Met de Java Management Extensies (JMX) en de MBeans technologie kun je een willekeurige Java-applicatie, device, service of andere resource zowel lokaal als op afstand monitoren en managen. Met de release van J2SE 5 is JMX aan de Java Standaard Editie toegevoegd. Daarom is het goed JMX nu wat meer in de schijnwerpers te zetten. In dit artikel gaat Willem Koppenol dan ook in op de architectuur, werking en het gebruik van JMX en het ontwikkelen van MBeans.

bespreking

Java Management Extensies

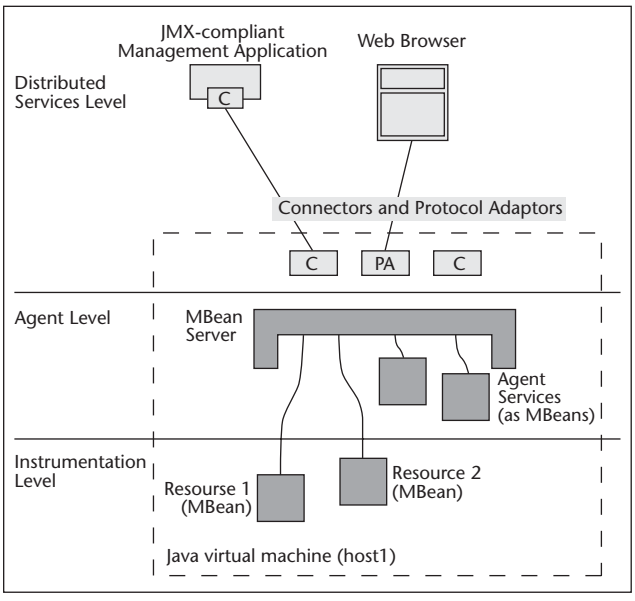
Monitoren en managen met JMX

Veel leveranciers van J2EE-applicatieservers gebruiken de JMX-technologie en in de veelgebruikte Open Source-producten JBoss en Tomcat is JMX een centraal element in de architectuur. JMX bestaat al geruime tijd maar was in het verleden alleen beschikbaar als optionele download. Voor veel ontwikkelaars is JMX mede daardoor een relatief onbekend fenomeen gebleven.

De Java Management Extensies (JMX) bieden een reeks tools en API's voor het monitoren en managen van Java-applicaties tijdens runtime. Bij het monitoren van applicaties kun je denken aan het verzamelen van statistieken over geheugen of CPU-gebruik of het opvangen en loggen van fout indicaties. Bij het managen van applicaties gaat het om zaken als het tijdens runtime veranderen van parameters en configuratiesettings of het uitvoeren van een garbage collection-slag in de applicatie. JMX kan gebruikt worden voor allerlei typen Java-applicaties. Het maakt niet uit of de te monitoren applicatie een eenvoudige desktop-applicatie is of een zware J2EE-applicatie. De JMX-technologie is een Java-standaard en is ontwikkeld via het Java Community Process (JCP). JMX bestaat eigenlijk uit twee Java Specification Requests (JSR's): JSR 3, Java Management Extensions en JSR 160, JMX Remote API. De huidige versie is JMX 1.2.

ARCHITECTUUR JMX JMX volgt de client-server architectuur en biedt een standaard-API. De server is een JMX Agent die wordt gekoppeld aan de applicatie die gemonitored en gemanaged moet worden. De koppeling komt tot stand doordat één of meer MBeans (managed beans), die de applicatie representeren en waarover later meer, zich bij de JMX Agent registreren. De client is een management-applicatie. Deze JMX Client maakt een connectie met de JMX Agent en start het monitoren of managen.

In JMX kun je een drietal lagen onderscheiden. De instrumentatielaag, de agent-laag en de laag van de gedistribueerde services. De instrumentatielaag geeft toegang tot de applicatie door middel van één of meer managed MBeans. De agent-laag geeft JMX Client appli-



