De Java Persistence API (JPA) specificatie is definitief en steeds meer frameworks en tools bieden ondersteuning voor deze nieuwe standaard. Dit is belangrijk omdat JPA maar één aspect van applicaties beschrijft: het persisteren van Java objecten in de database. In dit artikel wordt de ondersteuning die Spring 2.0 biedt voor JPA beschreven.

Spring 2.0 ondersteuning

Gebruik met de Java Persistence API

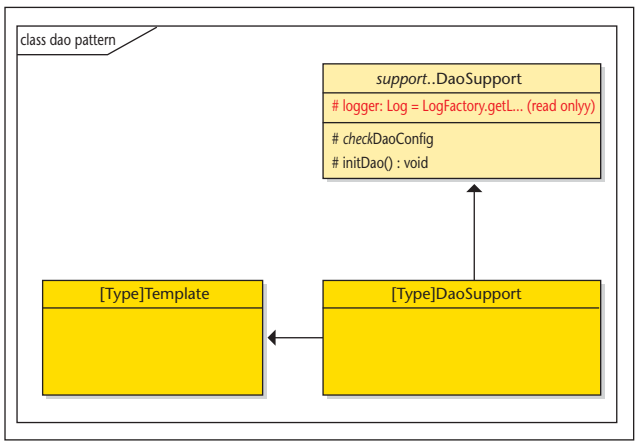
Het gebruik van Spring in een applicatie biedt een aantal voordelen. Het is een lichtgewicht framework dat de infrastructurele zaken zoals resource- en transactiebeheer afhandelt. Hierdoor kan de ontwikkelaar focussen op de business eisen van de applicatie. Spring gebruikt concepten als Dependency Injection en Aspect Oriented Programming waardoor het minimale impact heeft op de Java code die geschreven wordt.

De Java Persistence API is een onderdeel van de EJB 3.0 specificatie (JSR-220). Het is een standaard voor het persisteren van Java objecten in een Java Enterprise Edition (JEE) omgeving, maar ook in een standalone (Java Standard Edition, JSE) omgeving. Spring bundelt standaard de TopLink essentials, maar er kan ook gekozen worden voor andere implementaties zoals Hibernate. Er zijn twee manieren om de Java Persistence API met Spring te gebruiken. De eerste manier is door middel van het voor Spring-ontwikkelaars bekende DaoSupport-patroon. Een alternatief is zonder Spring-specifieke klassen, maar met annotaties en Aspect Oriented Programming (AOP). Beide manieren worden besproken, evenals algemene configuratiezaken zoals resource- en transactiebeheer en het gebruik van Entity Managers.

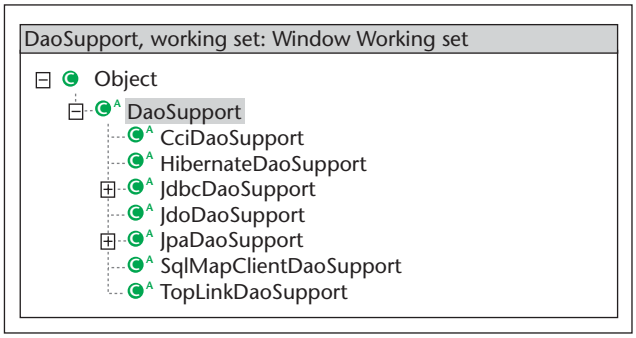
DAO PATTERN EN SPRING Spring maakt gebruik van het DAO-patroon voor het inkapselen en abstraheren van de toegang tot een datasource. (Zie ook http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html). De DAO is verantwoordelijk voor het verkrijgen en vastleggen van de domeinobjecten en het managen van de connectie. Figuur 1 beschrijft de verschillende klassen die betrokken zijn bij de implementatie van dit patroon in Spring.

DaoSupport is een abstracte klasse die verantwoordelijk is voor het initialiseren van de concrete Dao. In Figuur 2 staat een opsomming van de DaoSupport-klassen zoals die nu in Spring 2.0 aanwezig zijn.

