

# “Versant JDO Genie für transparente Persistenz mit relationalen Datenbanken.”

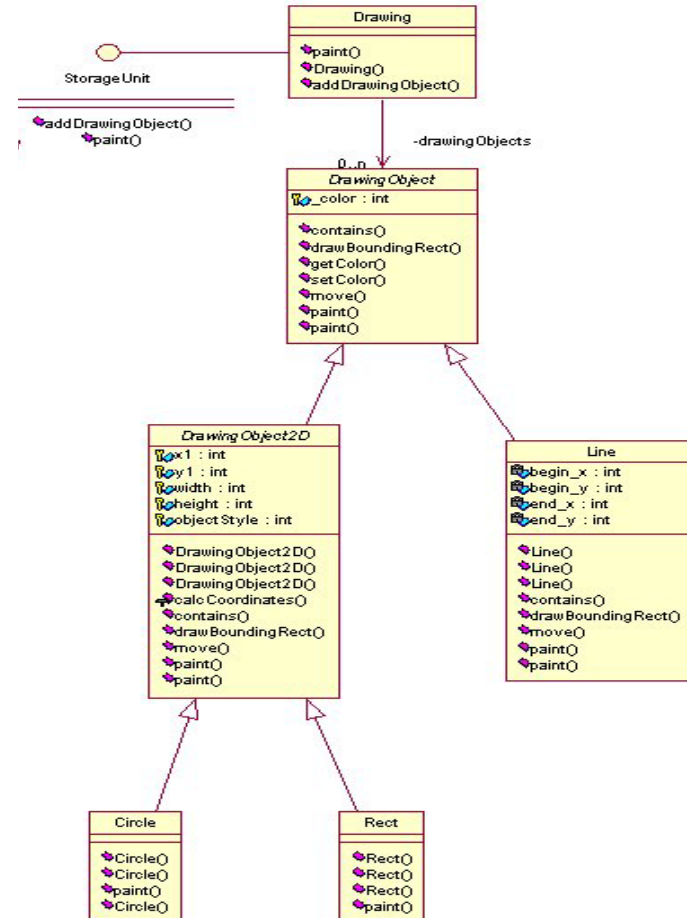
Martin Wessel  
Senior Pre-Sales Consultant  
Versant GmbH  
mailto: [Martin.Wessel@versant.net](mailto:Martin.Wessel@versant.net)

Copyright © Versant Europe 2004.

All products are trademarks or registered trademarks of the respective companies.  
The information contained in this document is property of Versant.

# Sie entwickeln Anwendungen in Java

- Wie machen Sie die Business-Objekte Ihrer Anwendungen persistent?



## Sie benutzen eine relationale Datenbank

- Zugriff mit JDBC
- Bruch in der Datenmodellierung
  - Das Businessmodell ist Java
  - Das Datenmodell ist relational
    - Schlechte Performance bei komplexen Datenmodellen
    - Entwicklung des Mapping-Codes ist sehr aufwendig

**- Sie plegen den Mapping-Code selber.**

## Sie haben bereits ein Mapping-Tool

- Die Pflege des Mapping-Codes übernimmt das Tool
- Das Mapping-Tool unterstützt keinen Standard

**- Ein Wechsel des Mapping-Tools ist sehr aufwendig**

# Transparente Persistenz mit JDO

- JDO ist Standard von Sun für transparente Persistenz
  - Versant ist Mitglied in der Expertengruppe
  - Aktuelle Version JDO 1.0.1
- Die Anwendungslogik ist Java
- Das Datenmodell ist Java

**+ Keine Beschränkungen durch das Mapping**

# Transparente Persistenz mit JDO

- Die Daten werden gespeichert in:
  - Relationalen Datenbanken
    - ORACLE, DB2, MySQL, ...
  - Objekt Datenbanken
    - FastObjects j2, FastObjects t7, Versant Developer Suite

**+ Sie wählen die richtige Datenbank**

# Transparente Persistenz mit JDO

- **Anwendungs Architekturen**
  - J2ME - Eingebettete Systeme
  - J2SE - Client/Server
  - J2EE - Enterprise Java Beans
    - Integration mit Java Connector Architecture (JCA)

**+ Sie wählen die richtige Architektur**

# Konzepte von JDO im Detail

- Entwicklungszyklus
- Persistence by Reachability
- Lazy Loading
- Change Tracking

