

Persistenz: Wie speichere ich meine Geschäftsobjekte?

Vorlesung: Software-Engineering für
große, betriebliche Informationssysteme

für Universität Leipzig, Sommersemester 2004

Institut für Software- und Systementwicklung

Professur Wirtschaftsinformatik, insbes. Informationsmanagement

Hans Hartmann (Generali VIS Informatik Ges.m.b.H., Wien)

Wolfgang Keller (AMB Generali Informatik Services GmbH, Aachen)

WHAT: The 4 Key Messages for „normal“ developers and architects

- know your application style before you decide for a certain way to implement persistence
- know the concept of transparent persistence
- don't develop your own green-field persistence layer unless you do it for fun. That made sense 10 years ago but in the presence of plenty of commercial and open source software for the area it is nowadays too expensive in most cases
- In case you run into problems, know where to find the patterns and explanations on the mechanics of persistence

WHAT this talk is NOT

- this talk is not a comparison for the special option how to provide persistence in J2EE environments, like
 - EJB persistence (CMP and BMP),
 - JDO,
 - and Hibernate
- discussions like that sometimes become a bit „inflamed“ – hence I avoid them ☺

HOW YOU PROFIT

- you might be prevented from making a few expensive mistakes, like
 - using a persistence mechanism that doesn't perform or doesn't fit the problem
 - getting project delay because of writing your own instead of using some product
- and you know where you find more information if you need help with persistence issues

HOW

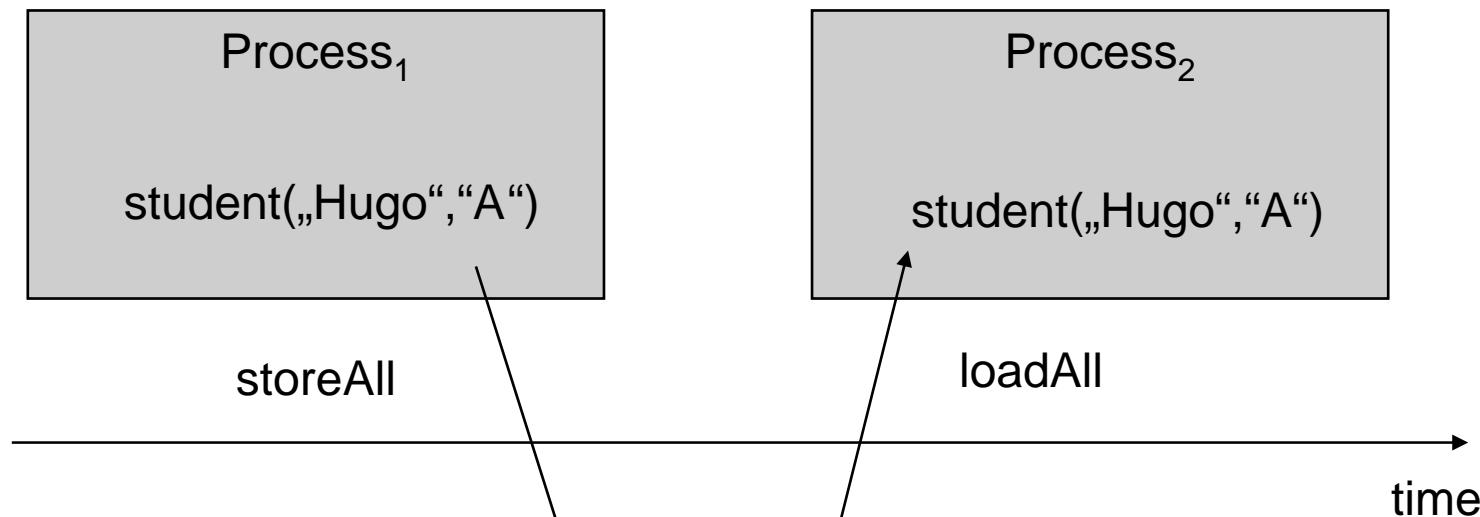
Overview

- what is persistence anyway?
 - persistence defined
 - the concept of transparent persistence
 - persistence interfaces
- application styles
 - when to use o/r mapping and when to use other options
- o/r mappers explained (how to ...) from the primitive to the complex
 - the basics of mapping
 - the basics of implementing o/r mapper features
 - oid, inheritance, relations, transactions,
- persistence in EJBs
- a few remarks on the state-of-the-art in .NET
- summary

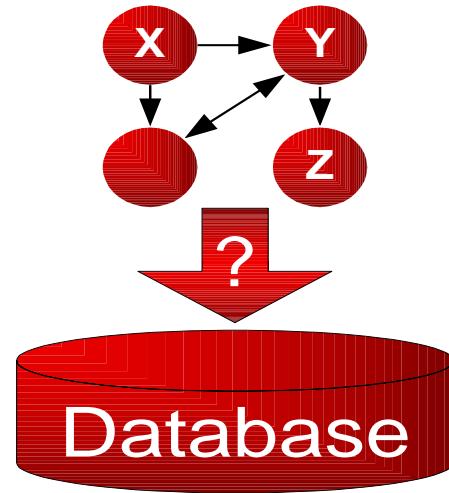
What is Persistence anyway?

Persistence defined

- persistence is the ability of an object to survive the lifecycle of the process in which it resides
- objects that „die“ with the end of a process are called transient



What are the Options to Implement Persistence?



Stream Persistence	Object Database	Object/ Relational Coupling
Object Server	Page Server	ORDBMS

O/R Access Layer

Generations of Database Technology (1)

How Databases evolved

- flat files
 - no efficient key based access
- ISAM/VSAM files
 - efficient access via a key but no concurrency, recovery, logging ...
- hierarchical DBMS (IMS-DB)
 - very efficient as long as access paths are used as planned. Still fastest existing „real“ databases
- network model
 - CODASYL and the like: multiple access paths but also problem as soon as you leave the pre-designed access paths

Generations of Database Technology (2)

How Databases evolved

- relational DBMS
 - very flexible in terms of access - but watch out for performance
 - avoid for graphs, trees, ...
- OODBMS
 - very good performance for pointer based navigation
 - weaknesses in query processing and also data manipulation languages
 - low market share
- Object/Relational Addendums
 - offered by big DB vendors on top of RDBMS
 - but no broad production experiences

What is Persistence anyway? The Concept of Transparent Persistence

- a persistence mechanism is called „transparent“ or also „orthogonal“ if persistent objects are treated no other in the programming environment than transient (non-persistent) objects

Little Q&A session

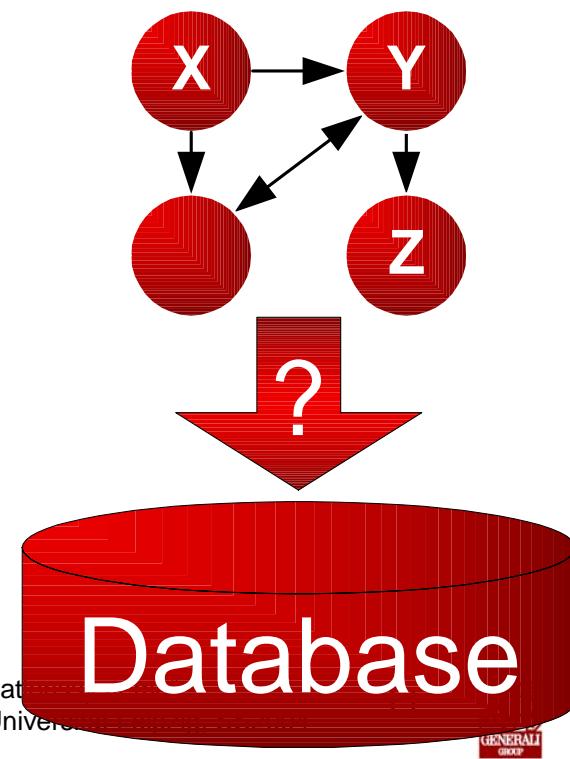
- Q: Is an object-oriented language with a persistence mechanism (be it transparent or non-transparent) automatically an object database?
- A: No - there's a set of features that distinguishes a „database“ from an arbitrary file system or „low profile“ persistence mechanism. The features are:
- Concurrency (Locking, Units of Work (Transactions))
 - Recovery, Logging
 - Security
 - Query Facilities
 - Secondary Storage Management

Little Q&A session

Q: What distinguishes making objects persistent from straight use of a relational database?