Voor iedere DaoSupport klasse is een bijbehorende 'Template'-klasse gedefinieerd (zie Figuur 1). Deze Template klasse bevat de methoden om gegevens op te halen en te bewaren. Figuur 3 beschrijft de implementatie van dit patroon voor de Java Persistence API. Deze



FIGUUR 1. Overzicht van Dao patroon in Spring.



FIGUUR 2. Overzicht van concrete DaoSupport klassen in Spring 2.0.

klassen zijn terug te vinden in de packages org.springframework.dao.support en org.springframework.orm.jpa.

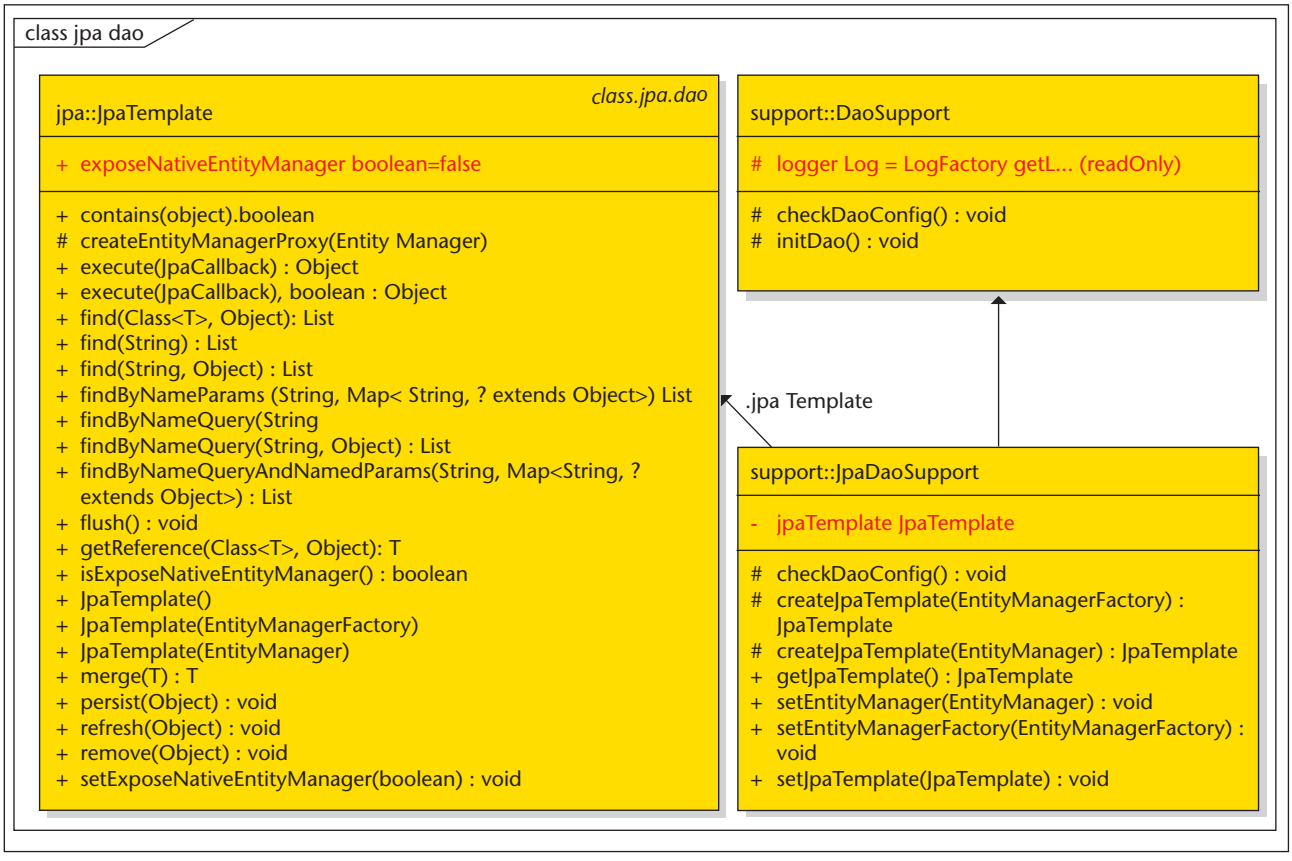
Er is een aantal methoden gedefinieerd in JpaDaoTemplate voor het ophalen en vastleggen van gegevens. De belangrijkste methoden zullen hieronder besproken worden aan de hand van een voorbeeld. Het voorbeeld is een deel van een simpele applicatie waarbij clubs en teams bewerkt kunnen worden (zie Figuur 4).

