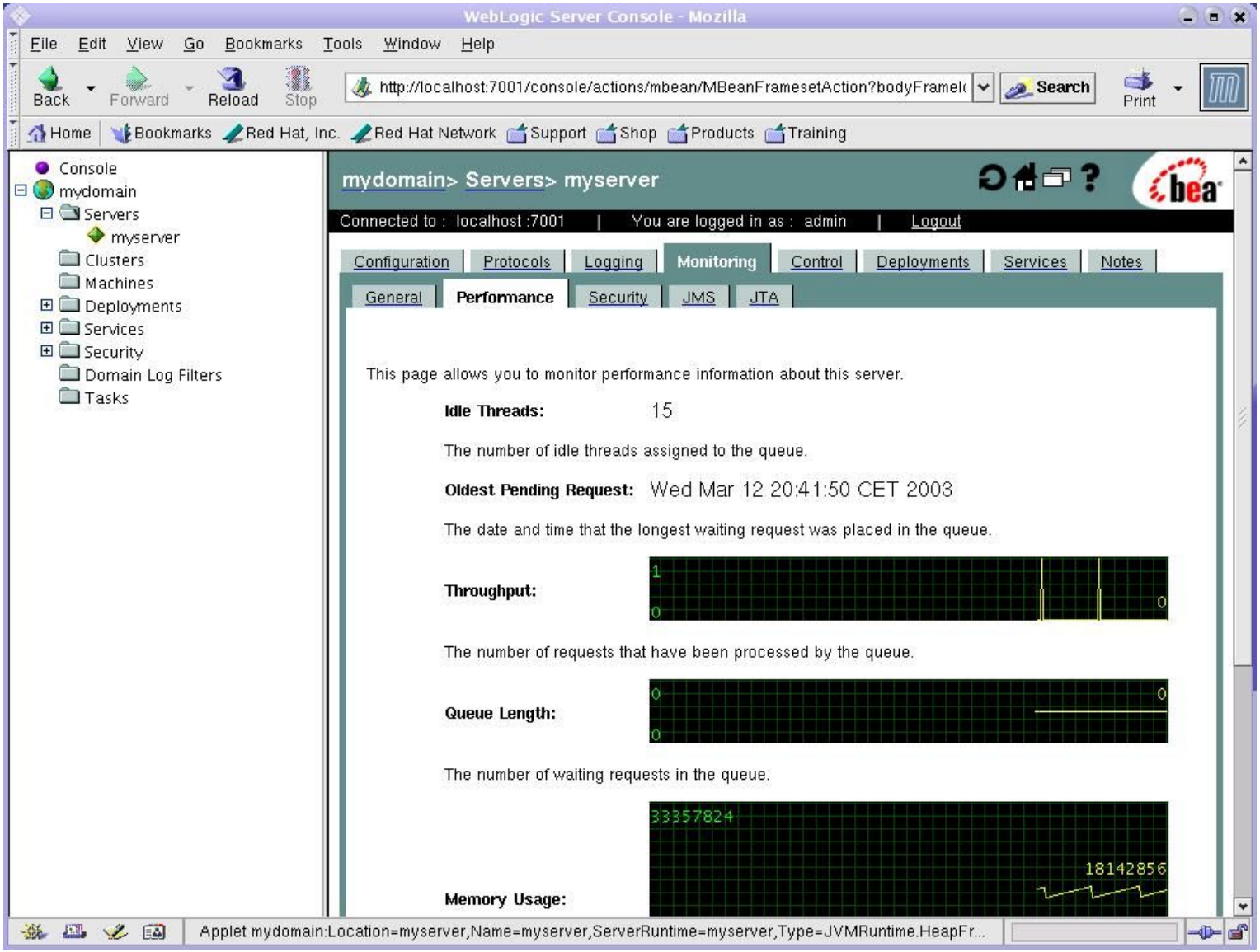
Monitoring eines Application Servers

- Relevante Parameter eines Application Server
  - Thread Pools (Auslastung, Sizing, Shrinking)
  - Connection Pools (Größe, Statement Cache,...)
  - Object Request Broker (Threads, CBV/CBR)
  - EJB Container (Pools/Caches, Passivierungen von SF SB's)
  - ...
- Wie monitored man einen Application Server?
  - Keine Standards, jeder Server bringt eigene Tools mit sich
  - Meist Web basiert, ergänzt durch CLI Tools
  - Manchmal SNMP Adapter, die jedoch meist nicht alle Parameter monitoren lassen

Beispiel WebSphere Resource Analyzer



WebLogic Console



Probleme beim J2EE Monitoring heute

- Derzeit meist kein ganzheitliches Management des Systems möglich
  - Meist wird nur der Application Server und evtl. die JVM überwacht
  - Applikationen/Komponenten als Black Box
    - Häufig bieten die Admin Tools nur Deploy/Undeploy Features
- Wie kann man die eigene Applikation überwachen?
  - Häufig muß derzeit auf 3rd Party Applikationen zurückgegriffen werden
  - Entwickler greifen oft zu selbst entwickelten Management Lösungen
    - Code Instrumentierung mittels System.out.println() oder Tracing Features
    - Meist unflexibel, schwer wartbar und oft statisch