# Employee

```
public class Employee {
    String name = null;
    float salary;

    private Employee () {
        String name = new String();
        salary = (float) 0;
    }

    public Employee (String name) {
        this.name    = name;
        this.salary = (float) 70000;
    }

    public void modifySalary(float delta) {
        salary = salary + delta;
    }

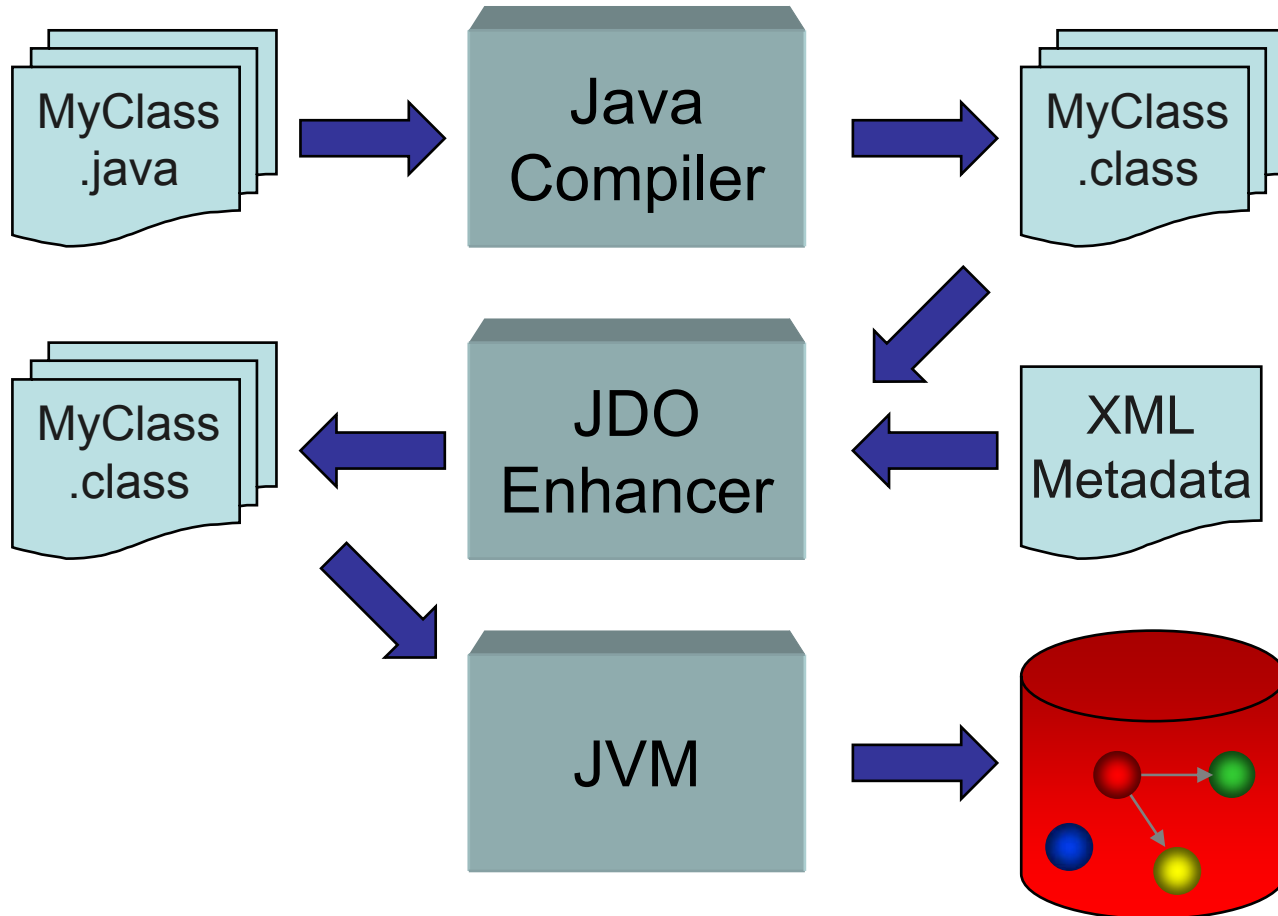
    ...}
}
```

# JDO Meta Data

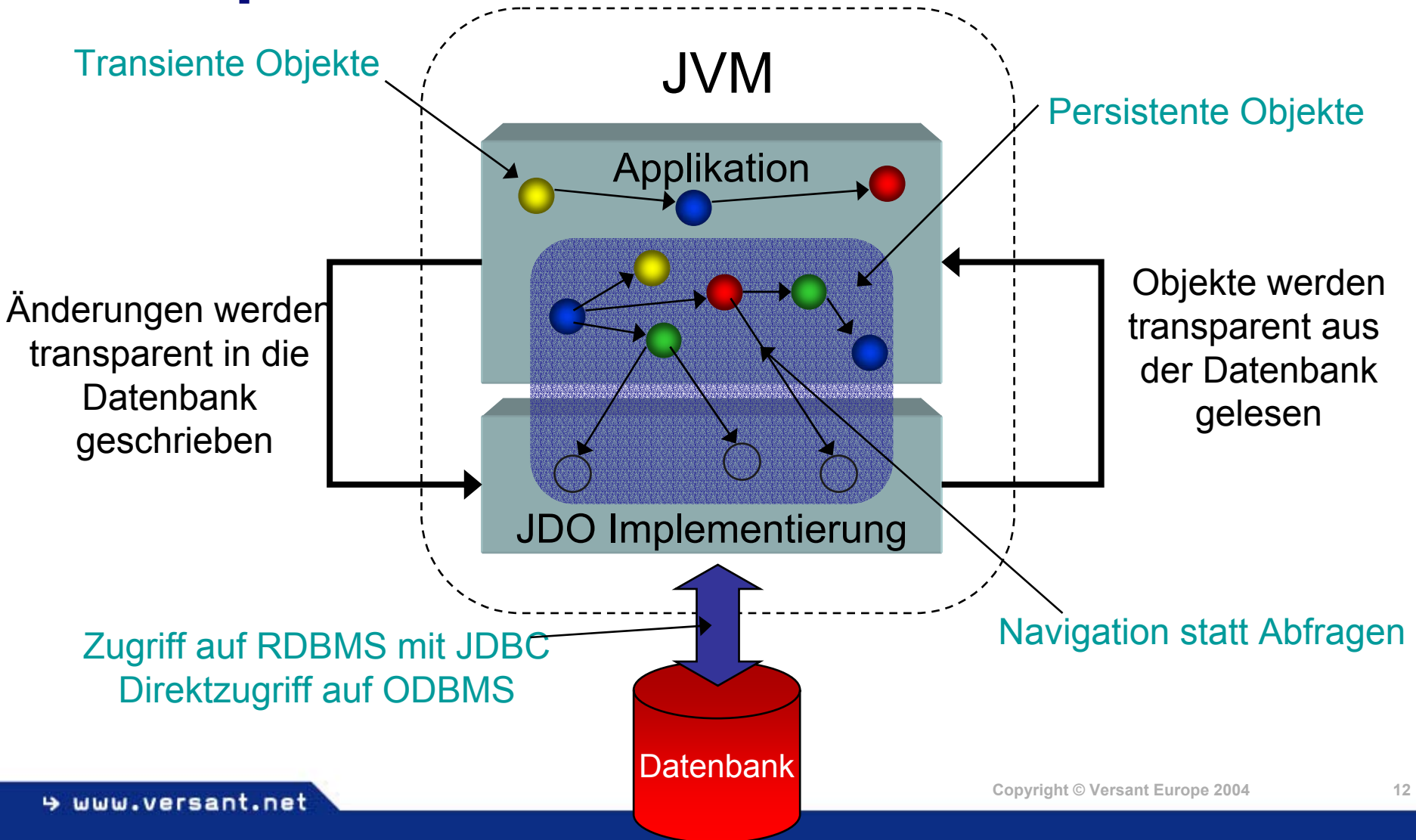
```
<jdo>
  <package name="dilbert">
    <class name="Employee">
      </class>

    <class name="Department">
      <field name="employees" embedded="true">
        <collection element-type="dilbert.Employee"/>
      </field>
    </class>
  </package>
</jdo>
```