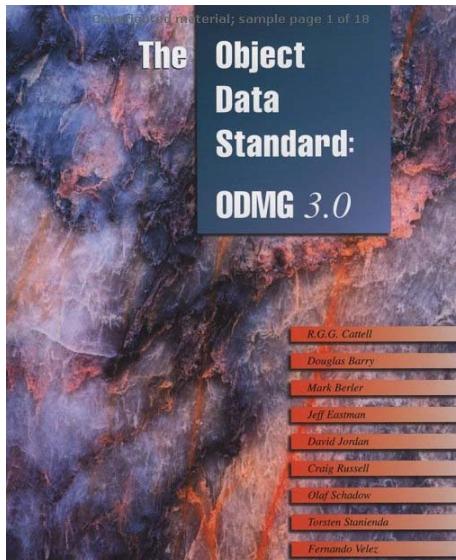
A: o-o languages have a rich set of features not present in relational databases or flat files, like:

- Complex objects
- Object identity
- Encapsulation
- Types and Classes
- Class or Type Hierarchies
- Overriding, overloading and late binding



The ODMG Standard and Interface

What's that?



- ODMG = Object Data Management Group (www.odmg.org)
- has published a standard for object databases in three editions
 - last edition from 2001
 - **the came JDO**
- ISBN 1-55860-647-5
- the standard is often used as a persistence extension to O-O languages.
- there are „language bindings“ for Java, C++, Smalltalk, ...
- **JDO is a follow up standard – only Java => no language bindings**

The ODMG standard and interface Code is better than long explanations :-)

```
public void apply()
{
    String in = readLineWithMessage("Edit Product with id:");
    int id = Integer.parseInt(in);

    // We don't have a reference to the selected Product.
    // So first we have to lookup the object.

    // 1. build oql query to select product by id:
    String oqlQuery = "select del from " +
                       Product.class.getName() +
                       " where _id = " + id;

    Database db = odmg.getDatabase(null); // the current DB
    Transaction tx = null;
    try
    {
        // 2. start transaction
        tx = odmg.newTransaction();
        tx.begin();

        // 3. lookup the product specified by query
        OQLQuery query = odmg.newOQLQuery();
        query.create(oqlQuery);
        DList result = (DList) query.execute();
        Product toBeEdited = (Product) result.get(0);

        // 4. lock the product for write access
        tx.lock(toBeEdited, Transaction.WRITE);
    }
}
```

The ODMG standard and interface Code is better than long explanations :-)

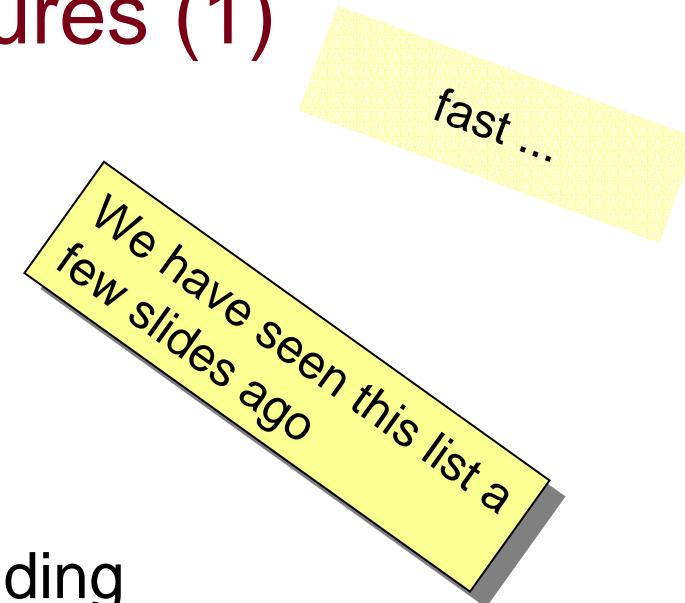
```
// 5. Edit the product entry
System.out.println("please edit existing product");
in = readLineWithMessage(
    "enter name (was " + toBeEdited.getName() + "):");
toBeEdited.setName(in);
in = readLineWithMessage(
    "enter price (was " + toBeEdited.getPrice() + "):");
toBeEdited.setPrice(Double.parseDouble(in));
in = readLineWithMessage(
    "enter available stock (was "+toBeEdited.getStock() + "):");
toBeEdited.setStock(Integer.parseInt(in));

// 6. commit transaction
tx.commit();
}
catch (Throwable t)
{
    // rollback in case of errors
    tx.abort();
    t.printStackTrace();
}
}
```

This is transparent enough but not really what was envisioned when transparent persistence was defined and „invented“

The Object-Oriented Database System Manifesto – Mandatory Features (1)

- Complex objects
- Object identity
- Encapsulation
- Types and Classes
- Class or Type Hierarchies
- Overriding, overloading and late binding
- Computational completeness



feature list compiled from: [Atk+89] Malcolm P. Atkinson, François Bancilhon, David J. DeWitt, Klaus R. Dittrich, David Maier, Stanley B. Zdonik: [The Object-Oriented Database System Manifesto](#). in "Deductive and Object-Oriented Databases", Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), pp. 223-240

The Object-Oriented Database System Manifesto – Mandatory Features (2)

- Extensibility (of the Type System)
- Persistence
- Secondary storage management
- Concurrency
- Recovery
- Ad Hoc Query Facility

We have seen a similar list as the „database distinctive features“

feature list compiled from: [Atk+89] Malcolm P. Atkinson, François Bancilhon, David J. DeWitt, Klaus R. Dittrich, David Maier, Stanley B. Zdonik: [The Object-Oriented Database System Manifesto](#). in "Deductive and Object-Oriented Databases", Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), pp. 223-240

The Object-Oriented Database System Manifesto – Goodies

- Multiple inheritance
- Type checking and type inferencing
- Distribution
- Design transactions (Nested, Parallel)
- Versions (Schema Evolution)

plus there are a few
goodies - scientists are
allowed to have visions

feature list compiled from: [Atk+89] Malcolm P. Atkinson, François Bancilhon, David J. DeWitt, Klaus R. Dittrich, David Maier, Stanley B. Zdonik: [The Object-Oriented Database System Manifesto](#). in "Deductive and Object-Oriented Databases", Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), pp. 223-240

HOW

Overview

- what is persistence anyway?
 - persistence defined
 - the concept of transparent persistence
 - persistence interfaces
- application styles
 - when to use o/r mapping and when to use other options
- o/r mappers explained (**how to ...**) from the primitive to the complex
 - the basics of mapping
 - the basics of implementing o/r mapper features
 - oid, inheritance, relations, transactions,
- persistence in EJBs
- a few remarks on the state-of-the-art in .NET
- summary

Chapter Overview

Application Styles

- key messages
- a few very important attributes for starters
 - single user systems
 - serialization and flat file persistence
 - multi user systems with check-in // check-out persistence
 - again serialization and flat file persistence
 - number of objects - database size
- access patterns
 - object navigation versus record querying
- the list of forces a.k.a. nonfunctional requirements