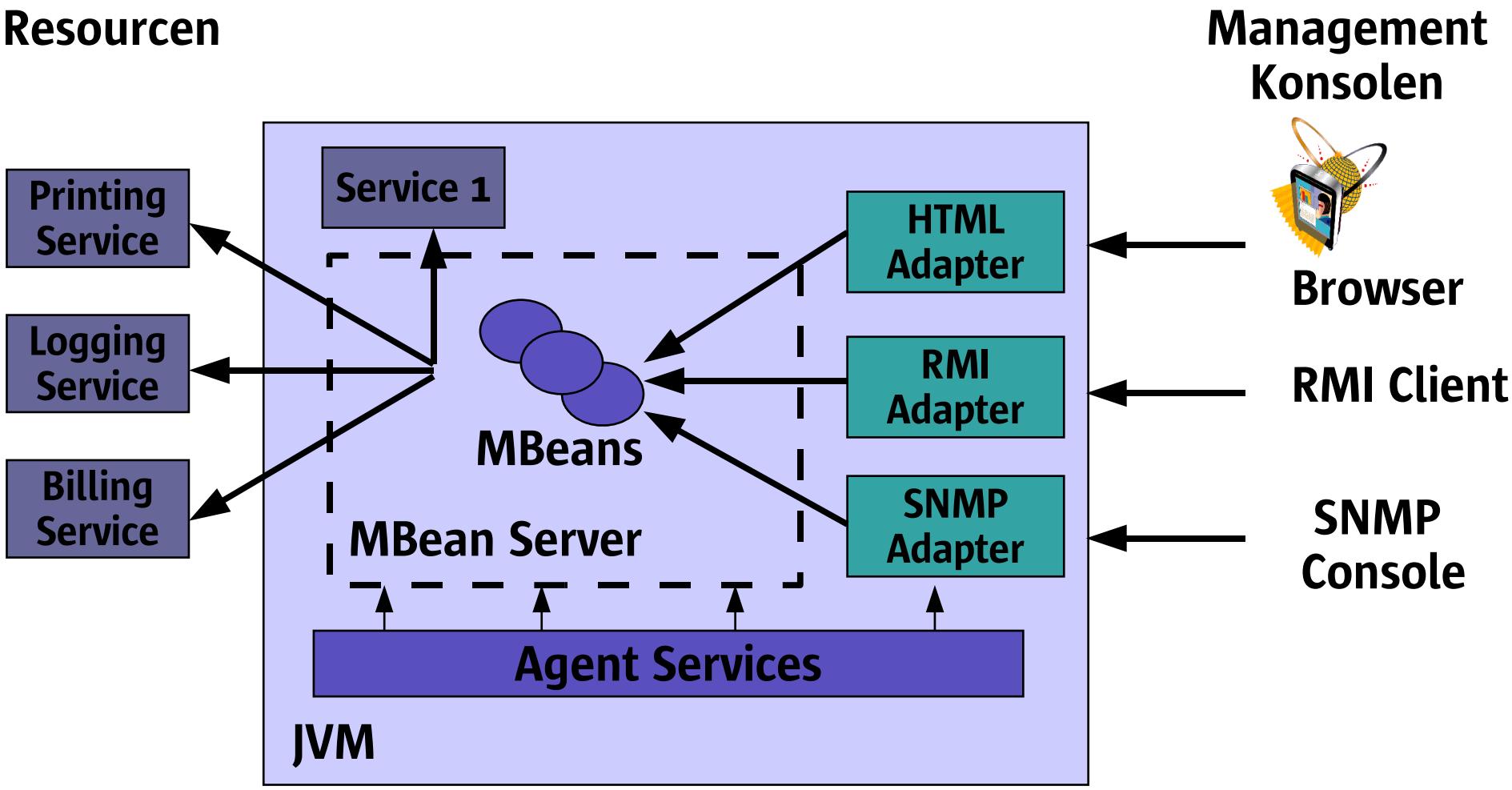
Java Management Extensions (JMX)

- JMX ist ein optionales Package für J2SE/J2EE
  - Enstanden aus dem Produkt Sun Java Dynamic Mangement Kit (JDMK)
- JMX besteht aus einer Architektur, einer API und Services für das Management von Java Applikationen
  - Ermöglicht das Überwachen von Java Applikationen mit Standard Management Konsolen
  - Wird kontinuierlich im Java Community Process weiterentwickelt
  - Bewährte Technologie (seit mehr als 4 Jahren verfügbar)
  - Findet sukzessive Einzug in neue Java Plattformen
  - Kann auch in J2EE Servern benutzt werden