EXECUTE De eerste methode die opvalt in Figuur 3 is de execute methode. Deze callback-methode biedt de mogelijkheid om de EntityManager rechtstreeks te gebruiken. Hierna volgt een voorbeeld daarvan:

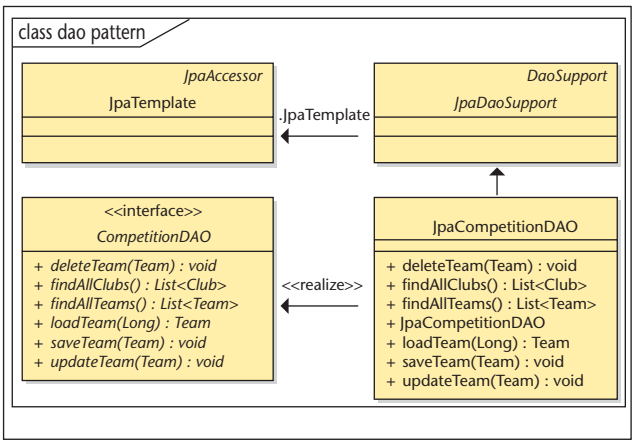
```
/**
* Finds all teams
* @see CompetitionDao#findAllTeams
*/
 public List<Team> findAllTeams() {
   return (List)getJpaTemplate().execute(new
JpaCallback(){

    public Object doInJpa(final EntityManager
em) throws PersistenceException {
  Query queryObject =
em.createNamedQuery(TEAM_FIND_ALL);
  return queryObject.getResultList();
  }

   }, false);
 }
```

Default wordt een dynamische Proxy gecreëerd die de methoden van de onderliggende provider (TopLink, Hibernate, KDO) 'wrapped'. Deze proxy negeert de close-methode omdat Spring de resources beheert. Als het gebruik van een dynamische proxy niet acceptabel is, dan moet in de callback 'true' meegegeven worden.

FIND Behalve de callback is er ook een aantal methoden gedefinieerd voor veel voorkomende operaties. Eén van die operaties is "find".



FIGUUR 3. Traditionele Spring Dao met Template.



FIGUUR 4. Voorbeeld van het gebruik van JpaTemplate en JpaDaoSupport.

Hieronder staat een voorbeeld hoe deze gebruikt kan worden.

```
/**
 *Loads the team with the given id
 *@param teamId id of the team that is loaded
 *@return Team the team with the id or
<code>null</code> if none is found
 * @see CompetitionDao#loadTeam
 */
public Team loadTeam(Long teamId) {
  return getJpaTemplate().find(Team.class,
teamId);
}
```

Er zijn meerdere varianten voor het ophalen van gegevens. In plaats van een id, kan een query of een query met parameters gebruikt worden.

MERGE, PERSIST, FLUSH, REMOVE De 'merge', 'persist', en 'remove'-methoden roepen de gelijknamige methoden aan op de EntityManager. Onderstaande voorbeelden laten het gebruik hiervan zien.

```
/**
 * @see CompetitionDao#updateTeam
 */
 public void updateTeam(Team team) {
    getJpaTemplate().merge(team);
 }

/**
 * @see CompetitionDao#saveTeam
 */
public void saveTeam(Team team) {
  //we need to make sure that the club is
managed by the context
  // This means that we need to call
getReference on it.
  Long id = team.getClub().getId();
  if(id != null){
     team.setClub(getJpaTemplate().
getReference(Club.class, id));
  }

   getJpaTemplate().persist(team);
}

/**
 * @see CompetitionDao#deleteTeam
 */
 public void deleteTeam(Team team) {
     getJpaTemplate().remove(team);
  }
```