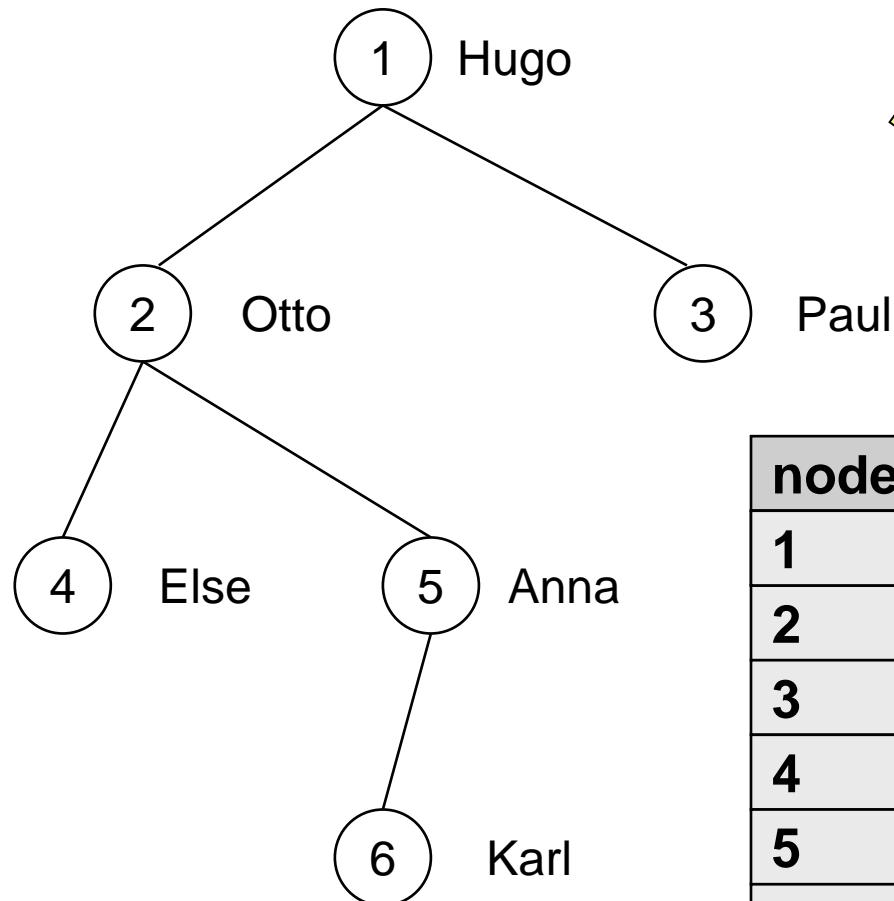
Application Styles

Key Messages



- avoid using relational databases for tree like and graph like data structures
- avoid using flat files or OODBs for business systems that would be normally implemented using a relational database
 - systems with a high level of concurrency
 - and with a large number of short transactions

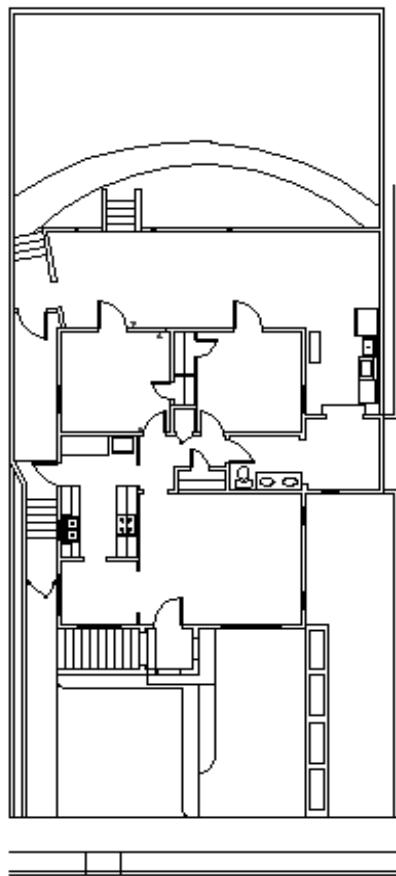
Please! No Tree Structures in a Relational Database!



navigating from Hugo
to Otto costs 3 select
statements

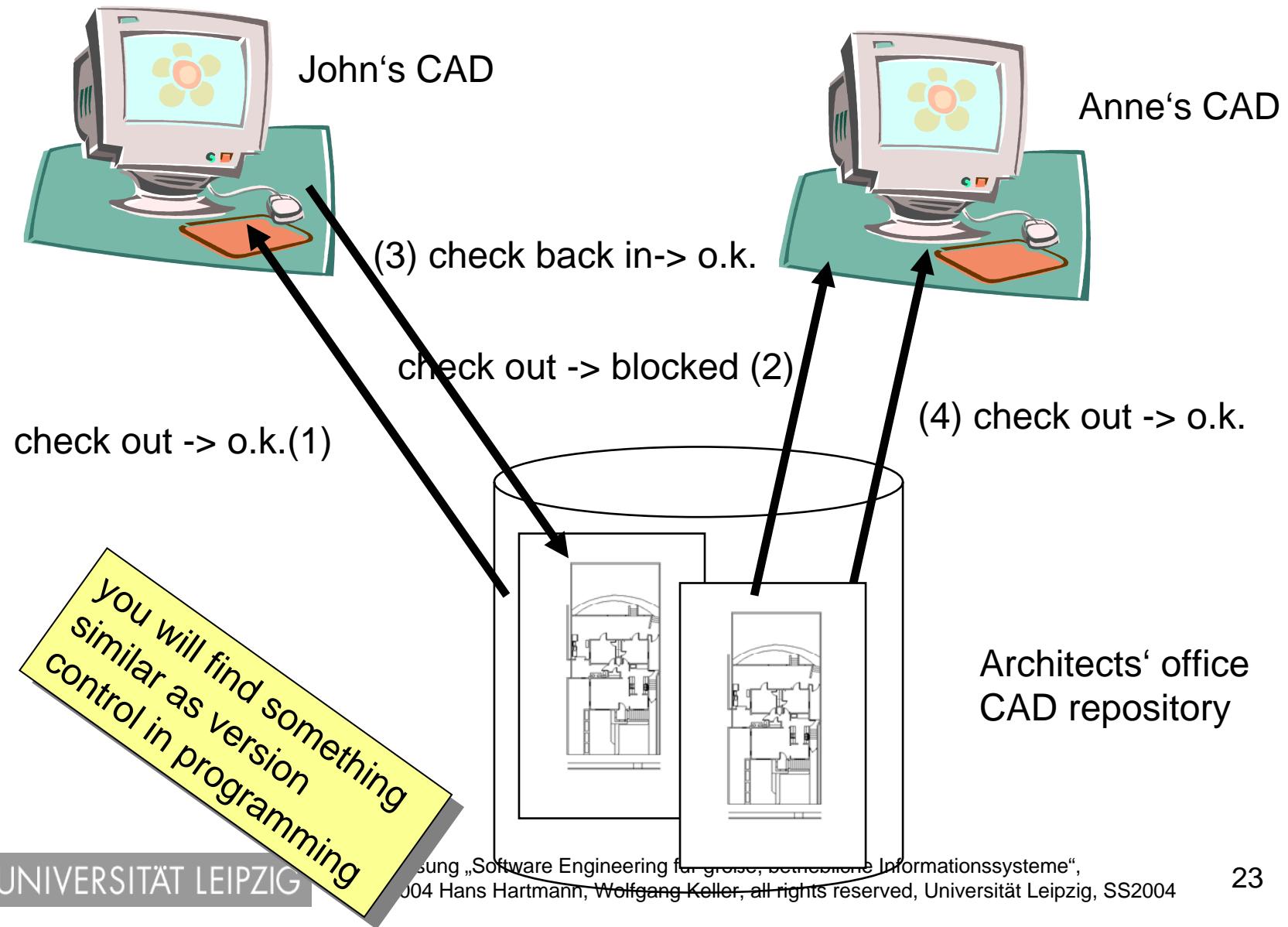
node id	parent	value
1	null	Hugo
2	1	Otto
3	1	Paul
4	2	Else
5	2	Anna
6	5	Karl

Single User System: Typical Example for a „Single User System“ - A User is editing a CAD Model of a Family Home



- CAD models and the like are typically represented by complex object models
- the size of low profile CAD models is in the mega bytes range
- it does not make too much sense if two people edit the same „small“ model at the same time
- serializing such a model into a flat file can do the job - often better than putting the model in a RDB

Now assume there are 7 Architects who do each a Piece of the Job



Size matters :-)

your computer's memory (1 Gig)

size of application data
(megs)