JMX Terminologie

- Instrumentierung
  - Realisierung von managebaren Ressourcen als JMX MBeans
- MBean
  - Designpattern für managebare Java Beans
- MBeans Server
  - In Process Registry für MBeans
- JMX Agent
  - Service Container für MBeans und Server
- Connectors/Adapter
  - Stellen Protokolladapter für Remote Zugriff dar

JMX Architektur

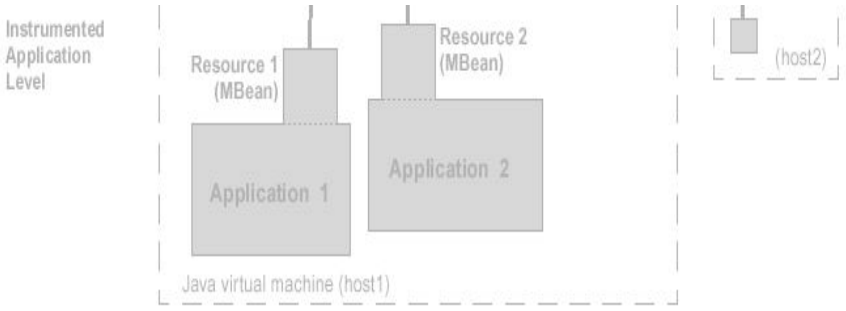


Typische Einsatzgebiete für JMX

- Auslesen/Ändern von Applikationsparametern zur Laufzeit
- Zur Verfügung stellen von Statistiken
  - Performance
  - Ressourcenbenutzung
  - Logs/Probleme
- Zustellung von Nachrichten
  - Fehler
  - Zustandsänderungen
- Automatisches Monitoring kritischer Systemkomponenten

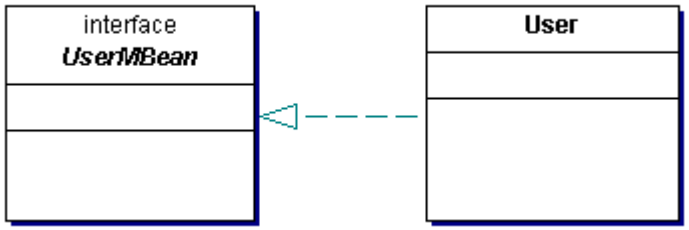
Instrumentation Level

- Alles was als Java Objekt darstellbar ist kann mit JMX instrumentiert werden (HW, SW, Services,...)
- Die Instrumentierung erfolgt über MBeans, von denen JMX verschiedene Varianten vorsieht
  - Standard MBeans
  - Dynamic MBeans
  - Model MBeans
  - Open Mbeans



Standard MBeans

- Einfachster Weg für JMX Instrumentierung
- Management Interface wird vom User explizit spezifiziert
- Agent benutzt Introspection für das Erkennen der Management Interfaces
- Verpflichtend seit JMX 1.0
- Vererbung wird unterstützt
- Nachteil: Statisches Interface



Tutorial: HelloWorld MBean

- JMX spezifiziert Namenskonventionen für Standard MBeans
  - Die managebare Ressource muß ein Interface mit gettern/settern für R/W Attribute bzw. gettern für ReadOnly Attribute spezifizieren
  - Interface und Implementierung müssen im gleichen Package sein
  - Das Interface trägt den Namen **`<Ressource>MBean`**
  - Eine Klasse **`<Ressource>`** muß das Interface implementieren
  - Exponiert werden Attribute, Methoden, Konstruktoren und Notifikationen