VOORDELEN GEBRUIK VAN DAOSUPPORT We hebben nu gezien hoe we de JpaDaoTemplate en JpaDaoSupport kunnen gebruiken in onze applicatie. Deze aanpak heeft verschillende voordelen:

- Foutafhandeling wordt geregeld door Spring. Fouten uit de database worden vertaald naar een uniforme set, onafhankelijk van de onderlinge technologie (JDBC, TopLink, Hibernate, JPA etc).
- De stijl is analoog aan de TopLinkTemplate en HibernateTemplate, met standaardmethoden voor standaardoperaties als find en persist.

Er is een aantal methoden gedefinieerd in JpaDaoTemplate voor het ophalen en vastleggen van gegevens

- Automatische deelname aan transacties.
- Gebruik van resources die gedefinieerd zijn in de Spring container.

Het nadeel van de JpaDaoSupport is dat er een extra laag boven op de bestaande JPA geïntroduceerd wordt. Daarom zal ik in het hierna volgende een alternatief behandelen.

JPA ZONDER SPRING KLASSEN Er is ook een manier om gebruik te maken van Spring, zonder dat specifieke klassen uit het framework in de Java-code opduiken. Hieronder volgt de JpaCompetitionDao zonder dat gebruik gemaakt wordt van de JpaDaoSupport en JpaTemplate:

```
public class JpaCompetitionDao implements
CompetitionDao {

  private EntityManager em;

   /** named query to find all teams.*/
    private static final String TEAM_FIND_ALL
= "Team.findAll";

    /** name of the query to find all
clubs.*/
    private static final String CLUB_FIND_ALL
= "Club.findAll";

  @PersistenceContext
  public void setEntityManager(EntityManager
entityManager) {
        this. em = entityManager;
    }


  /**
   * @see CompetitionDao#findAllClubs()
   */
  public List<Club> findAllClubs() {
    Query query = em.createQuery(CLUB_FIND_
ALL);
    return query.getResultList();

  }

  /**
   * @see CompetitionDao#findAllTeams()
   */
  public List<Team> findAllTeams() {
    Query query = em.createQuery(TEAM_FIND_
ALL);
    return query.getResultList();

  }

  /**
   * @see CompetitionDao#loadTeam(Long)
   */
  public Team loadTeam(Long teamId) {
    return em.find(Team.class, teamId);
  }

  /**
   * @see CompetitionDao#saveTeam(Team)
   */
  public void saveTeam(Team team) {
    Long id = team.getClub().getId();
        if(id != null){
            team.setClub(em.
getReference(Club.class, id));
        }
        em.persist(team);

  }
```

```
  /**
   * @see CompetitionDao#updateTeam(Team)
   */
  public void updateTeam(Team team) {
     em.merge(team);

  }

public void deleteTeam(Team team) {
     em.remove(team);
  }

}
```

Er vallen een paar dingen op:

- De JpaCompetitionDao extend niet van JpaDaoSupport
- De EntityManager is beschikbaar in de klasse, zonder callback.
- De EntityManager wordt op de 'standaard' JPA manier verkregen, door de @PersistenceContext annotatie.

De voordelen van het gebruik van Spring (resource- en transactiebeheer) kunnen nog steeds benut worden, zonder de overhead van een extra laag.

