Een 'content repository' is een hiërarchisch gestructureerde verzameling gegevens en faciliteiten om deze gegevens op te slaan, te beheren en toegankelijk te maken. Daarnaast onderscheidt een content repository zich veelal door het bieden van versiebeheer, authenticatie, autorisatie, transacties en zoekfaciliteiten.

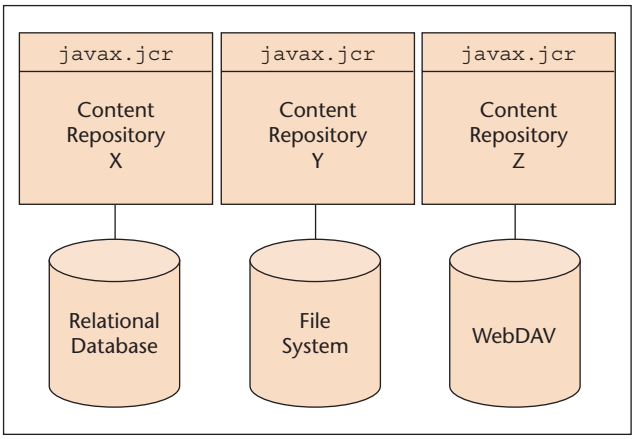
Java Content Repository API

Gegevenspersistentie en -beheer

De meest voorkomende toepassingen van een content repository voor gegevenspersistentie zijn:

- Content beheer systemen; meestal gebruikt om het mogelijk te maken voor niet-technische gebruikers om bijvoorbeeld inter- of intranet pagina's inhoudelijk te beheren.
- Document management systemen; gebruikt om grote hoeveelheden documenten (van origine digitaal of ingescand) te beheren.

De populariteit van dergelijke systemen is de laatste jaren enorm toegenomen en er is dan ook een ware wildgroei ontstaan aan commerciële en open-source software die doorgaat voor content repository.



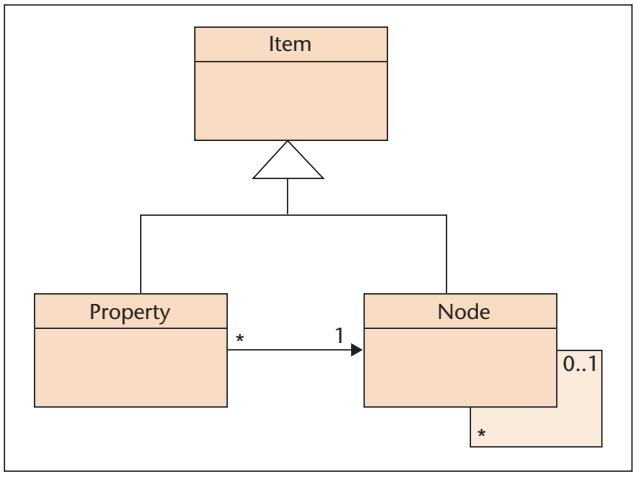
Figuur 1. Content repository's met verschillende opslagmethoden.

Dit is de Java-community niet ontgaan en om de hierdoor anders onvermijdelijke 'vendor lock-in' tegen te gaan heeft ze gezorgd voor een standaardspecificatie, genaamd de Content Repository API for Java (JSR 170).

Java Content Repository API De JCR API is een gemeenschappelijke, content-repository onafhankelijke API voor toegang tot content repository's vanuit Java. De voordelen hiervan zijn evident:

- Ontwikkelaars kunnen volstaan met het zich eigen maken van één API.
- Het gebruik van de JCR API beschermt tegen mogelijk ongewenste situatie waar men van een bepaalde leverancier (vendor) afhankelijk is.

Het nadeel van de JCR en iedere standaard-API is echter, dat leverancierspecifieke functionaliteiten die niet in de API gevat zijn ook niet gebruikt kunnen worden zonder de eerdergenoemde voordelen teniet te doen. De specificatie is onderverdeeld in 'compliance levels'. Dit geeft leveranciers van content repository's de mogelijkheid om de specificatie geheel of slechts deels te ondersteunen. Level 1 beslaat het zoeken van items, het lezen hiervan en heeft de mogelijkheid gegevens vanuit de content repository te exporteren naar XML. Level 2 voegt hier de mogelijkheid tot toevoegen van nieuwe items, het modificeren van bestaande items en het importeren van de eerdergenoemde XML aan toe. Daarnaast beschrijft de specificatie een aantal



Figuur 2. UML diagram van Item, Property, Node en hun relaties.

functionaliteiten die optioneel zijn, zoals transacties, versiebeheer, event observatie en lifecycle management.