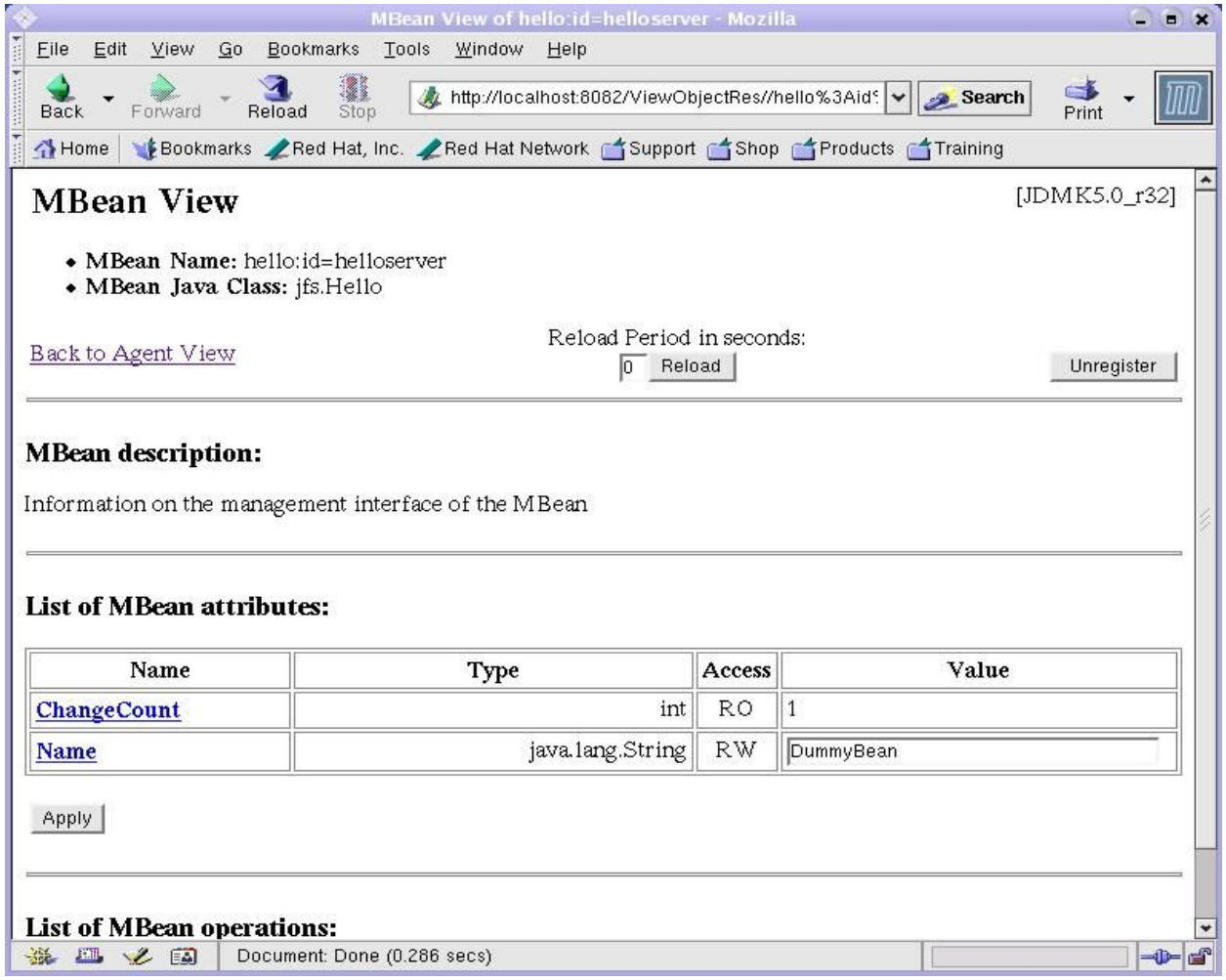
Tutorial: Hello MBean

```
public interface HelloMBean {

    public String getName();
    public void setName(java.lang.String str1);
    public int getChangeCount();
}
public class Hello implements HelloMBean {

    int ctr;
    String name;

    public int getChangeCount() {
        return ctr;
    }
    public String getName() {
        return name;
    }
    public void setName(java.lang.String str1) {
        name = str1;
        ctr++;
    }
}
```

Tutorial: MBean Registrierung

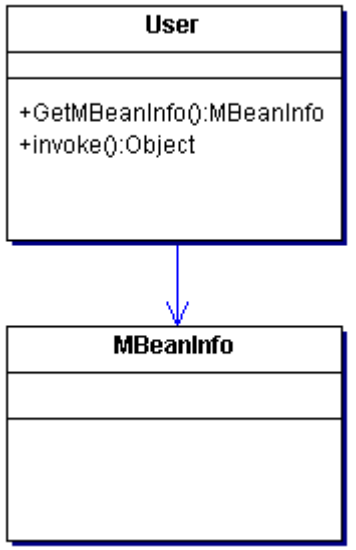
```
//MBeans Server instanziieren
MBeanServer myMBeanServer = MBeanServerFactory.createMBeanServer();

//HelloWorldMBean
Hello h = new Hello();
//HTML Adapter
CommunicatorServer htmlAdaptor = new HtmlAdaptorServer();

try {
      ObjectName on =  new ObjectName("hello:id=JFS2003-HelloServer");

      myMBeanServer.registerMBean(h, on);
      ObjectName htmlAdaptorObjectName = null;
      ObjectInstance htmlAdaptorInstance =
        myMBeanServer.registerMBean(htmlAdaptor, htmlAdaptorObjectName);
      //Start the WebServer
      htmlAdaptor.start();
     }
catch(Exception e) {
      e.printStackTrace();
}
```

JMX Browser


MBean View of hello:id=helloserver - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8082/ViewObjectRes//hello%3Aid%
Search
Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
MBean View
[JDMK5.0_r32]
MBean Name: hello:id=helloserver
MBean Java Class: jfs.Hello
Back to Agent View
Reload Period in seconds:
0
Reload
Unregister
MBean description:
Information on the management interface of the MBean
List of MBean attributes:
Name
Type
Access
Value
ChangeCount
int
RO
1
Name
java.lang.String
RW
DummyBean
Apply
List of MBean operations:
Document: Done (0.286 secs)