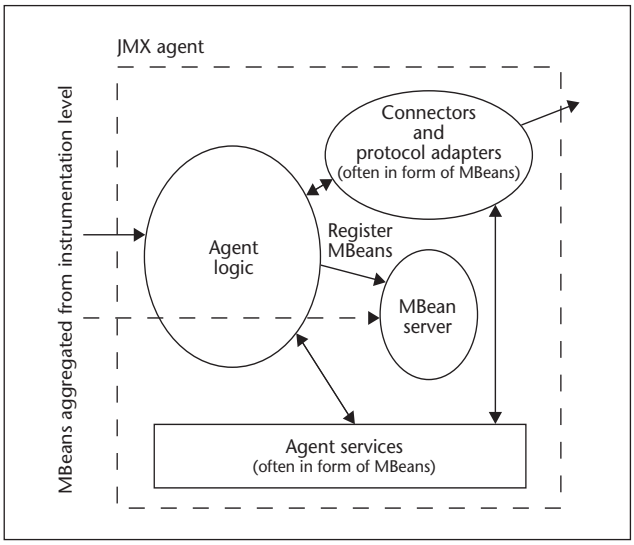
Figuur 1. JMX Architectuur.

caties van afstand toegang tot alle geregistreerde MBeans. Tenslotte is er de laag van de gedistribueerde services. Protocol-adapters en connectors zorgen er daarin voor dat JMX Client applicaties van buitenaf toegang krijgen tot de JMX Agent.

MBeans In de instrumentatielaag vinden we de zogeheten managed beans ofwel MBeans. Een MBean is een fundamenteel concept in JMX. Een MBean is de representant van een applicatie, een service, een device of andere resource die gemanaged moet worden. MBeans zijn Java-objecten en stellen aan de buitenwereld een management-interface ter beschikking dat bestaat uit een reeks attributen en methoden. De attributen volgen het standaard-naamgevingspatroon van setters en getters zoals dat voor gewone Java Beans gebruikelijk is. Iedere applicatie die zich wil laten managen via JMX stelt een MBean ter beschikking. De MBean is de link tussen de te managen applicatie en de rest van het JMX Framework. MBeans kunnen ook events genereren op grond van bijvoorbeeld het bereiken van bepaalde grenswaarden. De JMX specificatie definieert vier typen MBeans: standaard MBeans, dynamische MBeans, model MBeans en open MBeans.

JMX Agent In de agent-laag vinden we de JMX Agent die beschouwd kan worden als de hub van het JMX Framework. Via een JMX Agent kunnen JMX Client applicaties de MBeans aansturen. De voornaamste component van de JMX agent is de MBean server, die fungeert als een server van objecten die gemanaged kunnen worden. MBeans moeten eerst bij de MBean server worden geregistreerd alvorens de managementactiviteiten kunnen starten. Een JMX Agent kent zelf ook een aantal services voor het managen van MBeans zoals het dynamisch laden van MBeans. Minimaal moet de JMX-agent voorzien zijn van één communicatie-adaptor of connector om de toegang te kunnen verlenen aan een JMX Client. JMX-agents bevinden zich vaak op dezelfde machine als de resources die ze controleren maar dit is geen vereiste. De implementatie van een JMX Agent is niet afhankelijk van de resources die de JMX Agent moet gaan managen. Iedere resource die voorzien is van de juiste instrumentatie volgens de JMX Specificatie (MBeans) kan gemanaged worden. Evengoed is de JMX Agent niet afhankelijk van de JMX Client die haar wil gaan gebruiken.

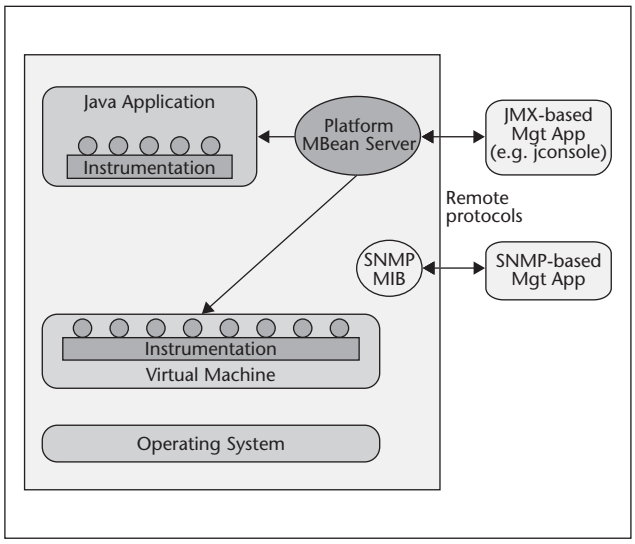
De MBean is de link tussen de te managen applicatie en de rest van het JMX Framework



Figuur 2. Structuur van een JMX agent.