VERGELIJKING De JCR API kent overeenkomsten met JDBC toegang tot relationele databases en daarom wordt ook wel gesproken van een 'JDBC voor content repository's'. Een content repository bevindt zich echter op een ander niveau daar men bij het gebruik van een Java Content Repository zich niet hoeft te bekommeren om hoe de gegevens daadwerkelijk worden opgeslagen. Het zou kunnen dat een bepaald content repository gebruik maakt van een relationele database om gegevens fysiek op te slaan, maar het kan evenwel het filesysteem, WebDAV, of een XML-bestand zijn (zie Figuur 1).

JCR DATAMODEL De specificatie beschrijft een zeer overzichtelijk datamodel waardoor de leercurve niet al te steil is. Een content repository bestaat uit één of meer 'workspaces'. Een workspace bestaat uit een boomstructuur van 'items'. Een item is een 'node' of een 'property'. Een property bevat de daadwerkelijke gegevens. Een node bevat een lijst met nul of meer andere items. Hierdoor geven nodes structuur aan de verzameling gegevens, zoals directory's structuur geven aan files in een filesysteem (zie Figuur 2).

VOORBEELD Als voorbeeld schrijven we een adresboekapplicatie die gebruik maakt van een content repository. De specifieke content repository implementatie die gebruikt wordt in dit voorbeeld is Apache Jackrabbit.

```
import org.apache.jackrabbit.core.
TransientRepository;
import javax.jcr.*;
import javax.jcr.query.*;
```

```
public class AddressBook {

  private Repository repository;

  private Session session;

  public AddressBook() throws Exception {
    System
        .setProperty(
            "org.apache.jackrabbit.
repository.home",
            "/tmp/contrep/addressbook");

    repository = new TransientRepository();
    session = repository
        .login(new SimpleCredentials(
            "username", "password"
                .toCharArray()));
  }

  public void addAddressBookEntry(
      String name, String address)
      throws Exception {
    Node adentry = session
        .getRootNode()
        .addNode("adentry", "nt:unstructu-
red");
    adentry.setProperty("name", name);
    adentry.setProperty("address", address);
    session.save();
  }

  public void listAddressBookEntries()
      throws Exception {
    NodeIterator nit = session.getRootNode()
        .getNodes("adentry");

    while (nit.hasNext()) {
      Node adentry = nit.nextNode();
      System.out.println(adentry.getProperty(
          "name").getString()
          + " : "
          + adentry.getProperty("address")
              .getString());
    }
  }

  public void clearAddressBook()
      throws Exception {
    NodeIterator nti = session.getRootNode()
        .getNodes("adentry");

    while (nti.hasNext()) {
      nti.nextNode().remove();
    }
    session.save();
  }

  public void close() throws Exception {
    session.logout();
```

```
  }

  public String lookupAddress(String name)
      throws Exception {
    QueryManager queryManager = session
        .getWorkspace().getQueryManager();
    Query query = queryManager.createQuery(
        "//adentry[@name= '" + name + "']",
        Query.XPATH);
    QueryResult queryResult = query.execu-
te();

    NodeIterator nit = queryResult.getNo-
des();

    if (nit.hasNext())
      return nit.nextNode().getProperty(
          "address").getString();
    return "Niet gevonden";
  }

  public static void main(String[] args)
      throws Exception {

    AddressBook addressBook = null;
    try {
      addressBook = new AddressBook();
      addressBook.clearAddressBook();
      addressBook.addAddressBookEntry(
          "James Gosling", "Java Drive 96");
      addressBook
          .addAddressBookEntry(
              "Bjarne Stroustrup",
              "C++ Drive 85");

      System.out.println(addressBook
          .lookupAddress("James Gosling"));

      addressBook.listAddressBookEntries();
    } finally {
      if (addressBook != null)
        addressBook.close();
    }
  }
}
```

De JCR API classes en interfaces bevinden zich het javax.jcr package. Een javax.jcr.Repository representeert een content repository en een javax.jcr.Session object wordt gebruikt om te lezen en te schrijven naar een content repository. In dit Java-fragment is ter demonstratie gebruik gemaakt van een content repository specifieke afhankelijkheid, een TransientRepository object. In productiecode is het gebruik van de Java Naming and Directory Interface (JNDI) of een ander configuratiemechanisme, zoals het dependancy injection design pattern, aan te bevelen om deze afhankelijkheid te elimineren.