Dynamic MBeans

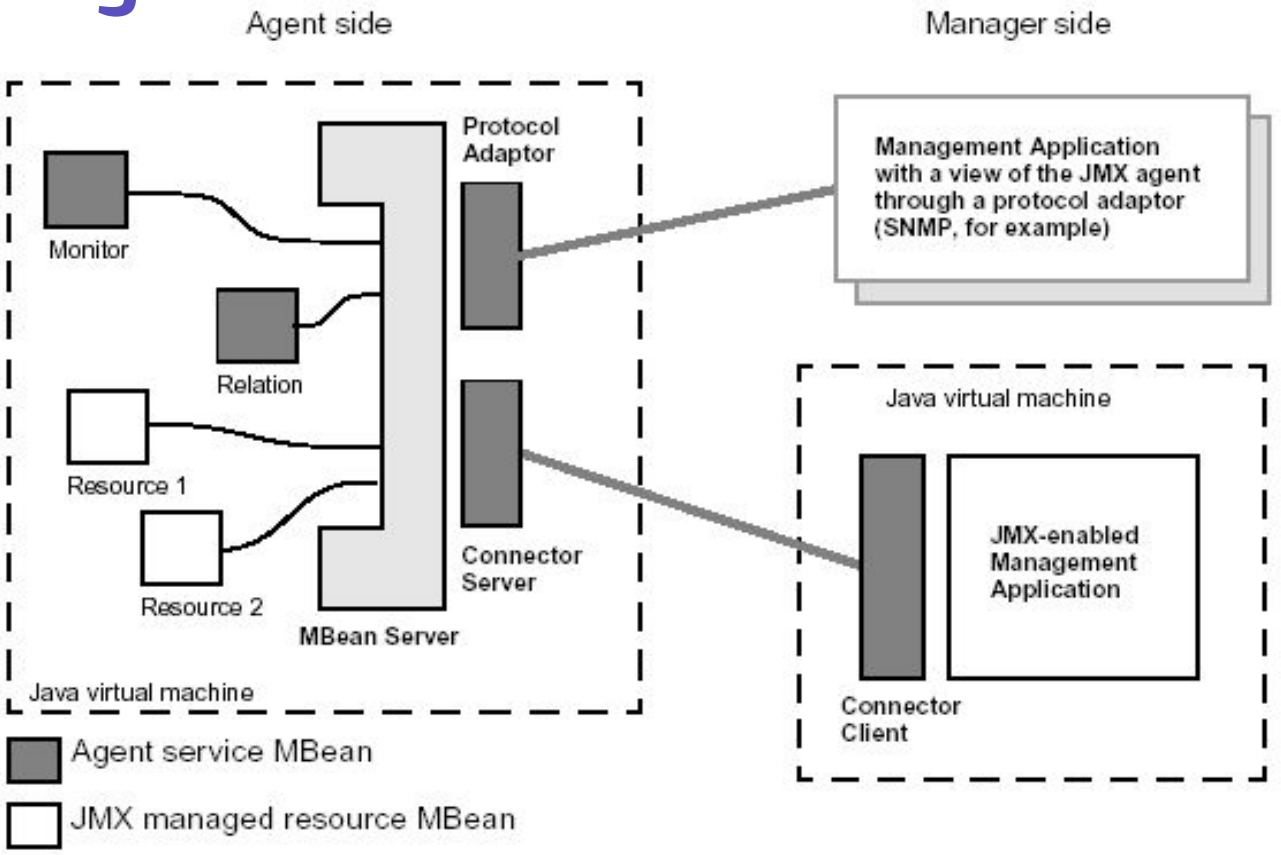
- Entwickler hat mehr Kontrolle
- Management Interfaces werden dynamisch zur Laufzeit erzeugt
  - Interface kann modifiziert werden ohne den Code zu ändern
  - Interface kann zur Laufzeit verschiedene Datenquellen benutzen um sich zu aktualisieren (z.B. XML, DB, ...)
- MBeans implementieren das Interface **`DynamicMBean`**
  - Methoden **`get/setAttribute(s), getMBeanInfo`** und **`invoke`**



Code Sniplet: HelloWorld als Dynamic MBean

```
public MBeanInfo getMBeanInfo() {
   attributes[0] = new MBeanAttributeInfo("Greeting",
                       "java.lang.String",
                       "Hello World Text",
                       true,
                       true,
                       false);

   Constructor[] constructors = this.getClass().getConstructors();
   dConstructors[0] = new MBeanConstructorInfo("HelloDynamic",
                            constructors[0]);


   return new MBeanInfo(this.getClass().getName(),
                  "Test einer Dynamic MBean",
                  attributes,
                  constructors,
                  operations,
                  new MBeanNotificationInfo[0]);

  }
```

JMX Agent Level



- besteht aus: Mbean Server, Standard Agent Services,Connectoren und Adaptoren
- kontrolliert die Ressourcen und stellt sie den Management Anwendungen zur Verfügung
- MBean Server als Registry für MBeans

JMX Services

- Timer Agent Service
  - konfigurierbarer Scheduler
- Monitor Agent Service
  - beobachtet andere Mbeans
  - Counter Monitor, Gauge Monitor, String Monitor
- M-let Agent Service
  - dynamic class loading von Mbeans von remote URLs
- Relation Agent Service
  - definiert Verknüpfungen und Rollen zwischen MBeans für Abfragen und Event-Verteilung