- flat file and check in / check out feasible

size of application data (terabytes)



the memory size of your PC
is peanuts compared to this

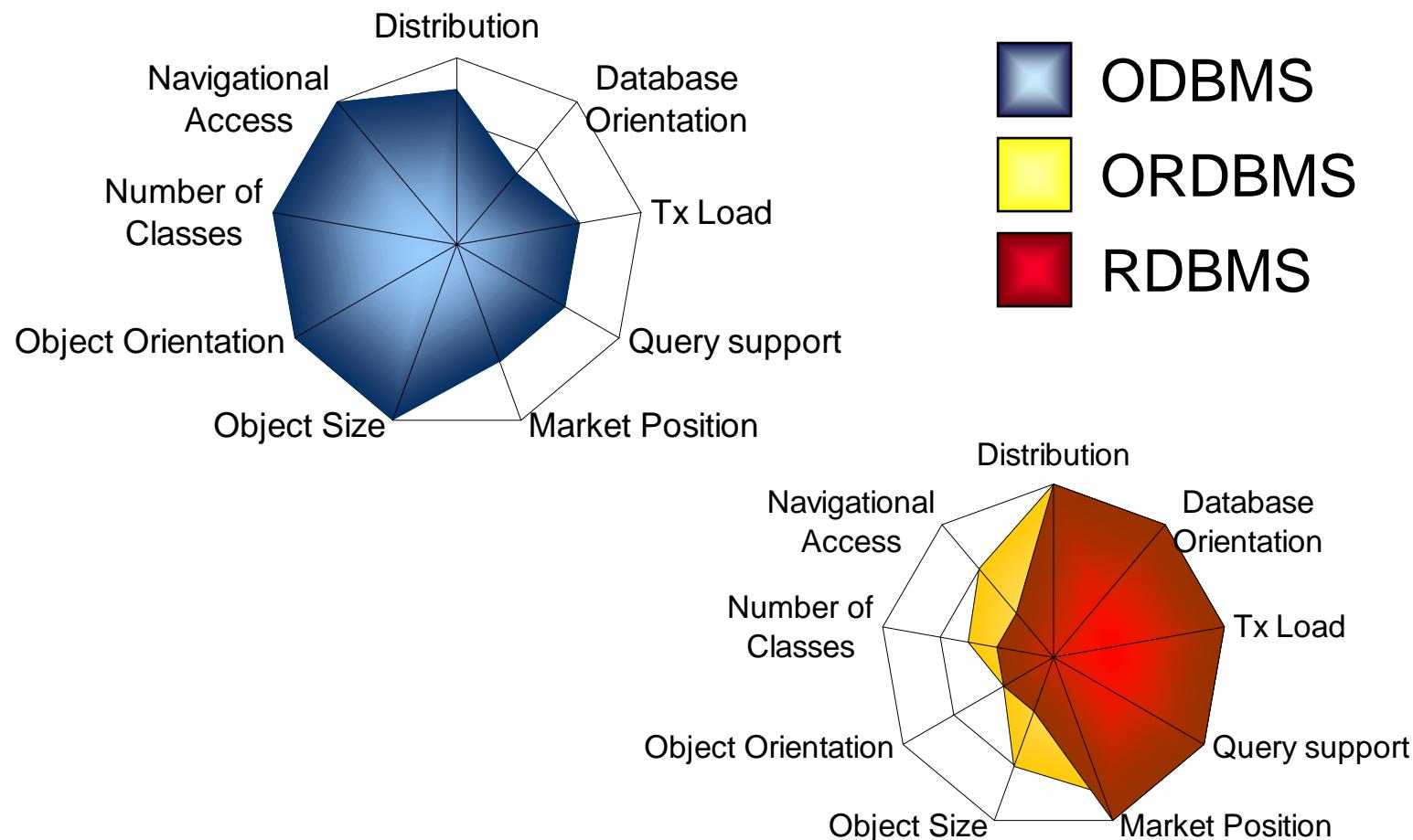
- PLEASE use a database



A few Rules of Thumb

- You have high concurrency - many users working the same data
 - use a „real“ database like an RDBMS or an OODBMS
- You need „true“ database features like recovery, logging, concurrency
 - use a relational or object database :-)
- Your amount of user data is several times larger than the working storage of your computer
 - use a relational or object database
- Your amount of user data is small compared to your computer size, concurrency is low to non-existent, the problem is a check in / check out problem
 - consider using stream persistence
- You build an Enterprise Information System like order entry, bookkeeping and the like
 - do what everybody does - use a RDBMS

For a more educated Decision have a look at Jens Coldewey's Tutorial on "Choosing Database Technology"
it's free!



Available at: <http://www.coldewey.com/publikationen/ChoosingDatabaseTech.pdf>

26



Bad news for the OODB community

They have ~0,6% Market Share

Category			Market Share (%) 2002	Market Share (%) 2001	Growth Rate (%) 2001-2002
	2002	2001			
Desktop DBMS	455	502	5	6	-9
Object DBMS	42	54	1	1	-22
Pre-relational DBMS	1,225	1,228	15	14	0
RDBMS	6,642	7,139	79	80	-7
Total DBMS	8,363	8,922	100	100	-6

Worldwide Database Revenue by Category for 2000-2001 Based on
New License Sales (Millions of Dollars).
Source: Gartner Dataquest (June 2003)

HOW

Overview

- what is persistence anyway?
 - persistence defined
 - the concept of transparent persistence
 - persistence interfaces
- application styles
 - when to use o/r mapping and when to use other options
- o/r mappers explained (**how to ...**) from the primitive to the complex
 - the basics of mapping
 - the basics of implementing o/r mapper features
 - oid, inheritance, relations, transactions,
- persistence in EJBs
- a few remarks on the state-of-the-art in .NET
- summary

Chapter Overview (1/2)

O/R Mappers ... from the Primitive to the Complex

- mapping „straight“ objects
 - 1 class => 1 table
- the CRUD pattern
 - create, read, update, write
- object identity and the identity cache
- 1:n relations and lazy loading
- persistence „without programming“: exploiting meta information

Chapter Overview (2/2)

O/R Mappers ... from the Primitive to the Complex

- mapping inheritance, polymorphism
- transactions

Key Messages

- even if I tell you ho to build one ☺
don't build o/r mappers - buy them
 - unless you are a open source or commercial developer of such layers or in the research business
- avoid using implementation inheritance, multiple inheritance, polymorphic queries and other O-O gadgets for plain old domain objects in business systems.
 - This is only rarely needed and it only costs
 - O/R mappers get over complicated

Mapping „straight“ Objects

```
class Gangster
  string name
  string nick_name
  int badness
```



Each field is mapped to a database column
this looks simple, but

```
SQL> desc gangster
Name                           Null?    Type
-----                         -----
NAME                          NOT NULL VARCHAR2( 255 )
NICK_NAME                     VARCHAR2( 64 )
BADNESS                        NOT NULL NUMBER( 10 )
```

Source: Idea from the JBoss Crime Portal Tutorial
<http://rzm-hamy-wsx.rz.uni-karlsruhe.de/Training/JBoss-3.0/html-generated/crimeportal.html>

Mapping „straight“ Objects but ... The Devil is in the Details

- How do you map variable length data types like e.g. strings?
 - lots of VARCHARS do not speed up a database
- enums need special treatment
- SQL data types are not semantically identical with e.g. Java data types
- how do you deal with aggregated complex types like e.g. records?
 - Put them in an extra table (slower)
 - unfold them into various tables (tougher to maintain)
- Different mapper vendors have different answers