Connectors en Adaptors In de distributed services laag bevinden zich de connectors en adaptors. Connectors zijn nodig om een JMX Agent te kunnen laten communiceren met een JMX Client. De communicatie vereist zowel een component aan de agent-kant als aan de client-kant. De JMX specificatie definieert een connector die is gebaseerd op RMI. Deze connector moet standaard aanwezig zijn in alle JMX-implementaties en is ook aanwezig in J2SE 5.0. De RMI-connector ondersteunt de RMI-transportprotocollen, RMI/JRMP en RMI/IIOP.

Door een connector kun je op afstand een connectie met een MBean in een MBean Server maken en er operaties op uitvoeren net alsof deze operaties lokaal worden uitgevoerd. Een connector is specifiek voor één bepaald protocol, maar een JMX Client merkt daar niets



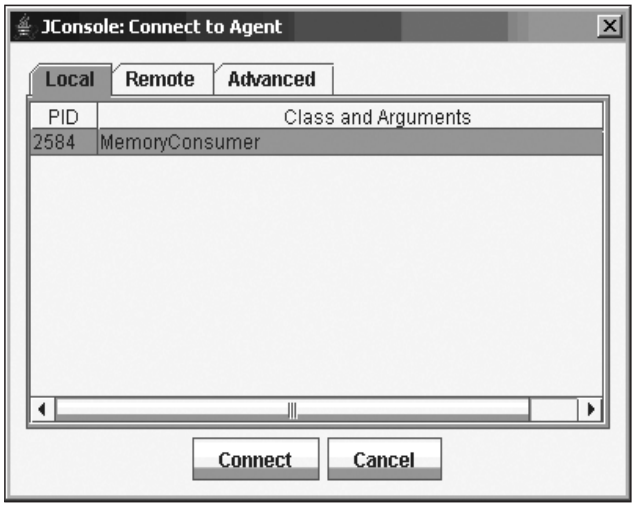
Figuur 3. JMX Architectuur met Platform Beans

van omdat alle connectors hetzelfde remote interface hebben. De JMX Remote API voorziet in een remote interface op de MBean server. Je kunt ook een connector implementeren die is gebaseerd op een protocol dat niet in de JMX Remote API standaard is gedefinieerd zoals een HTTP/S connector.

JMX adapters vertalen het ene protocol in het andere en maakten het mogelijk de MBeans aan te sturen vanuit managementapplicaties die oorspronkelijk gericht zijn op andere protocollen. Populaire adapters zijn SNMP (Simple Network Management) adapters en HTML adapters.

J2SE 5.0 Platform MBeans In de J2SE 5.0 release is de JVM (Java Virtual Machine) ruimschoots voorzien van management en monitor instrumentatie. Bij de J2SE 5.0 worden namelijk een aantal standaard MBeans meegeleverd die een Java applicatie in de JVM kunnen managen en monitoren. Zo zijn er onder meer MBeans aanwezig voor het observeren van threads, het gebruik van geheugen, log-settings van applicaties en het laden van classen. Deze MBeans worden geregistreerd bij de platform MBean Server. Een JMX Client kan dan een connectie maken met de platform MBean Server en de applicatie in de JVM managen. Deze MBeans staan los van de applicatiespecifieke MBeans die mogelijk ook aanwezig zijn.

JMX Clients JMX Clients zijn managementapplicaties die de JMX API gebruiken. Een voorbeeld van een JMX Client applicatie is JConsole, die in J2SE 5.0 standaard wordt meegeleverd en gevonden kan worden in de bin directory van de Java installatie. JConsole maakt gebruik van vele JMX 1.2 mogelijkheden en kan zowel lokaal als op afstand gebruikt worden. Het op afstand monitoren van applicaties verdient daarbij de voorkeur. De JConsole applicatie verbruikt namelijk zelf een aanzienlijke hoeveelheid resources en bij het monitoren van een applicatie op de lokale machine zal de aanwezigheid van JConsole de metingen beïnvloeden.



Figuur 4. Start dialoog venster van JConsole

JMX instellen Eén van de voordelen van JMX is dat het een runtime configuratie-optie is en dat de J2SE 5.0 beschikt over standaard-MBeans. JMX kan gebruikt worden om een willekeurige applicatie tijdens runtime te monitoren. Er hoeft bij het compileren van de code niets te worden toegevoegd of aangeroepen. Als voorbeeld beschouwen we een eenvoudige MemoryConsumer applicatie die in een lus steeds nieuwe objecten aanmaakt. Na verloop van tijd ontstaat er bij het draaien van de applicatie geheugen gebrek en zal de garbage collector de niet gebruikte objecten opruimen:

> JMX-agents bevinden zich vaak op dezelfde machine als de resources die ze controleren

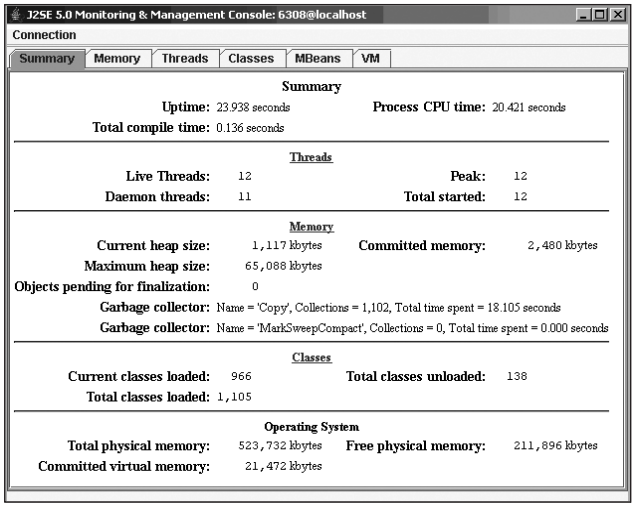
```
public class MemoryConsumer {
    char[] ca = new char[10000];
    public static void main(String[] args) {
    while(true) {
        MemoryConsumer m = new
MemoryConsumer();
    }
   }
}
```

We zullen het gedrag van deze applicatie observeren met JConsole die daarbij gebruik maakt van de standaard aanwezige J2SE 5.0 Platform MBeans. Om deze applicatie met JMX te kunnen managen en monitoren moeten we de JMX Agent voor deze applicatie activeren. Dit kan door bij het opstarten de volgende Java systeem eigenschap mee te geven:

```
java -Dcom.sun.management.jmxremote
MemoryConsumer
```

Een applicatie wordt geconfigureerd voor managen en monitoren op afstand door de volgende systeem eigenschappen bij de start van de applicatie mee te geven:

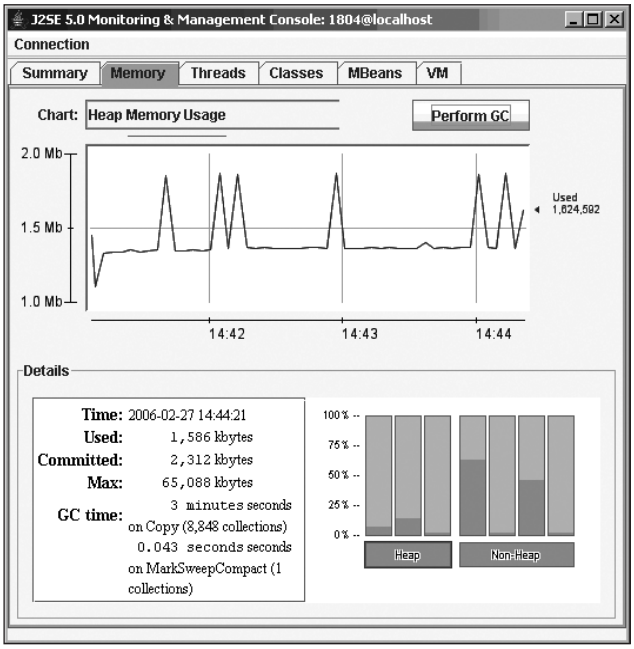
```
java -Dcom.sun.management.jmxremote.port=9999
-Dcom.sun.management.jmxremote.
authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
MemoryConsumer
```



Figuur 5. Overzichtsscherm JConsole.

Een opmerking voor Windows-gebruikers is dat JMX om veiligheidsredenen niet goed zal werken als het systeem opstart vanaf een FAT32-partitie. Zowel voor lokale monitoring als monitoring op afstand is een NTFS-partitie nodig.

JCONSOLE De standaard JMX Client kunnen we starten door in een command prompt het commando JConsole te geven. JConsole zal dan opstarten met een dialoogvenster waarin alle op dat moment lokaal actieve Java-applicaties staan waarvoor een JMX Agent is geactiveerd en die dus gemonitord kunnen worden. Voor het op afstand monitoren van een applicatie geven we parameters zoals hostnaam, port en eventuele inloggegevens op. Het monitoren start door een applicatie te selecteren en op 'Connect' te klikken. JConsole zal vervolgens contact leggen met de JMX Agent van de applicatie en als eerste een scherm tonen met overzicht gegevens van de performance van de applicatie in termen van thread gebruik, CPU tijd, geheugen gebruik en het aantal geladen classes. Deze gegevens worden verzameld door de J2SE Platform MBeans en door JConsole in de schermen verwerkt.