Benutzung von Monitoren

- Mittels Standard Adaptern/Konnektoren werden die Informationen mittels eines Pull Modells abgerufen
- Für die Überwachung von System-Ressourcen können Monitore verwendet werden die andere MBeans überwachen
  - StringMonitor für die Überwachung von String Attributen
  - GaugeMonitor für Überwachung von Grenzen (Float oder Double Werte)
  - CounterMonitor für Zähler
- Überwachende MBeans werden notifiziert

Beispiel für JMX Monitoring

```
public class MyMonitor implements NotificationListener {
...
            CounterMonitor counterMonitor = new CounterMonitor();
            counterMonitor.addNotificationListener(this, null, null);
            try {
                 counterMonitor.addObservedObject(standardObsObjName);
                 counterMonitor.setObservedAttribute("ChangeCount");
                 counterMonitor.setNotify(true);
                 counterMonitor.setInitThreshold(new Integer(4711));
                 counterMonitor.setOffset(new Integer(2));
                 counterMonitor.setGranularityPeriod(1000);
                 } catch (Exception e) { e.printStackTrace(); }
            counterMonitor.start();
 ...
public void handleNotification(Notification notification, Object obj) {
            MonitorNotification notif = (MonitorNotification)notification;
            Monitor monitor = (Monitor)notif.getSource();
            String type = notif.getType();
            if (type.equals(MonitorNotification.THRESHOLD_VALUE_EXCEEDED)) {
                System.out.println(notif.getObservedAttribute() +
                 " has reached the threshold");

            } catch (Exception e) {
                e.printStackTrace();
            }
    }
```

JMX und J2EE?

- JMX wurde für J2SE entwickelt bevor es J2EE gab!
- Es gibt derzeit keine Standard Integration von JMX in J2EE Umgebungen
- Viele J2EE bieten jedoch bereits heute mit J2EE 1.3 eine JMX Implementierung
  - Beispiele: JBoss, BEA, IBM, Sun,...
  - Häufig wird JMX als Backbone der Application Server Implementierung verwendet
    - Web/EJB-Container, Threadpools etc. sind als MBeans instrumentiert um Performancestatistiken zu liefen oder um die Konfiguration vorzunehmen
  - JMX kann jedoch auch für eigene J2EE Komponenten benutzt werden

Typische J2EE Monitoring Aufgaben

- Performance Daten
  - Ermitteln der Aufrufzahlen für Servlets/JSP
  - Antwortzeiten (min,max,avg)
- Transaktionen
  - Anzahl von Commits/Rollbacks pro EJB
  - Durchschnittliche Anzahl von Beans die gelockt sind
- Connection Pools
  - Anzahl von Connections im Pool
  - Avg. Wartezeit auf Pool-Connections
- Überwachung der eigenen Applikation
  - Backend Ressourcen
  - Fachliche Properties

Beispiel: JMX in WebLogic

- WebLogic ist intern mit JMX instrumentiert und bietet verschiedene Möglichkeiten auf JMX zuzugreifen
  - Standard JMX 1.0 APIs
    - Bootstrapping per JNDI erforderlich
    - Auflistung der MBean ObjectNames per
      ' java weblogic.Admin QUERY <mydomain>:* '
  - weblogic.management Package das typsichere Wrapper um JMX bildet
  - RMI und SNMP Adapter/Konnektoren

Beispiel Code mit BEA Management APIs

```
Context ctx = env.getInitialContext();

MBeanHome home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
Iterator applications = home.getMBeansByType("ApplicationRuntime").iterator();

   while (appIterator.hasNext())
    {
     ApplicationRuntimeMBean runtime = (ApplicationRuntimeMBean) appIterator.next();
     ComponentRuntimeMBean[] components = runtime.lookupComponents();
     for (int i=0; i<components.length; i++)
         {
        if (components[i] instanceof WebAppComponentRuntimeMBean)
             {
             WebAppComponentRuntimeMBean webcomp =
                  (WebAppComponentRuntimeMBean)components[i];

             ServletRuntimeMBean[] svlt = webcomp.getServlets();
             for (int j=0; j<svlt.length; j++)
                  {
                  System.out.println("ServletName:" + svlt[j].getServletName()
                  + " was called :" +svlt[j].getInvocationTotalCount() + " times");
                  }
             }
         }
    }
```

Beispiel Code JMX konform

```
MBeanHome home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME); //BEA only !!!
homeServer = home.getMBeanServer();

Set mbeans = homeServer.queryNames(
            new ObjectName("medrec:ApplicationRuntime=MedRecServer_MedRecEAR,
            EJBComponentRuntime=MedRecServer_MedRecEAR_SessionEJB,
            Location=MedRecServer,
            Name=MedRecServer_MedRecEAR_SessionEJB_MailSessionEJB,
            ServerRuntime=MedRecServer,Type=EJBPoolRuntime,*"), query);

        for (Iterator itr = mbeans.iterator(); itr.hasNext(); )
        {
            ObjectName mbean = (ObjectName)itr.next();

            String att[] = {"BeansInUseCount","AccessTotalCount"};
            AttributeList list = homeServer.getAttributes(mbean,att);

            Iterator it = list.iterator();
            while (it.hasNext()){
              Attribute a = (Attribute)it.next();
              System.out.println(a.getName() + " : " + a.getValue());
            }

        }
```