Mapping Straight Objects

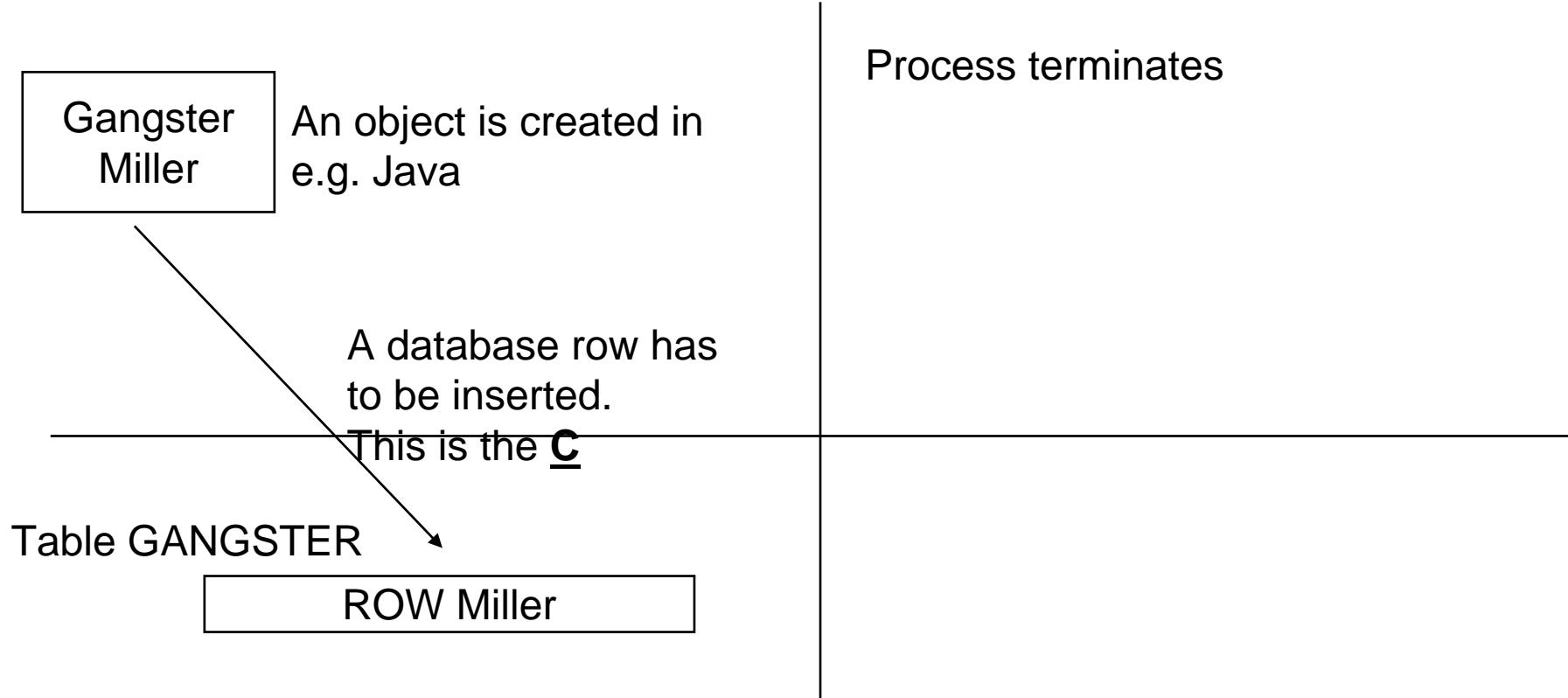
You need an Identifier (id, a.k.a. oid)

```
Gangster bigBoss;  
Gangster capo;  
Gangster arrestHim;  
  
capo = new Gangster( „Miller” , „the Killer” , 9 );  
bigBoss = new Gangster( „Miller” , „the Smart” , 13 );  
  
arrestHim = Gangster.getByName( „Miller” ); // ????
```

- In O-O languages instances are implicitly identified
 - duplicate „keys“ are no problems
- if you want to find something in a database you need a unique key
- therefore in most cases a synthetic object id (OID) is added to the „pure“ domain model

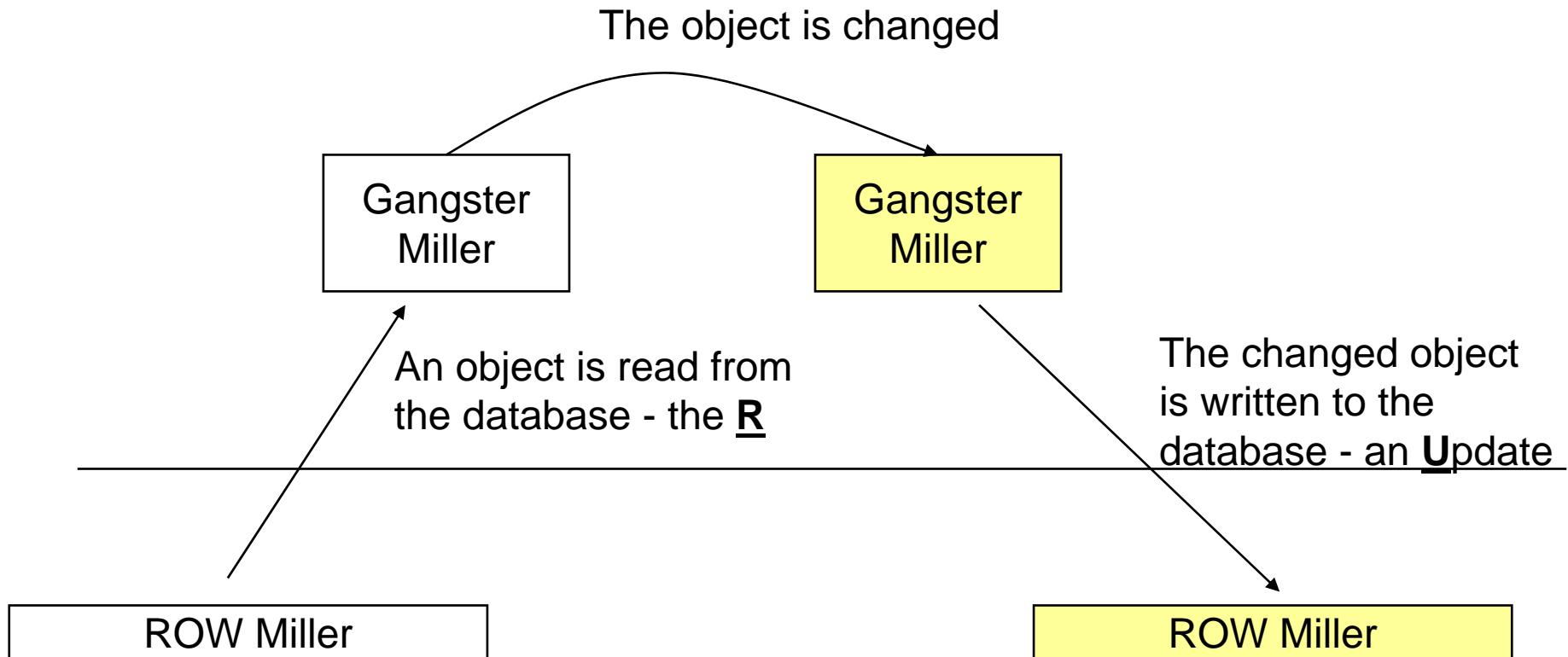
The CRUD pattern (1)

Or how are objects moved up and down
between the database and object space



The CRUD pattern (2)

Or how are objects moved up and down
between the database and object space



Read, Update (Write) Pattern

Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keiler, all rights reserved, Universität Leipzig, SS2004

Sample: The code that updates a Gangster EJB using Bean Managed Persistence (BMP)

Backup

```
private void storeRow() throws SQLException {
    String updateStatement =
        "update GANGSTER set NAME = ? , " +
        "NICK_NAME = ? , BADNESS = ? " +
        "where OID = ?";
    PreparedStatement prepStmt =
        con.prepareStatement(updateStatement);

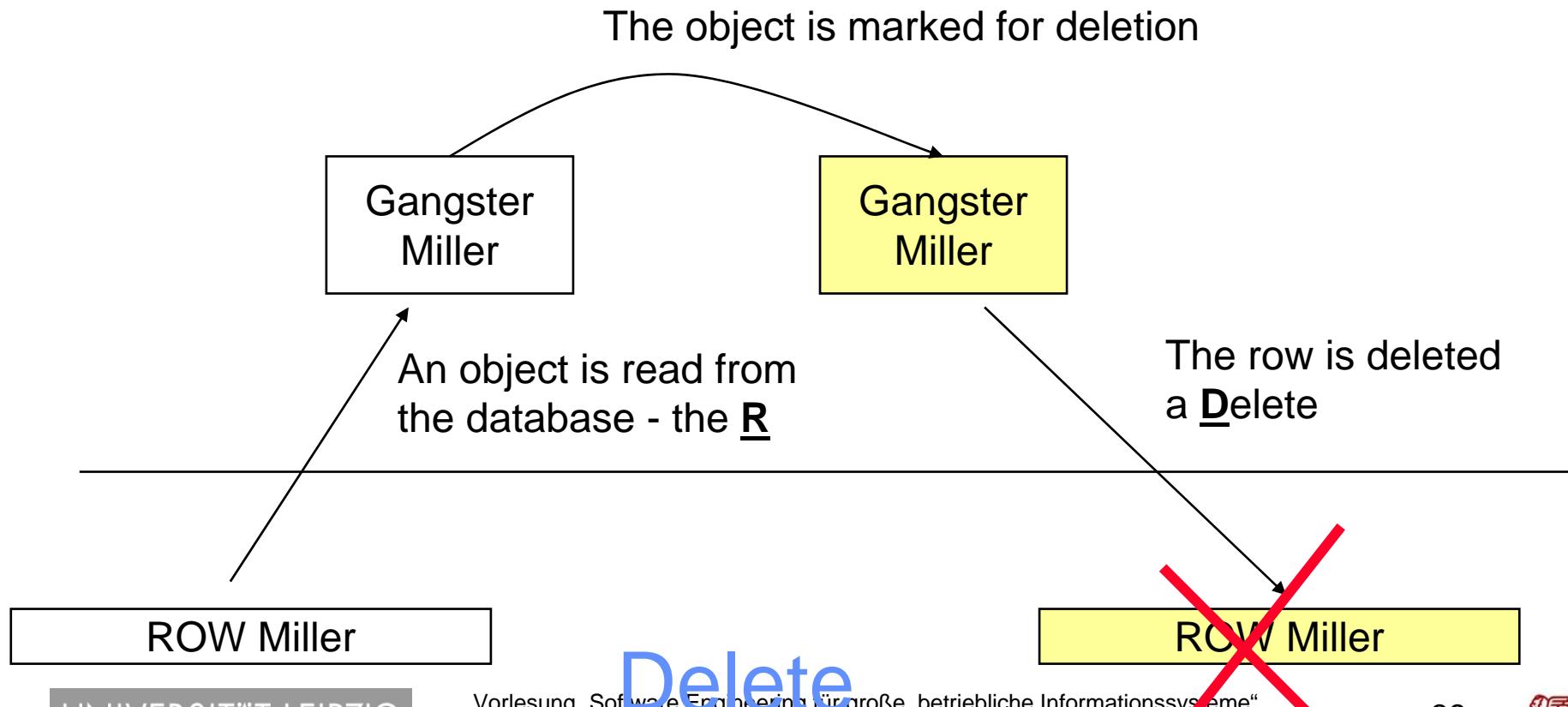
    prepStmt.setString(1, name);
    prepStmt.setString(2, nick_name);
    prepStmt.setDouble(3, badness);
    prepStmt.setString(4, oid);
    int rowCount = prepStmt.executeUpdate();
    prepStmt.close();

    if (rowCount == 0) {
        throw new EJBException("Storing row for id " + oid + " failed.");
    }
}
```