# JDO Entwicklungszyklus



# Transparente Persistenz mit JDO



# Department

```
public class Department {
    String name = null;
    Employee manager = null;
    Set employees = new HashSet();

    private Department() {}
    public Department (String name, Employee manager) {
        this.name    = name;
        this.manager = manager;
        employees = new HashSet();
    }

    public void addEmployee(Employee emp) {
        employees.add(emp);
    }
}
```

# Department (Fortsetzung)

```
public Employee getManager() {  
    return manager;  
}
```

```
public Iterator getEmployees() {  
    return employees.iterator();  
}
```

```
public int getNofEmployees() {  
    return employees.size();  
}
```

```
...}
```

# LoadDatabase

```
public class LoadDatabase
{
    public static void main (String[] args) {
        if (args.length != 1) {
            System.out.println ("Usage: LoadDatabase <properties>");
            System.exit (1);
        }
        String      propertiesfile = args [0];

        PersistenceManagerFactory pmFactory =
            JDOFactory.getPersistenceManagerFactory(propertiesfile);

        PersistenceManager pm =
            pmFactory.getPersistenceManager();

        Transaction trx = pm.currentTransaction();
    }
}
```

# LoadDatabase (Fortsetzung)

```
trx.begin();

Employee boss = new Employee ("Boss");
Department department = new Department ("RD", boss);
Employee emp1 = new Employee ("Walt");
Employee emp2 = new Employee ("Dilbert");

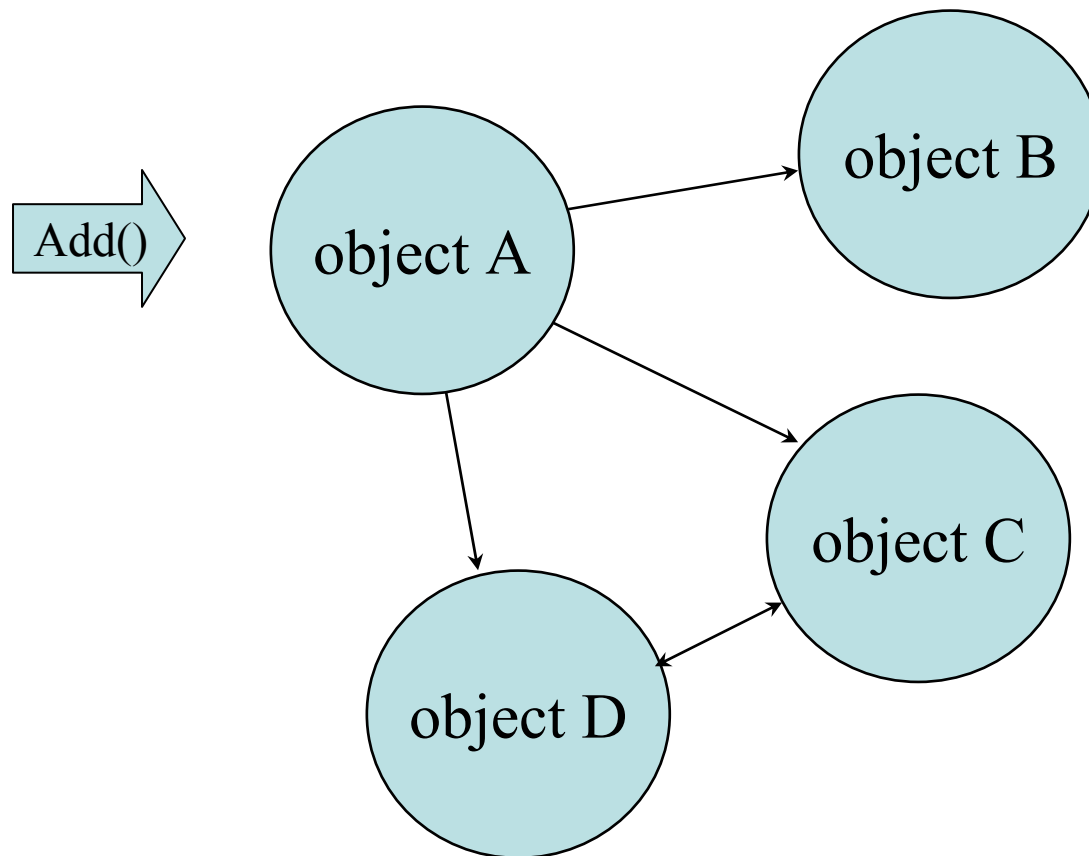
department.addEmployee (emp1);
department.addEmployee (emp2);

pm.makePersistent (department);

trx.commit();

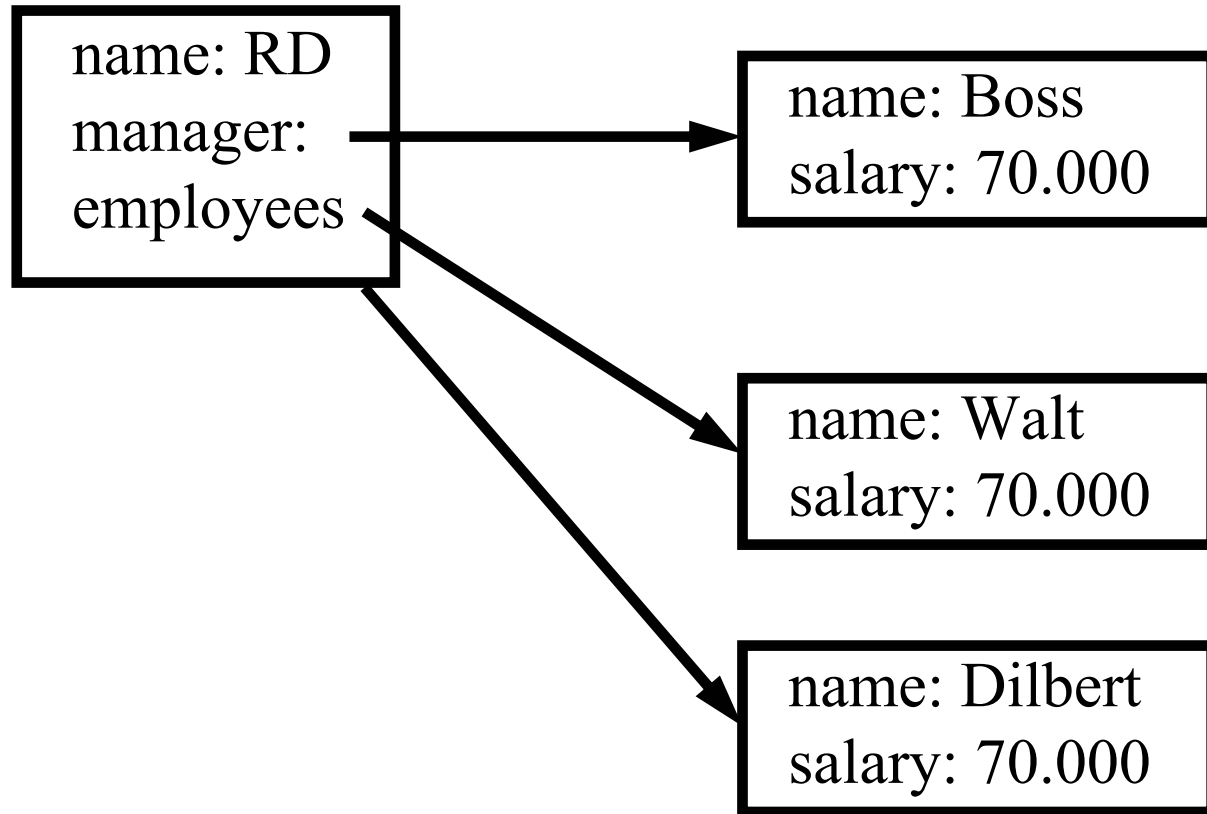
pm.close();
```