JMX in J2EE Servern

- Die JMX Instrumentierung der gängigen J2EE Server lässt sich für zahlreiche eigene Monitoring Aufgaben verwenden
  - Es können eigene MBeans für die Applikation in die MBeans Server registriert werden
    - z.B. Um dynamisch Traces zu triggern
  - Monitoring/Timer MBeans können zur Überwachung des Servers oder von eigenen kritischen Ressourcen eingesetzt werden
    - z.B. E-Mail Notifikation bei Überschreitung von Schwellwerten
    - Fortlaufendes Monitoring für Performance Degration
    - Monitoring eigener Connection Pools oder Resource Adapter
    - Erhöhen von gepoolten Ressourcen
    - Umschalten von Log Levels zur Laufzeit

Vorteile von JMX

- Verwendung existierender Technologien
  - Adapter für RMI, IIOP, HTTP
- Notifikationsmechanismen
  - Events, Alerts und Listener
- Management muß nicht selbst entwickelt werden
  - Entwickler müssen keine Adapter bzw. Protokolle implementieren
  - Aufwand für MBean Instrumentierung ist überschaubar
  - JMX Adapter/Konnektore erlauben flexible Integration unterschiedlicher Management Konsolen
- Referenz-Implementierung kostenlos downloadbar

Monitoring in der J2SE 1.5 Plattform

- Tiger (J2SE 1.5) enthält mehrere neue Management API's
  - JMX 1.2
  - JSR-163 “Profiling Architecture”
  - JSR-174 “Management and Monitoring Spec for the Java VM”
    - JSR 174 baut teilweise auf der Profiling Architektur auf
- Ziele für JSR 174:
  - Leichtgewichtiges Monitoring
  - Zuschaltbar zur Laufzeit (Event Listener registrierbar)
  - Support für verschiedene Management Consolen
  - Low Memory Detection
  - JMX für Remote Management (Remote MBean) und SNMP
  - Extension Points für Plattform spezifische Traps

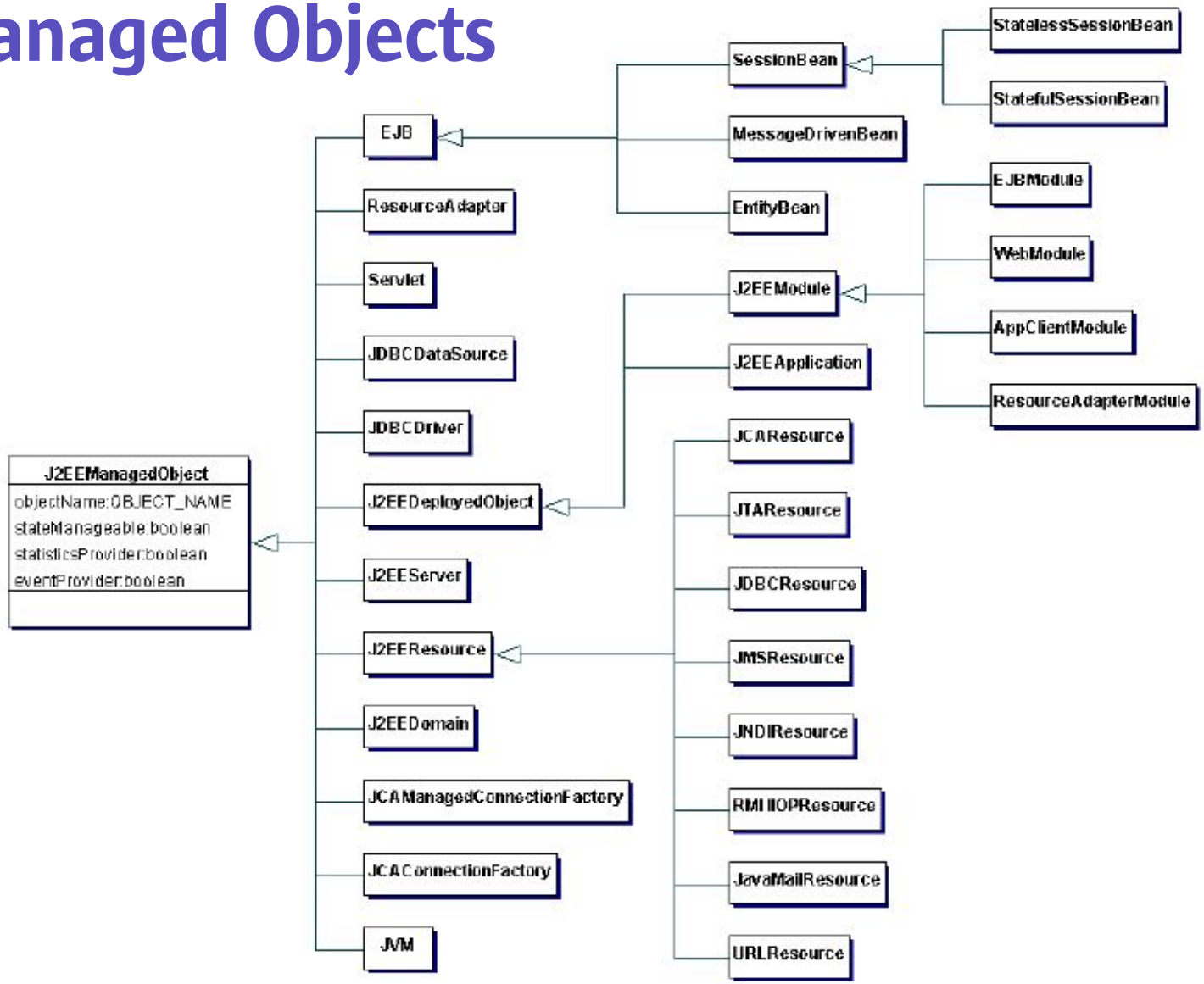
JSR-174 JVM Monitoring/Management
Auswahl einiger möglichen Statistiken