The CRUD pattern (3)

Or how are objects moved up and down between the database and object space

```
deleteHim = Gangster.getOID(1234567);  
deleteHim.markDeleted();
```



Where do you find the CRUD methods? There are several variants

- You can find the methods as an addendum in the domain classes
 - this is straight forward, but not considered very elegant
 - application of the „Multilayer Class“ Pattern*
- you can find the methods in an additional „data container object“**
 - results in a better separation of layers
- you can find code that generates the methods at runtime exploiting meta-information using e.g. the Reflection API in Java, or reflection if you work in Smalltalk
 - browse the the „Reflective CRUD pattern“***

* find it at <http://www.objectarchitects.de/arcus/publicat/multilay.ps.gz>

** see „Row Data Gateway“ in Fowler’s Patterns of Enterprise Apps

© *** <http://www.inf-cr.uclm.es/www/mpolo/yet/>

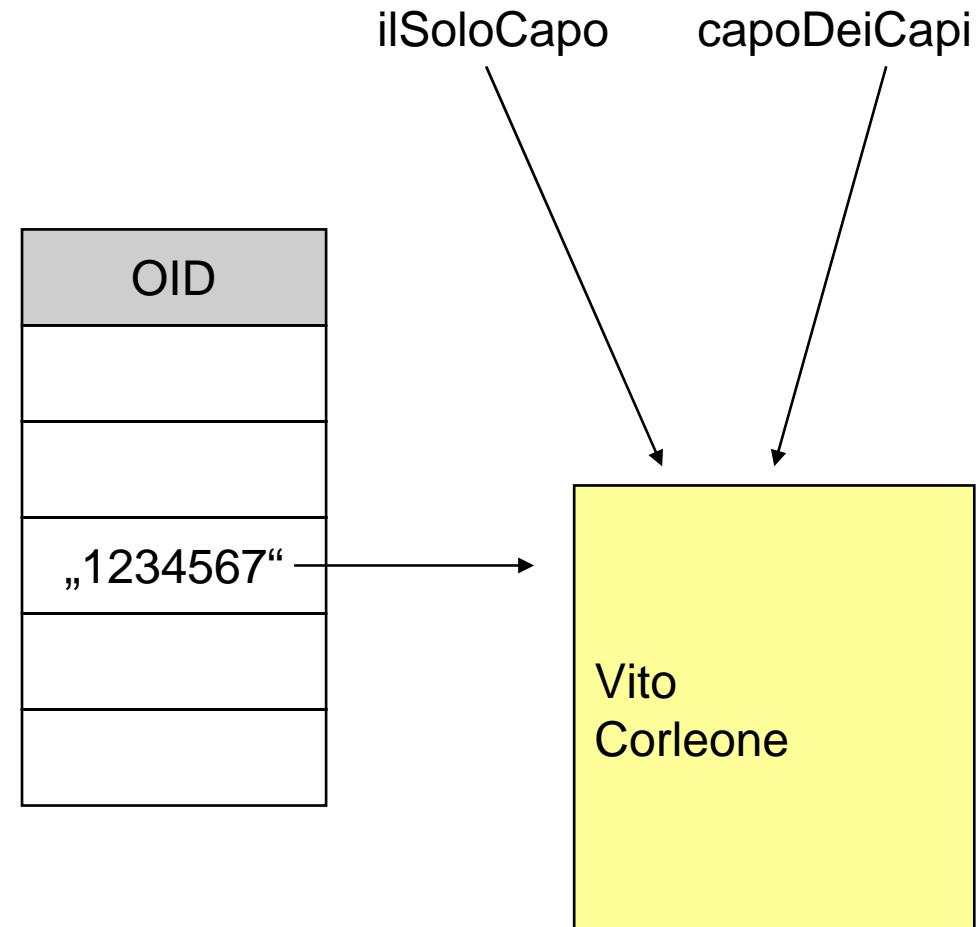
Object Identity and the Identity Cache

```
Gangster capoDeiCapi;  
Gangster ilSoloCapo;  
  
capoDeiCapi = Gangster.getByName( „Corleone“ , „Vito“ ); // (1)  
ilSoloCapo = Gangster.getByName( „Corleone“ , „Vito“ ); // (2)  
  
capoDeiCapi.setBadness(MAXBADNESS);  
if (MAXBADNESS != ilSoloCapo.getBadness()) {  
    // Palermo!!! - we've got a problem  
};
```

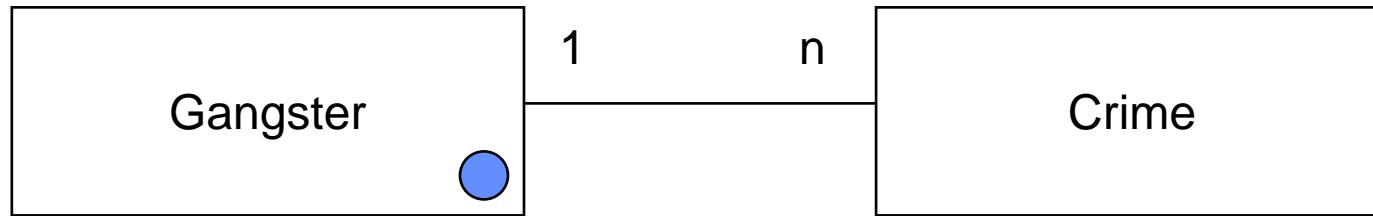
- the same object should be read once and only once from the database in one transaction
- there needs to be a mechanism to guarantee this
- and the mechanism should be hidden from the programmer using the persistence layer

The Identity Cache can be implemented as a Hash Table with an Entry per Loaded Object

- The first `getByName()` will return the „Vito Corleone“ object
- the second `getByName()` may try to register the „Vito Corleone“ object. It will get a handle to the object already registered and will return it
- There are various versions how to implement this in detail .. But with similar semantics.