# Persistence by Reachability

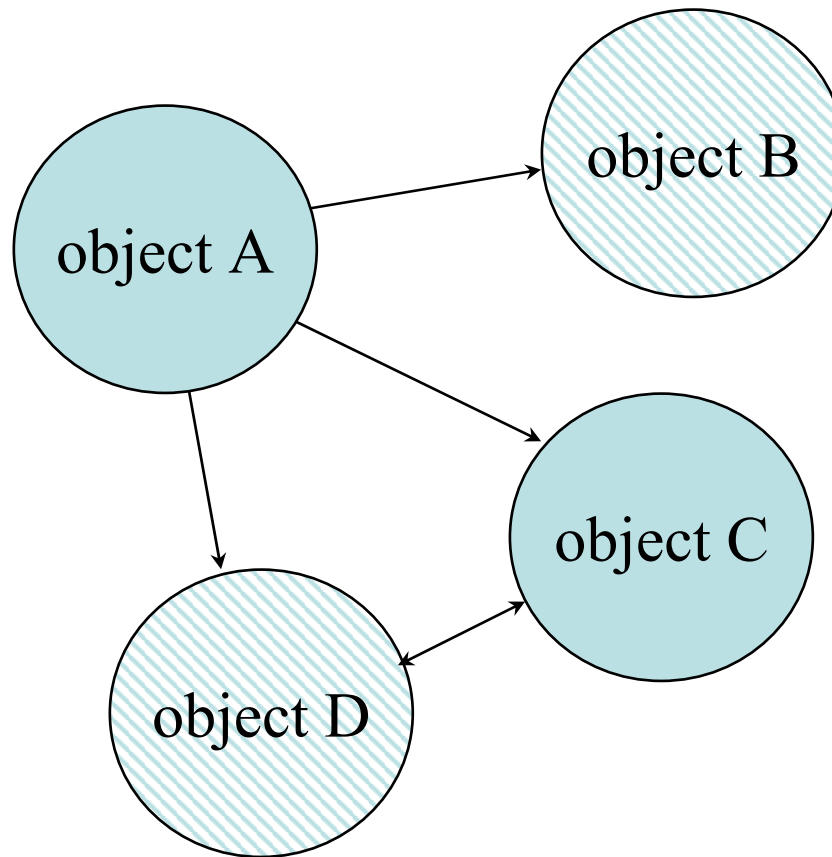


Durch Referenzen direkt oder indirekt erreichbare Objekte gehören ebenfalls dazu.