- Verpflichtende Counter
  - JVM Uptime, Thread creates, Thread Informationen (blocked,suspended, Running on JNI, Contentions,...), Class loading
  - Heap Verwaltung (Heap min/max, Alarm Threshold, Time spent in GC,...)
  - Einschalten von verbose:gc, Contention Monitoring
- Optionale Counter
  - Threads waiting on I/O, CPU Zeiten pro Thread
  - Heap alloc. pro Thread, Heap Alarm

J2EE Managament API (JSR-77)

- JSR-77 abstrahiert Management Features von Application Servern
  - Ziel ist es eine App Server unabhängige Schnittstelle für das Management der J2EE Infrastruktur bereitzustellen
  - JMX wird als Basis verwendet, jedoch erweitert um Spezifische Interfaces für J2EE
- Scope von JSR
  - Discovery und Navigation von Managebaren Objekten
  - Event Mechanismen über JMX Notifikationen
  - State Management
  - Performance Counter
  - Optionale Adapter für SNMP und WBEM/CIM

Managed Objects



Benutzung der Management API

- Alle Management Funktionalität wird über eine Stateless Session Bean exponiert
  - Erreichbar über JNDI Namen java:comp/ejb/mgmt/MEJB
  - Über Queries auf dem MBeans Server mit den Objektnamen können beliebige Attribute abgefragt werden
- In J2EE 1.4 müssen alle Application Server diese Management API unterstützen
  - JBoss 4.0 bietet bereits heute eine Implementierung dafür

Beispiel: Bootstrapping des MEJB

```
//Lookup auf die Management EJB
InitialContext ctx = new InitialContext();
Object o = ctx.lookup("ejb/mgmt/MEJB");

ManagementHome home = (ManagementHome)
    PortableRemoteObject.narrow(o, ManagementHome.class);
Management mejb = home.create();

//Default Domain ermitteln
String domain = mejb.getDefaultDomain();

//Query auf alle EJB Module absetzen
Set names = mejb.queryNames(new ObjectName(domain + ":j2eeType=EJBModule,*"), null);
while(itr.hasNext()) {
  ObjectName myname = (ObjectName)itr.next();
  System.out.println("EJB-Module: " + myname);
  ObjectName[] ejbs = (ObjectName[])mejb.getAttribute(myname, "ejbs");
  for(int i = 0; i < ejbs.length; i++) {
    System.out.println("EJB: " + ejbs[i]);
  }
}
```

Zusammenfassung

- Mit JMX steht eine flexible und bewährte Technologie zur Verfügung um Monitoring und Management von Applikationen und Servern vorzunehmen
- Nahezu alle gängigen Application Server arbeiten bereits heute mit JMX
  - Diese Infrastruktur kann auch für eigene Monitoring Aufgaben eingesetzt werden
- Kommenden Java Releases J2SE 1.5 /J2EE 1.4 bieten eine Standardisierung für das Applikationsmonitoring basierend auf JMX
- JMX erlaubt komplettes Monitoring des Java Stacks
  - Selbst tiefere Schichten wie OS und Hardware sind per JMX Wrapper integrierbar

Weitere Informationen

- JMX
  - java.sun.com/products/JavaManagement
- J2EE Management API
  - java.sun.com/j2ee/tools/management
- Java Dynamic Management Kit
  - http://www.sun.com/products-n-solutions/telecom/software/java-dynamic/
- Open Source JMX Implementierung
  - mx4j.sourceforge.net

Daniel Adelhardt

daniel.adelhardt@sun.com

Sun microsystems

We make the net work.