1:n Relations and Lazy Loading



```
public class Gangster {  
    ....  
    private Set crimes  
    ....  
  
    public void getAllCrimes (Set aCrimeSet) {  
        // lazy load goes here ...  
    };  
}
```

● Gangster-Row

● Crime1-Row

● Crime2-Row

● Crime3-Row

● OID, Primary Key

UNIVERSITÄT LEIPZIG

1:n Relations and Lazy Loading

- **Problem:** If you load a Gangster (getByName) you don't want to automatically load all crimes (fill the set)
 - **therefore** you don't fill the Set of Crimes when you load a gangster but implement a special „lazy Set“ that will not load anything before it is actually accessed by a getter-method
-
- This is known as „lazy loading“*, and various variants of „smart pointers“
 - There are variants of this mechanism depending on the persistence API you use - but most layers use it

Persistence „without programming“ - Exploiting Meta Information

```
private void storeRow() throws SQLException {
    String updateStatement =
        "update GANGSTER set NAME = ? , " +
        "NICK_NAME = ? , BADNESS = ? " +
        "where OID = ?";
    PreparedStatement prepStmt =
        con.prepareStatement(updateStatement);

    prepStmt.setString(1, name);
    prepStmt.setString(2, nick_name);
    prepStmt.setDouble(3, badness);
    prepStmt.setString(4, oid);
    int rowCount = prepStmt.executeUpdate();
    prepStmt.close();

    if (rowCount == 0) {
        throw new EJBException("Storing row for id " + oid + " failed.");
    }
}
```

Observation: code like **this** need not be written by hand

- it can be generated by a preprocessor
- or it can also be generated at run time using e.g. the Java Reflection API

Exploiting Meta Information

Different mappers use different approaches

- JDO uses a so called „Class Enhancer“ which „pre“processes Java .class files
 - the enhancements are not direct SQL code but calls to a persistence manager – but JDO would be another talk
 - see e.g. www.jdocentral.com
- EJB-CMP uses a lot of user provided meta-information
 - code generation at build time
 - see documentation of EJB containers - e.g. www.jboss.org
- Reflective CRUD exploits the Reflection API
 - code generation at run-time
 - see <http://www.inf-cr.uclm.es/www/mpolo/yet/>

Mapping simple Inheritance - Variant 1 One Inheritance Tree // One Table*

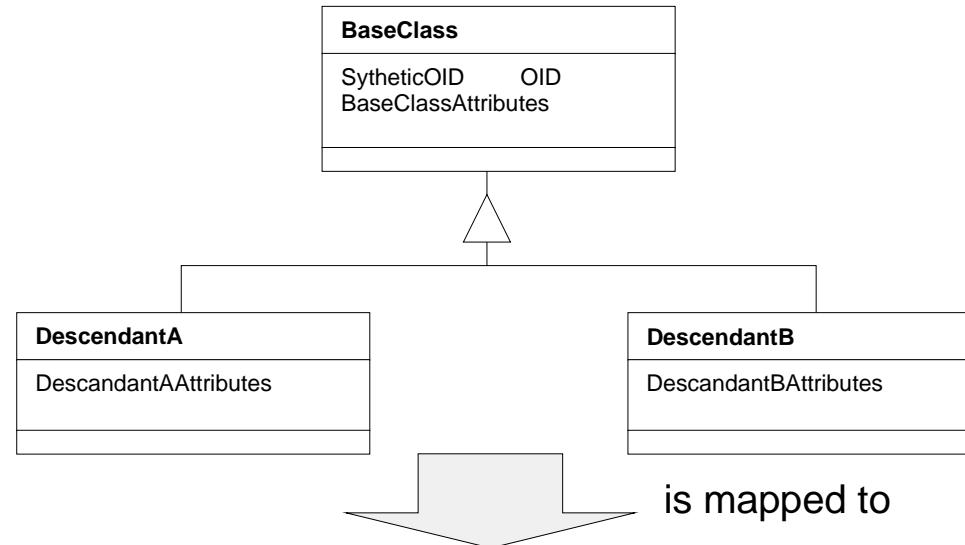


Table for BaseClass, DescendantA, DescendantB			
	SytheticOID, BaseClassAttributes	DescendantAAttributes	DescendantBAttributes
BaseClassInstance	Attribute Values	Null Values	Null Values
DescendantA Instance	Attribute Values	Attribute Values	Null Values
DescendantB Instance	Attribute Values	Null Values	Attribute Values

* if you're interested in table mappings there's a complete paper for free at
http://www.objectarchitects.de/ObjectArchitects/orpatterns/MappingObjects2Tables/mapping_object.htm

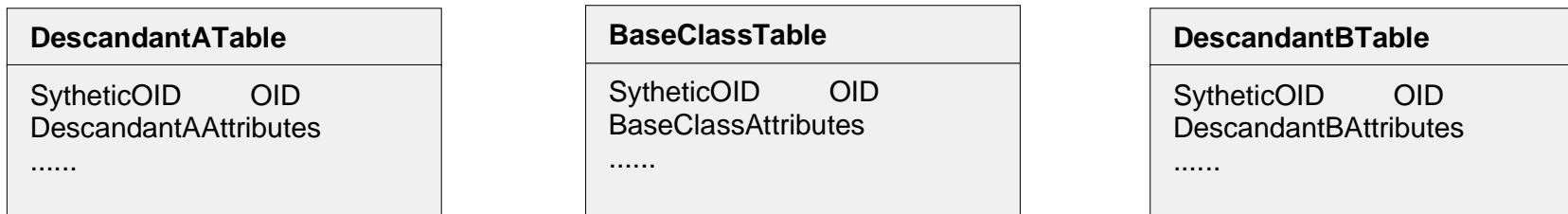
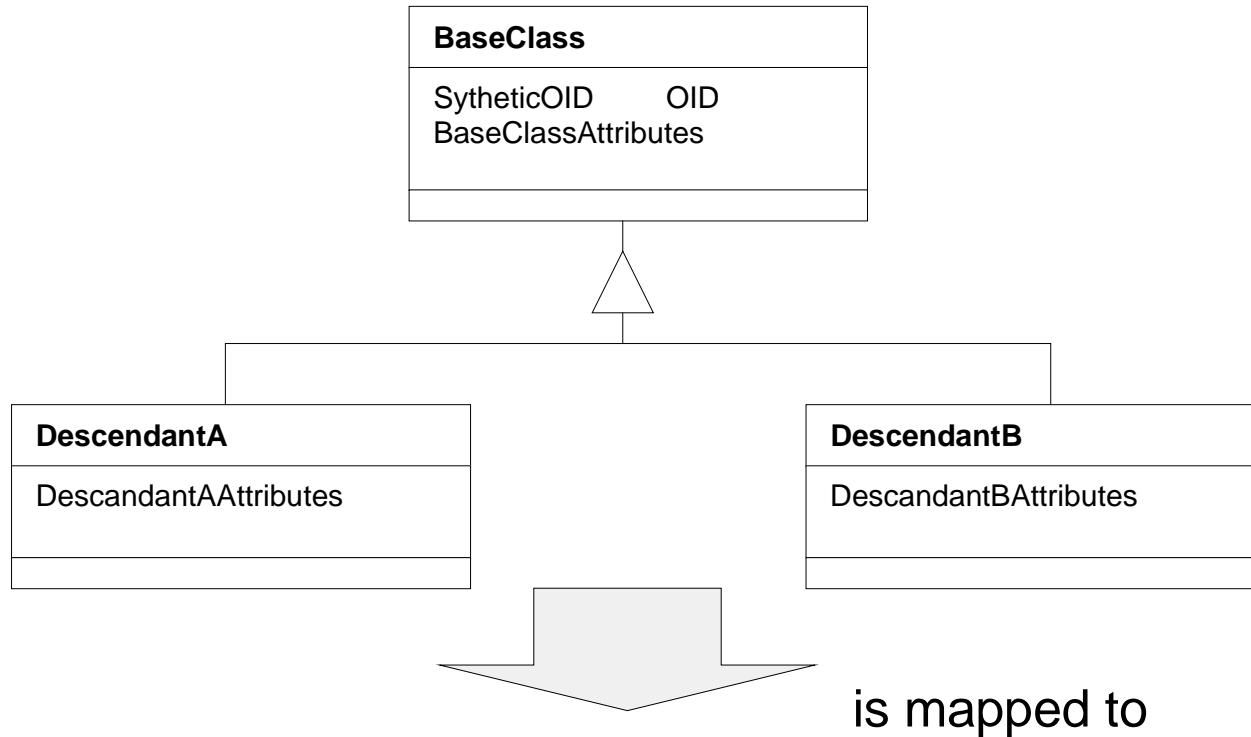
2004