# Nach LoadDatabase

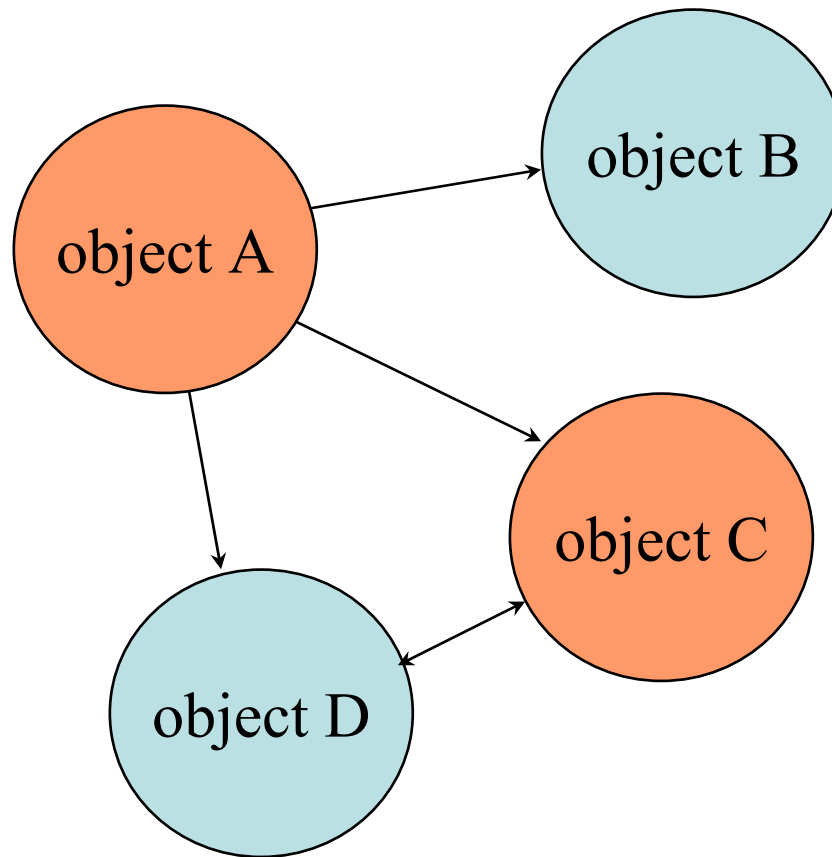


# Lazy Loading



Nur Objekte auf die wirklich zugegriffen wird, sind geladen.

# Change Tracking



Nur neue oder geänderte Objekte werden bei einem Commit() geschrieben

# YearlyJob

```
public class YearlyJob
{
    public static void main (String[] args) {
        if (args.length != 1) {
            System.out.println ("Usage: YearlyJob <properties>");
            System.exit (1);
        }
        String          propertiesfile = args [0];

        PersistenceManagerFactory pmFactory =
        JDOFactory.getPersistenceManagerFactory(propertiesfile);
        PersistenceManager pm =
            pmFactory.getPersistenceManager();

        Transaction trx = pm.currentTransaction();
    }
}
```

# YearlyJob (Fortsetzung)

```
trx.begin();

Query query = pm.newQuery( Department.class,
    "this.manager.name == empName");

query.declareParameters("String empName");

Collection result = (Collection) query.execute ("Boss");
Iterator it = result.iterator();

while (it.hasNext()) {
    typicalDilbertAction((Department) it.next());
}

trx.commit();

pm.close();
```

# TypicalDilbertAction

```
private static final float increase = 5000;

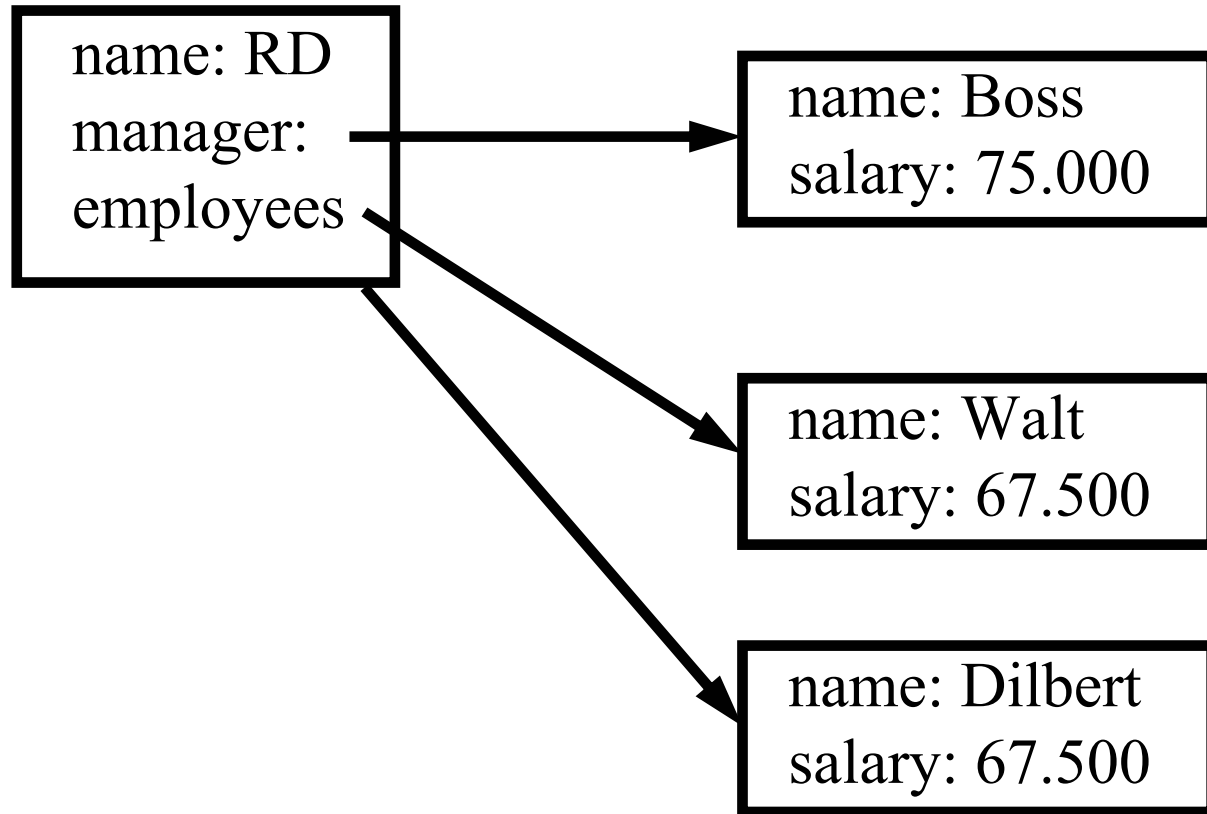
private static void typicalDilbertAction(Department dep) {
    Iterator employees = dep.getEmployees();

    // Manager gets 5000$ more salary
    dep.getManager().modifySalary(increase);

    float decrease = increase/dep.getNofEmployees();

    while (employees.hasNext()) {
        Employee emp = (Employee) employees.next();
        emp.modifySalary(-decrease);
    }
}
```