CONFIGURATIE Nu we gezien hebben hoe Spring geïntegreerd kan worden in onze code, bespreken we hoe we gebruik kunnen maken van het resource beheer, transactie beheer en foutafhandeling van Spring. Er zijn twee typen Spring containers: de BeanFactory en de ApplicationContext. Deze laatste is krachtiger en wordt over het algemeen gebruikt in applicaties. Deze ApplicationContext wordt geconfigureerd met behulp van een xml file. Hieronder volgt een voorbeeldconfiguratie van onze applicatie:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/
schema/beans"
       xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
       xmlns:aop="http://www.springframework.
org/schema/aop"
       xmlns:tx="http://www.springframework.
org/schema/tx"
       xsi:schemaLocation="http://www.
springframework.org/schema/beans http://
www.springframework.org/schema/beans/
spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/
spring-tx.xsd http://www.springframework.org/
schema/aop http://www.springframework.org/
schema/aop/spring-aop.xsd">
```

```
 <bean name="competitionDao"
       class="com.xebia.demo.hockey.dao.jpa.
JpaCompetitionDao">
  <property name="entityManagerFactory"
ref="entityManagerFactory"/>
 </bean>

<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.Loca
lContainerEntityManagerFactoryBean">
  <property name="persistenceUnitName"
value="competitionPersistenceUnit"/>
  <property name="dataSource"
ref="dataSource"/>
  <property name="jpaVendorAdapter">
   <bean class="org.springframework.orm.jpa.
vendor.TopLinkJpaVendorAdapter">
    <property name="showSql" value="true"/>
    <property name="generateDdl"
value="false"/>
   </bean>
  </property>
  <property name="loadTimeWeaver">
   <bean class="org.springframework.
instrument.classloading.
SimpleLoadTimeWeaver"/>
  </property>
 </bean>

 <!-- transaction manager that is used -->
 <bean id="transactionManager"
       class="org.springframework.orm.jpa.
JpaTransactionManager">
  <property name="entityManagerFactory"
ref="entityManagerFactory"/>
  <property name="dataSource"
ref="dataSource"/>
 </bean>

 <!-- the service class that is injected when
the JSF pages call competitionService -->
 <bean name="competitionService"
       class="com.xebia.demo.hockey.service.
impl.CompetitionServiceImpl">
  <property name="competitionDao">
   <ref bean="competitionDao"/>
  </property>
 </bean>

 <!-- JNDI DataSource for J2EE environments
-->
 <bean id="dataSource" class="org.
springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName" value="jdbc/
competitionDS"/>
 </bean>

<tx:annotation-driven/>

</beans>
```

RESOURCE BEHEER Zoals uit de configuratie gelezen kan worden, beheert Spring de resources. Entity Managers worden niet geopend en gesloten in de code, dat handelt Spring af. Hetzelfde geldt voor datasources en andere resources. Deze configuratie kan gebruikt worden door zowel de klasse die extend van JpaDaoSupport als voor de 'schone' JpaCompetitionDao.

TRANSACTIES Als we Spring willen gebruiken voor transactiebeheer, moeten we een Spring-specifieke annotatie toevoegen aan onze businessmethoden. Een voorbeeld hiervan zien we hieronder.

```
/**
 *
 * @see CompetitionService#saveTeam
 */
  @Transactional
  public void saveTeam(Team team) {
     competitionDao.saveTeam(team);
  }
```

In ons configuratiebestand geven we als volgt aan dat we gebruik willen maken van deze annotaties:

```
<tx:annotation-driven/>
```

Een alternatief voor de annotatie bij de methode is het gebruik van de xml-configuratie om de transacties te definiëren. Op deze manier wordt Spring-specifieke code (annotatie) vermeden in de Java-klasse:

```
<aop:config>
    <aop:pointcut id="competitionServiceMetho
ds" expression="execution(* service.Competiti
onService.*(..))"/>
    <aop:advisor advice-ref="txAdvice"
pointcut-ref="competitionServiceMethods"/>
  </aop:config>