Figuur 6. Geheugen gebruik van de MemoryConsumer applicatie.

Het geheugengebruik van de MemoryConsumer-applicatie is zien in de Memory tab van JConsole. Dit geheugengebruik fluctueert door de voortdurende aanmaak van nieuwe objecten afgewisseld met de opruimacties van de garbage collector. De standaard Memory MBean heeft verder een gc() methode die beschikbaar is onder de knop 'Perform GC'. Hiermee kun je handmatig om garbage collection verzoeken in de applicatie die wordt gemonitored.

Het gebruik maken van de standaard JMX MBeans in J2SE 5.0 kost dus niets extra en is een kwestie van opstartparameters. Het is echter ook mogelijk eigen MBeans te maken voor een applicatie en ook deze optie zullen we beschouwen.

STANDAARD MBEANS Iedere applicatie kunnen we voorzien van een MBean. Een standaard MBean bestaat uit een Java-interface en een Java-class die deze interface implementeert. De conventie is dat de naam van een MBean interface bestaat uit de naam van de Java-class met daarachter de suffix MBean. Als de naam van de Java class Car.java is, wordt de naam van de interface dus CarMBean.java. De methodes in de interface volgen standaard-naamgevingspatronen en definieren ofwel een attribuut ofwel een methode in de MBean. Een voorbeeld van een MBean interface is:

```
package novasoft.mbeans;
public interface CarMBean {
    public void accelerate();   //operatie
    public int getSpeed();      // getters
    en setters
    public void setSpeed(int v);// voor een
attribuut
}
```

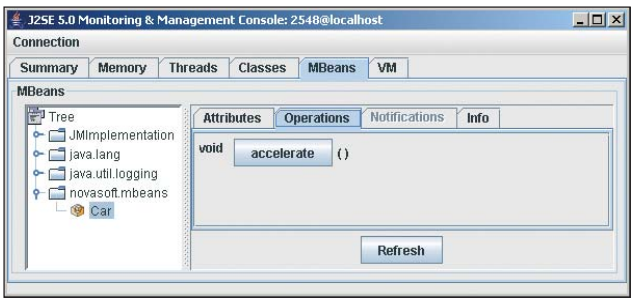
De corresponderende MBean is :

```
package novasoft.mbeans;
public class Car implements CarMBean {
    private int speed = DEFAULT_SPEED;
    private static final int DEFAULT_SPEED =
    50;
  public void accelerate() {
     speed=speed+1;
  }
  public synchronized int getSpeed() {
```

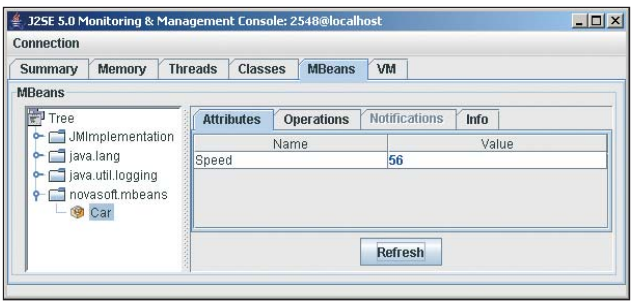
```
    return this.speed;
  }
  public synchronized void setSpeed(int v) {
    this.speed =v;
  }
}
```

Registratie van MBeans Om deze Car MBean te kunnen managen en monitoren moet de MBean eerst worden aangemaakt in een applicatie en geregistreerd worden bij de MBean-server. De code van de applicatie die dat realiseert is:

```
package novasoft.mbeans;
import java.lang.management.*;
import javax.management.*;
public class Main {
    public static void main(String[] args)
throws Exception {
  MBeanServer mbs = ManagementFactory.
  getPlatformMBeanServer();
  // Construeer de ObjectName voor de te
registreren MBean
  ObjectName name = new ObjectName("novasoft.
mbeans:type=Car");
  Car mbean = new Car();           // Maak de
  Car MBean
  mbs.registerMBean(mbean, name); //
Registreer de Car MBean
  System.out.println("Waiting forever...");
  Thread.sleep(Long.MAX_VALUE);
    }
}
```



Figuur 7. Accelerate() methode van de CarMBean



Figuur 8. Attributen van de CarMBean.