# Nach YearlyJob

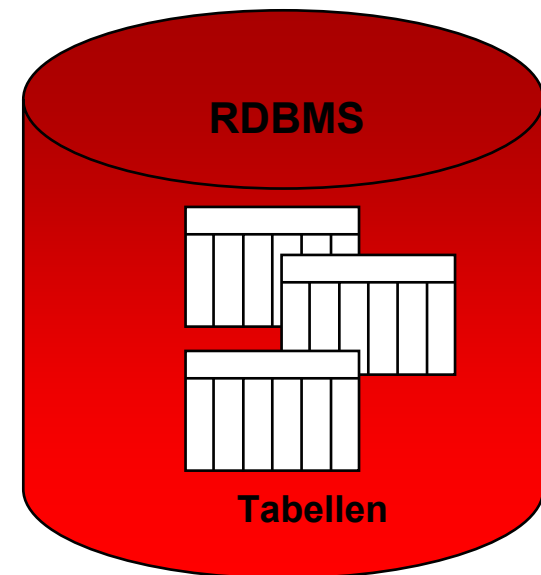


# JDO mit Relationalen Datenbanken

- Reverse Mapping
- Mapping
- Architektur der Anwendung
- Was bietet Versant JDO Genie

# Ihre relationale Datenbank ist schon da

- Sie wollen mit Java auf Ihre relationale Datenbank zugreifen.
- Lassen Sie sich die Klassen generieren.



## + Ihr JDO-Tool sollte Ihnen dabei helfen



<< *Tree view of classes for mapping*

Tree



<< *UML and E/R diagrams*

Diagrams



<< *Grid view for advanced users*

Grid



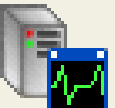
<< *View create table statements and do schema migration*

Schema



<< *Develop JDOQL and SQL queries and run BeanShell scripts*

Queries



<< *View and analyze performance and event logs*

Console



<< *Reverse engineer Java source files from database tables*

Generate

- Versant JDO  
Genie  
Workbench

# Relationales Mapping

- Primärschlüssel
  - Datastore Identity
  - Application Identity
- Mapping von Klassenhierarchien
  - Flat
  - Vertical