  <tx:advice id="txAdvice"
transaction-manager="myTxManager">
    <tx:attributes>
      <tx:method name="deleteTeam"
propagation="REQUIRES_NEW"/>
    <tx:method name="*"
propagation="REQUIRED"/>
    </tx:attributes>
  </tx:advice>
```

Wat aan beide opties opvalt, is dat er Spring-specifieke annotaties (of xml-configuraties) gebruikt worden. Dit heeft te maken met de uitgebreide transactiemogelijkheden die Spring biedt. Zo kunnen in de Spring-configuratie niet alleen 'propagation'-grenzen maar ook 'isolation'-niveaus gedefinieerd worden.

In de Spring-configuratie niet alleen 'propagation'-grenzen maar ook 'isolation'-niveaus gedefinieerd worden

FOUTAFHANDELING Spring biedt naast resourcemanagement ook foutafhandeling. Fouten uit de onderliggende datalaag worden vertaald naar generieke Spring-excepties, onafhankelijk van de gekozen technologie (TopLink, Hibernate of JDBC). Wanneer men gebruik maakt van de JpaTemplate, worden alle exceptions vertaald. Als de foutafhandeling ook gewenst is in de variant die niet extend van JpaDaoSupport, kan dit toegevoegd worden door de annotatie @Repository op de klasse en een bean-configuratie in de ApplicationContext.

```
@Repository
public class CompetitionServiceImpl
implements CompetitionService {
```

Het configuratiebestand moet de volgende bean bevatten:

```
<bean class="org.springframework.orm.jpa.
support.PersistenceAnnotationBeanPost
Processor" />
```

ENTITYMANAGERS Zoals eerder besproken, beheert Spring de resources. Dit geldt ook voor de EntityManagers. Er zijn drie manieren om een Entity Manager te configureren met Spring:

1. LocalEntityManagerFactoryBean. Deze bean creeert een EntityManager volgens de JPA-standaard voor het lokale bootstrap-contract. De LocalEntity ManagerFactoryBean wordt vooral gebruikt in stand-alone Clients waarbij de configuratie uit de persistence.xml wordt gelezen. Eigenschappen als global transacties worden niet ondersteund. Deze variant wordt vooral gebruikt bij unit- en integratie testen. Een typische configuratie ziet er als volgt uit:

```
<bean id="entityManagerFactory"
       class="org.springframework.orm.jpa.
LocalEntityManagerFactoryBean">
  <property name="persistenceUnitName"
value="competitionPersistenceUnit"/>
</bean>
```

2. LocalContainerEntityManagerFactoryBean. Dit is de Spring-implementatie van de JPA-standaard voor het container bootstrap-contract. De persistence.xml wordt uitgelezen voor de configuratie, maar kan in de applicatiecontext worden overschreven. Ook wordt het gebruik van meerdere persistence units ondersteund door middel van de PersistenceUnitManager. Deze fungeert als een centrale repository. Als deze niet gedefinieerd is in de ApplicationContext, creëert de FactoryBean er intern zelf één. Een voorbeeld van de configuratie is als volgt:

```
<bean id="entityManagerFactory"
       class="org.springframework.orm.jpa.Loca
lContainerEntityManagerFactoryBean">
  <property name="persistenceUnitName"
value="competitionPersistenceUnit"/>
  <property name="dataSource"
ref="dataSource"/>
  <property name="jpaVendorAdapter">
   <bean class="org.springframework.orm.jpa.
vendor.TopLinkJpaVendorAdapter">
    <property name="showSql" value="true"/>
    <property name="generateDdl"
value="false"/>
   </bean>
  </property>
  <property name="loadTimeWeaver">
   <bean class="org.springframework.
instrument.classloading.
SimpleLoadTimeWeaver"/>
  </property>
 </bean>
```

3. De EntityManager van de container. Deze kan worden opgezocht met JNDI in de ApplicationContext. xml. De configuratie hiervan is simpel:

```
<jndi:lookup id="entityManagerFactory"
jndi-name="jpa/competitionPersistenceUnit"/>
```

Conclusie In dit artikel heb ik laten zien dat het gebruik van Spring met de Java Persistence API een aantal voordelen biedt:

- Rijke transactie mogelijkheden van Spring in combinatie met JPA
- Resource beheer
- Foutafhandeling
- Testbaarheid door het configureren van EntityManagerFactory in plaats van het coderen van deze factory.

De ontwikkelaar heeft de keuze om het bekende patroon van de DaoSupport en bijbehorende Template te gebruiken, dan wel de JPA code aangevuld met annotaties en configuratie. Beide bieden dezelfde voordelen, het hangt af van de achtergrond en smaak van de ontwikkelaar welke methode de voorkeur verdient.

Referenties

Spring algemeen:
http://www.springframework.org

JPA:
faq: http://java.sun.com/javaee/overview/faq/persistence.jsp
specificatie: http://www.jcp.org/en/jsr/detail?id=220
reference implementatie: https://glassfish.dev.java.net/javaee5/persistence/

Spring JPA:
Reference: http://static.springframework.org/spring/docs/2.0.x/reference/orm.html#orm-jpa
Oracle howto: http://www.oracle.com/technology/tech/java/index.html

Lonneke Dikmans is werkzaam bij Xebia Group.

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

Artikelen over onderwerpen als software-ontwikkeling, Java, UML, eXteme Programming en nog veel meer vindt u in het Online Archief van Array Publications. Vaktijdschriften als Software Release, Java Magazine, Database Magazine en ons Oracle vakblad Optimize hebben hun artikelenarchief online gezet. Dankzij de heldere zoekstructuur vindt u snel wat u zoekt op www.release.nl.

Sun geeft broncode Java-platform vrij

Sun Microsystems, de uitvinder en oorspronkelijke ontwikkelaar van de Java-technologie, geeft vandaag de eerste delen van de broncode vrij, voor Java Platform Standard Edition (Java SE) en Java Platform Micro Edition (Java ME). De Java-technologie wordt als gratis software vrijgegeven onder de GNU General Public License versie twee (GPLv2). Ook voegt Sun de GPLv2-licentie toe aan de Java Platform Edition (Java EE), die via project GlassFish al een jaar beschikbaar was onder de Common Development en Distribution License (CDDL).

De aankondiging betekent één van de grootste bijdragen van source code onder de GPL-licentie. Met meer dan 3,8 miljard apparaten die gebruik maken van Java, kent de technologie een enorm bereik. Van mobiele telefoons tot smartcards, Java-technologie biedt één platform voor software-innovatie. Door Java in open source beschikbaar te stellen en klanten daarnaast commerciële producten met volledige garantie te bieden, verwacht Sun dat het gebruik van Java-technologie verder toeneemt.

Sun geeft drie belangrijke software-componenten vrij van Java SE; Java HotSpot technologie, javac en JavaHelp software. Java HotSpot technologie is de Sun Java Virtual Machine (JVM) en het belangrijkste component van de Java Runtime Environment (JRE). Deze vertaalt Java-code naar het besturingssysteem en de chiparchitectuur waardoor Java-software overal gebruikt kan worden. Javac is de compiler (het vertaalprogramma) die Java-broncode analyseert op fouten en omzet in programmacode.

De broncode voor Sun-implementatie van Java ME, voor mobiele toepassingen, is nu ook vrij beschikbaar in de Java.net gemeenschap. Dit platform biedt nu al mobiele dataservices op 1,5 miljard telefoontoestellen. Ook geeft Sun de code vrij voor Java's ME test- en compatibiliteitskit. Sun stelt deze technologie nu beschikbaar om de ontwikkeling van het platform te versnellen, fragmentatie te verminderen en ontwikkelkosten voor Java ME omlaag te brengen. Hiermee biedt Sun eenvoudig toegang tot de nieuwste Java ME-platformtechnologie zodat de gehele Java ME-community zelf actief kan bijdragen aan de verdere ontwikkeling.

"Door Sun's implementatie van Java in open source beschikbaar te stellen, stimuleren we nieuwe vormen van samenwerking tussen ontwikkelaars en verdere innovatie door het gebruik van NetBeans. Ook verwachten we dat het Java-platform de basisinfrastructuur wordt voor een nieuwe generatie internet,- desktop,- mobiele,- en zakelijke applicaties," vertelt Rich Green, executive vice president Software bij Sun. "Voor het vrijgeven van de Java Developer Kit (JDK), werkt Sun nauw samen met distributeurs van het GNU/ Linux besturingsysteem, die binnenkort de JDK samen met GNU/Linux kunnen aanbieden."