Allereerst wordt toegang tot de content repository op locatie '/tmp/contrep/addressbook' verschaft. Dit is een 'create-or-replace'-actie, dus zal als deze niet bestaat een lege repository worden aangemaakt. Wanneer een content repository geconfigureerd is om een vorm van authenticatie en autorisatie uit te voeren, dan kunnen de zogenaamde 'credentials' aan javax.jcr.Repository.login() meegegeven worden.

Een nieuwe toevoeging aan het adresboek kan gedaan worden met AddressBook.addAddressBookEntry(). Hier is te zien hoe met javax.jcr.Session.getRootNode() de root node van de huidige workspace opgevraagd kan worden. Een nieuwe node met daaronder twee property's voor de naam en het adres worden vervolgens hieraan toegevoegd. De gebruikte javax.jcr.Repository implementatie maakt gebruik van transacties waardoor modificaties vergankelijk (transient) zijn. Met javax.jcr.Session.save() worden de wijzigingen definitief gepersisteerd. In AddressBook.listAddressBookEntries() en AddressBook.clearAddressBook() is te zien hoe met behulp van javax.jcr.Node.getNodes() nodes die aan een bepaald naampatroon voldoen op te vragen zijn. De resultaten worden respectievelijk geprint en verwijderd door middel van javax.jcr.Item.remove().

De specificatie beschrijft een zeer overzichtelijk datamodel waardoor de leercurve niet al te steil is

XPath is een taal ontwikkeld door de W3C voor het adresseren van informatie uit XML-documenten. Tijdens het opstellen van JSR 170 is echter onderkent dat XPath met enige aanpassingen evenwel toepasbaar is op andere gestructureerde gegevens en dus ook een Java Content Repository. In AddressBook.lookupAddress() is te zien hoe door gebruik van XPath syntax een node met daaronder een property met een bepaalde naam te vinden is.

JCR 2.0 PREVIEW In het Java-fragment is te zien dat nodes getypeerd kunnen worden, zie AddressBook.addAddressBookItem() waar ter verduidelijking expliciet het default type 'nt:unstructured' aan de node toegekend wordt. Door een primair type toe te kennen kunnen restricties opgelegd worden aan de items die zich onder een ander item mogen bevinden. Bijvoorbeeld, twee voorgedefinieerde types zijn 'nt:directory' en 'nt:file' waarbij een 'nt:directory' node afdwingt dat zijn kinderen enkel van type 'nt:directory' of 'nt:file' zijn.

In de huidige JCR-specificatie (JSR 170) is het niet mogelijk om zelf nieuwe types te declareren. Hierdoor is

het dus onmogelijk dit te doen zonder afhankelijk te zijn van leverancier specifieke faciliteiten. Zeer recentelijk is een 'early draft' van JCR 2.0 (JSR 283) ter review beschikbaar gekomen. In deze versie van de specificatie is de meest in het oog springende nieuwe functionaliteit de mogelijkheid om zelf type declaraties te construeren.

Conclusie Een content repository is niet de heilige graal en zeker niet de vervanging van relationele databases en object-relational-mapping. Althans, niet in alle gevallen. Object-relational-mapping kan gebruikt worden om allerlei typen applicaties te persisteren. Een content repository daarentegen is enkel effectief inzetbaar bij toepassingen die zich concentreren op hiërarchisch gestructureerde gegevens, zoals content- en document management systemen. Er is veel werk verzet om tot de initiële JCR API te komen en belangrijke partijen als IBM, BEA, Oracle en FileNet zijn op de wagen gesprongen. Uiteraard blijft er nog wel een en ander te wensen over. De grootste tekortkoming is de onmogelijkheid om zelf types te definiëren, maar gezien de 'early draft' van JSR 283 lijkt dit in JCR 2.0 opgepakt te worden. Om zelf aan de slag te gaan met een Java Content Repository zijn er momenteel vier beschikbaar die de volledige JSR 1.0 specificatie implementeren. Drie daarvan zijn verkrijgbaar onder een open-sourcelicentie (Apache Jackrabbit, eXo JCR en Alfresco) en er is één commerciële, genaamd CRX.

Referenties

- JSR 170: Content Repository for JavaTM technology API - http://jcp.org/en/jsr/detail?id=170
- JSR 283: Content Repository for JavaTM technology API 2.0 - http://jcp.org/en/jsr/detail?id=283
- ONJava.com: What is Java Content Repository - http://www.onjava.com/lpt/a/6745
- Apache Jackrabbit - http://jackrabbit.apache.org/
- Catch Jackrabbit and the JCR API - Catch Jackrabbit and the Java Content Repository API

Drs. ing. Dick Eimers is momenteel werkzaam bij Atos Origin BAS, waar hij zich bezighoudt met Java en open-sourcetechnologie.