Na het (JMX enabled) opstarten van de applicatie wordt de MBean geregistreerd onder een bepaalde naam en type. Als we de applicatie selecteren bij het opstarten van JConsole komt de MBean vervolgens tevoorschijn in de schermen van JConsole. JConsole laat het attribuut Speed zien met een initiële waarde van 50. JConsole laat ook de methode accelerate() zien en na herhaald klikken op accelerate() zien we dat het attribuut Speed is toegenomen. We hebben de applicatie van buiten af aangestuurd en een attribuut van waarde veranderd.

Dynamic MBeans Naast de standaard MBeans zijn er nog andere types. Bij een dynamic MBean verneemt de JMX agent tijdens runtime uit metadata welke attributen en methoden de MBean ter beschikking heeft. Een dynamic MBean kent geen standaard setters en getters voor het benaderen van attributen.

Connectors zijn nodig om een JMX-agent te kunnen laten communiceren met een JMX-client

Alle attributen worden gelezen met de generieke `getAttribute()` functie waaraan een String die het attribuut identificeert wordt meegegeven. Analoog is er ook een generieke `setAttribute()` functie. Methodes op dynamic MBeans worden aangeroepen via de generieke invoke() functie waaraan de functienaam en parameters als Strings worden meegegeven. De JMX agent hoeft de MBean niet door middel van introspection tijdens runtime te onderzoeken om de metadata te verkrijgen. De metadata kan worden verkregen door een aanroep van `getMBeanInfo()`. Een voordeel van het gebruik van dynamic MBeans is dat ze gemakkelijker kunnen omgaan met veranderingen.

ModelMBean Een speciale dynamic MBean is de RequiredModelMBean. Zoals de naam al aangeeft is dit een generieke en configureerbare MBean die verplicht in alle JMX-implementaties aanwezig is. In plaats van met eigen MBeans te werken kun je ook de RequiredModelMBean gebruiken. Java Resources die gemanaged willen worden instantiëren de RequiredModelMBean met de `createMBean()` methode van de MBeanServer. De resource zet vervolgens MBeanInfo and Descriptors voor de RequiredModelMBean instantie. Via de Descriptors kunnen waarden en methoden in de te managen applicatie worden gedefinieerd en gemapt op attributen en operaties van de RequiredModelMBean. Deze mapping

kan je dynamisch en programmatisch tijdens runtime definiëren of lezen uit een XML-bestand. Iedere RequiredModelMBean die geïnstantieerd wordt in de MBeanServer kan gemanaged worden. Attributen en operaties zijn van afstand toegankelijk via de connectors of adaptors die een connectie hebben met de MBean-server.

MX Services JMX biedt meer dan alleen een framework voor management op afstand. JMX kent ook een aantal voorgedefinieerde services die een belangrijke rol kunnen spelen in de ontwikkelactiviteiten. Een aantal van deze additionele services, die overigens zelf als MBeans zijn geïmplementeerd, kunnen als volgt omschreven worden:

- *De Monitor service* observeert een attribuut van een andere MBean en stuurt een bericht aan geregistreerde luisteraars als het attribuut een bepaalde grenswaarde overschrijdt. Mogelijke luisteraars zijn andere MBeans of JMX Client applicaties.
- *De Timer-service* stuurt een bericht aan geregistreerde luisteraars op een specifiek tijdstip of na het verstrijken van een bepaald interval.
- *De M-let service* kan MBeans dynamisch instantiëren en bij de MBean-server registreren. De MBeans waarvoor dit mogelijk is, zijn gedefinieerd in een m-let XML-bestand door middel van MLET tags.

Slotwoord Het belang van JMX moet niet onderschat worden. JMX is met name door de MBeans-technologie veel meer dan een passief monitoring-systeem voor Java-applicaties. Door het gebruik van MBeans kun je Java-applicaties, groot en klein, aansturen en er berichten van ontvangen. We krijgen ook steeds meer te maken met gedistribueerde applicaties die op afstand gemanaged moeten worden. JMX biedt daarvoor een uitstekend model. JMX is een standaard en levert een herbruikbaar framework. JMX wordt ondersteund door vele spelers in de markt, kan worden uitgebreid en aangepast met adaptors en connectors. Het is niet onaannemelijk dat JMX steeds meer een kern wordt van het ontwikkelproces en dat iedere Java-applicatie via MBeans benaderd kan worden.

drs. Willem Koppenol is Senior Trainer en Product Specialist Software Development bij Twice IT Training.