# Relationales Mapping

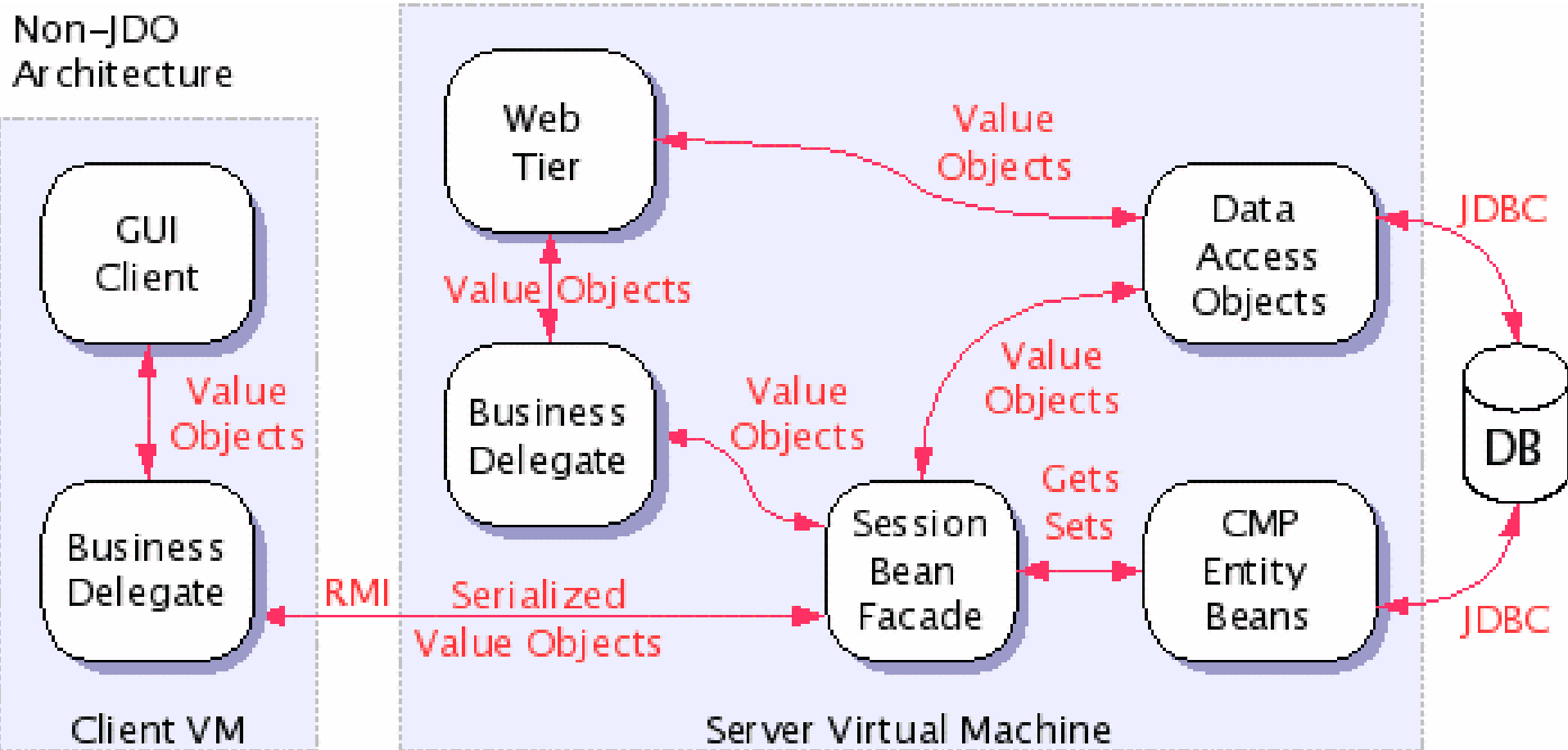
- Feld Mapping
  - Standard Mapping für alle Java Typen
  - Kundenspezifische Anpassungen
  - Automatische Felder (Datum, Zeit)
- Mapping von Referenzen
  - Referenzen auf zusammengesetzte Primärschlüssel
  - Kaskadierendes Löschen
- Mapping von Collections
  - JDK 1.5 Generics
  - Kaskadierendes Löschen
  - Automatisches Löschen von “Waisen”
  - Sortierung der Elemente
    - auch wenn die Liste eigentlich unsortiert ist

# Queries und Transaktionen

- Queries, JDOQL
  - Fetch Groups
  - Erweiterungen (JDO 2.0 preview)
    - toLowerCase, contains, containsKey, isEmpty, sql
- Transaktionen und Locking
  - Pessimistisches Locking
  - Optimistisches Locking

# Mehrstufige Architektur ohne JDO

Non-JDO Architecture

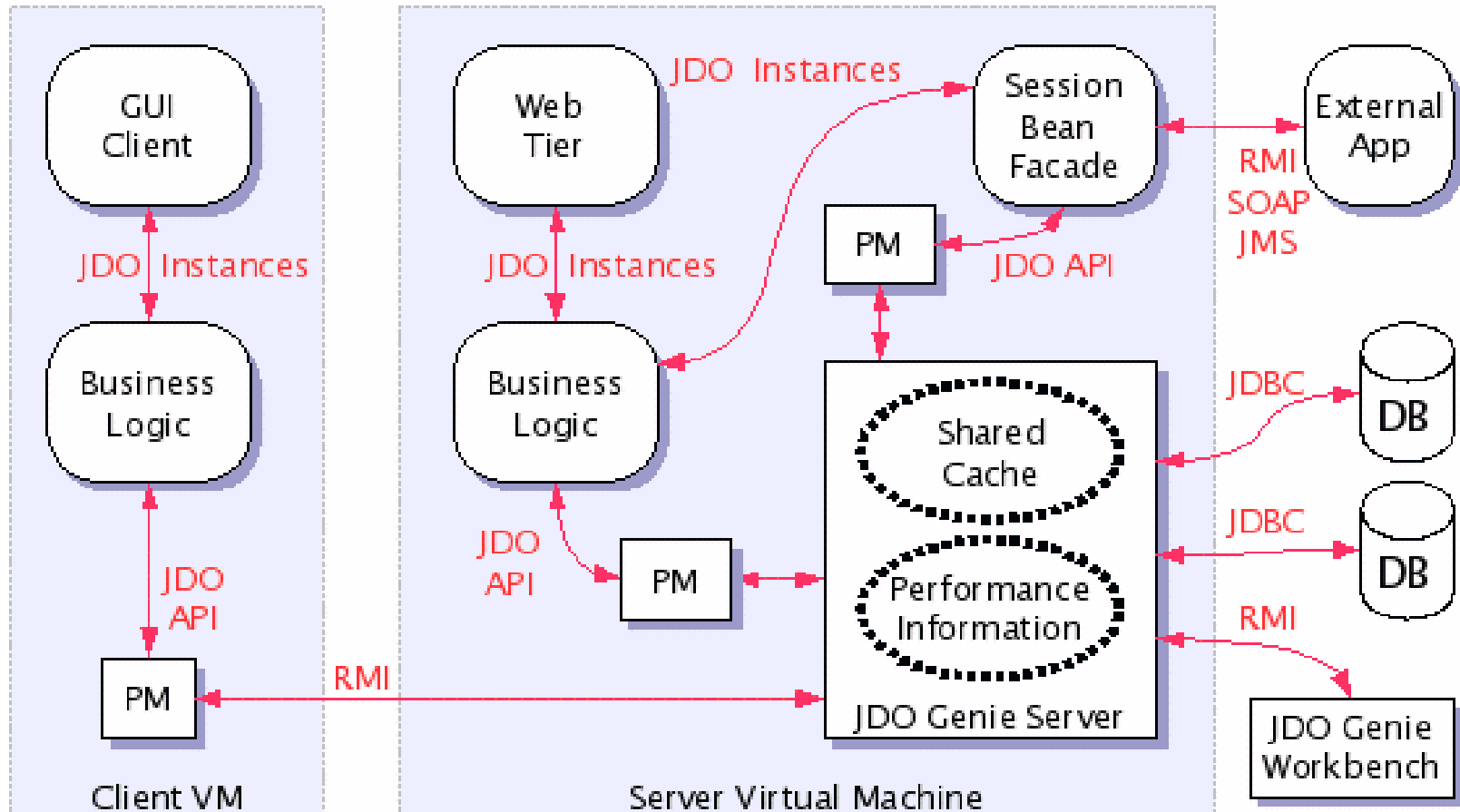


# Cache Management

- Cache pro PersistenceManager ist standard
- Versant JDO Genie bietet einen gemeinsamen Cache für alle PersistenceManager einer PersistenceManagerFactory
  - Benachrichtigungen über Änderungen in einer Clusterumgebung
  - Entfernte PersistenceManager können diesen Cache ebenfalls nutzen
  - Umfangreiches Monitoring

# Mehrstufige Architektur mit JDO Genie

JDO Genie Architecture



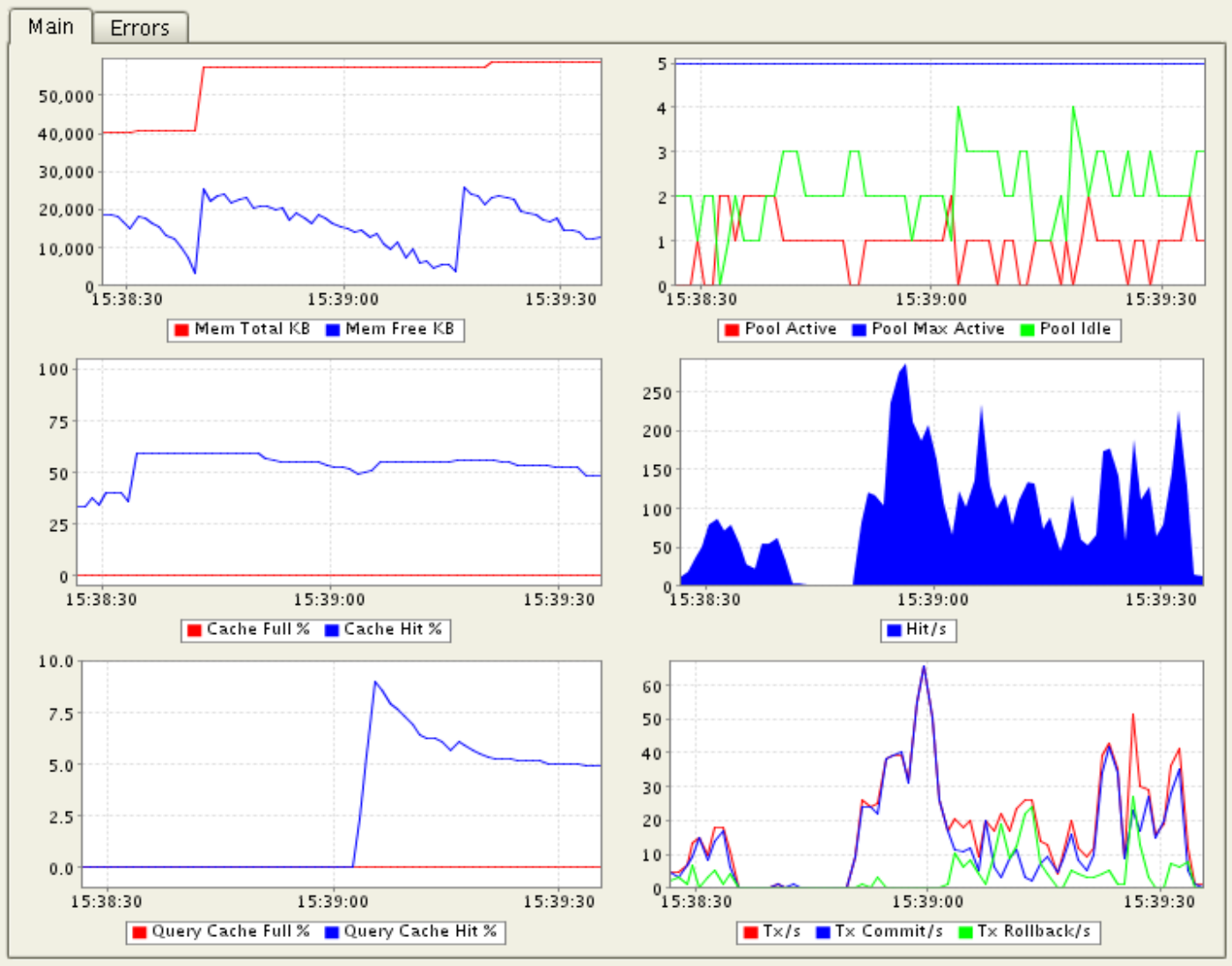
File Edit Build Run Window Help



Performance Configuration Event Log Persistence Managers Remote Clients Cluster

Connect Not Disconnect Refresh Auto refresh Refresh rate (secs) 1

- Tree
- Diagrams
- Grid
- Schema
- Queries
- Console



## Was Sie mitnehmen sollen!

- Transparente Persistenz mit JDO
  - Sie wählen die richtige Datenbank
  - Keine Beschränkung durch das Mapping
  - Sie wählen die richtige Architektur
- Ihr Versant JDO Genie hilft Ihnen bei Ihrer täglichen Arbeit

# Erfinden Sie das Rad nicht neu