46



Mapping simple Inheritance - Variant 2

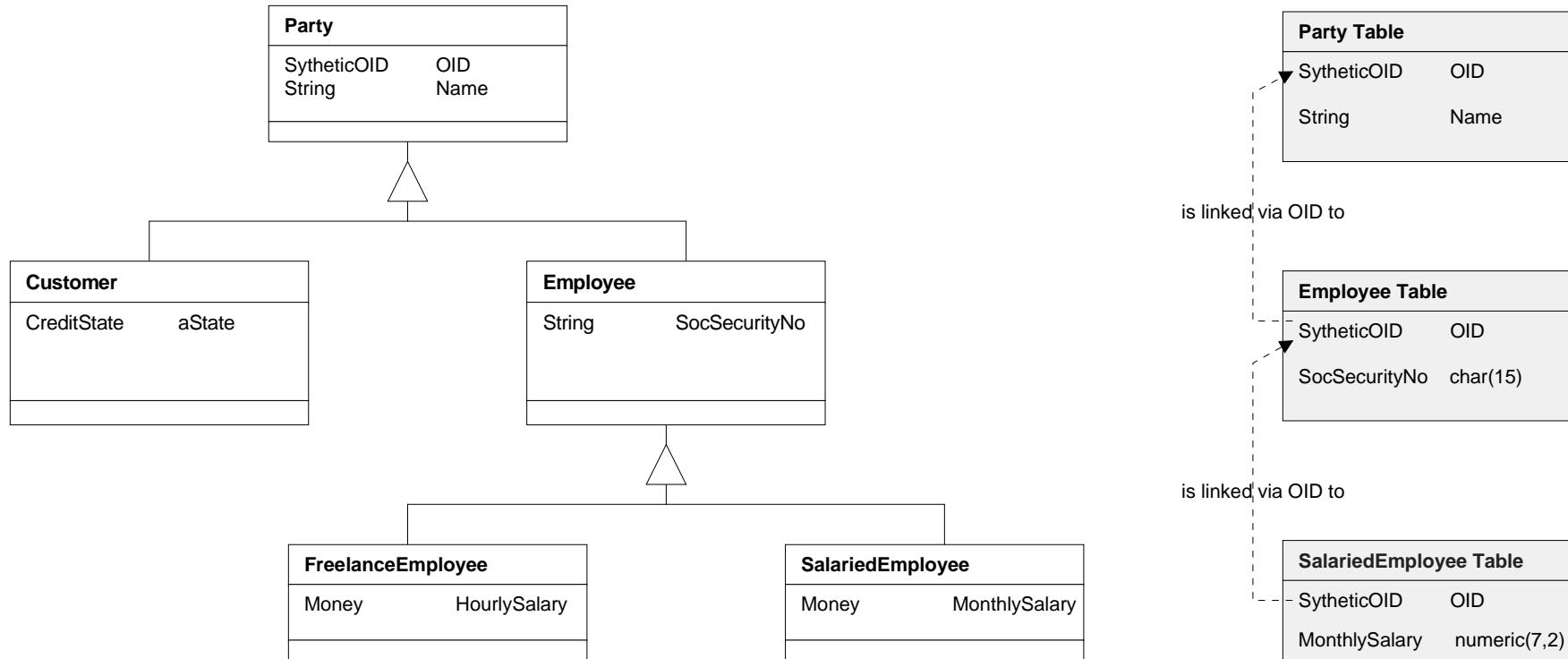
One Class // One Table



* if you're interested in table mappings there's a complete paper for free at
http://www.objectarchitects.de/ObjectArchitects/orpatterns/MappingObjects2Tables/mapping_object.htm

Mapping simple Inheritance - Variant 2

The main drawback is evident - Performance



Constructing one object instance needs up to 3 selects

Polymorphic queries potentially need to visit all tables

Mapping - Summary

- there are various aspects that need to be taken into account (see table below // forces)

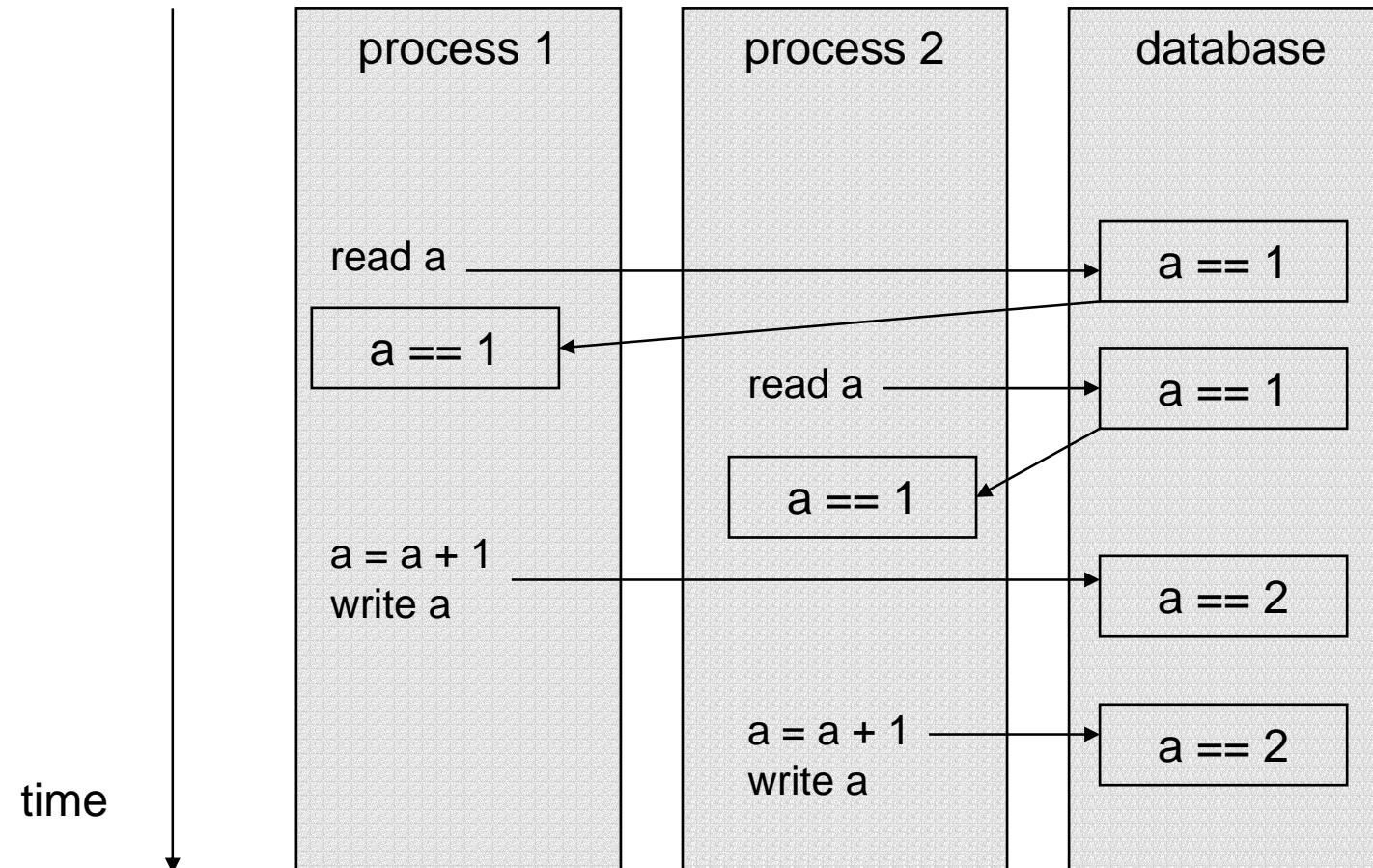
Pattern	Performance			Space Consumption	Flexibility, Maintainability	Ad-hoc Queries
	Write/ Update	Single Read	Polymorphic Queries			
Single Table Aggregation	+	+	*	+	-	-
Foreign Key Aggregation	-	-	*	+	+	+
One Inheritance Tree One Table	+o	+o	+	-	+	+
One Class One Table	-	-	-o	+	+	-
One Inheritance Path One Table	+	+	-	+	-	-
Objects in BLOBs	+o	+o	o	+	-	-
Foreign Key Association	-	o	*	+	+	+
Association Table	-	o	*	+	+	+

+ good, - poor, * irrelevant, o depends, see detailed discussion

- there's ample and also free literature on the web
 - http://www.objectarchitects.de/ObjectArchitects/orpatterns/mapping_patterns_for_free
 - or get it from Scott Ambler at <http://www.agiledata.org/essays/mappingObjects.html>
 - Martin Fowler's „Patterns of Enterprise Application Architecture“ has the same in a book

Transactions and Locking

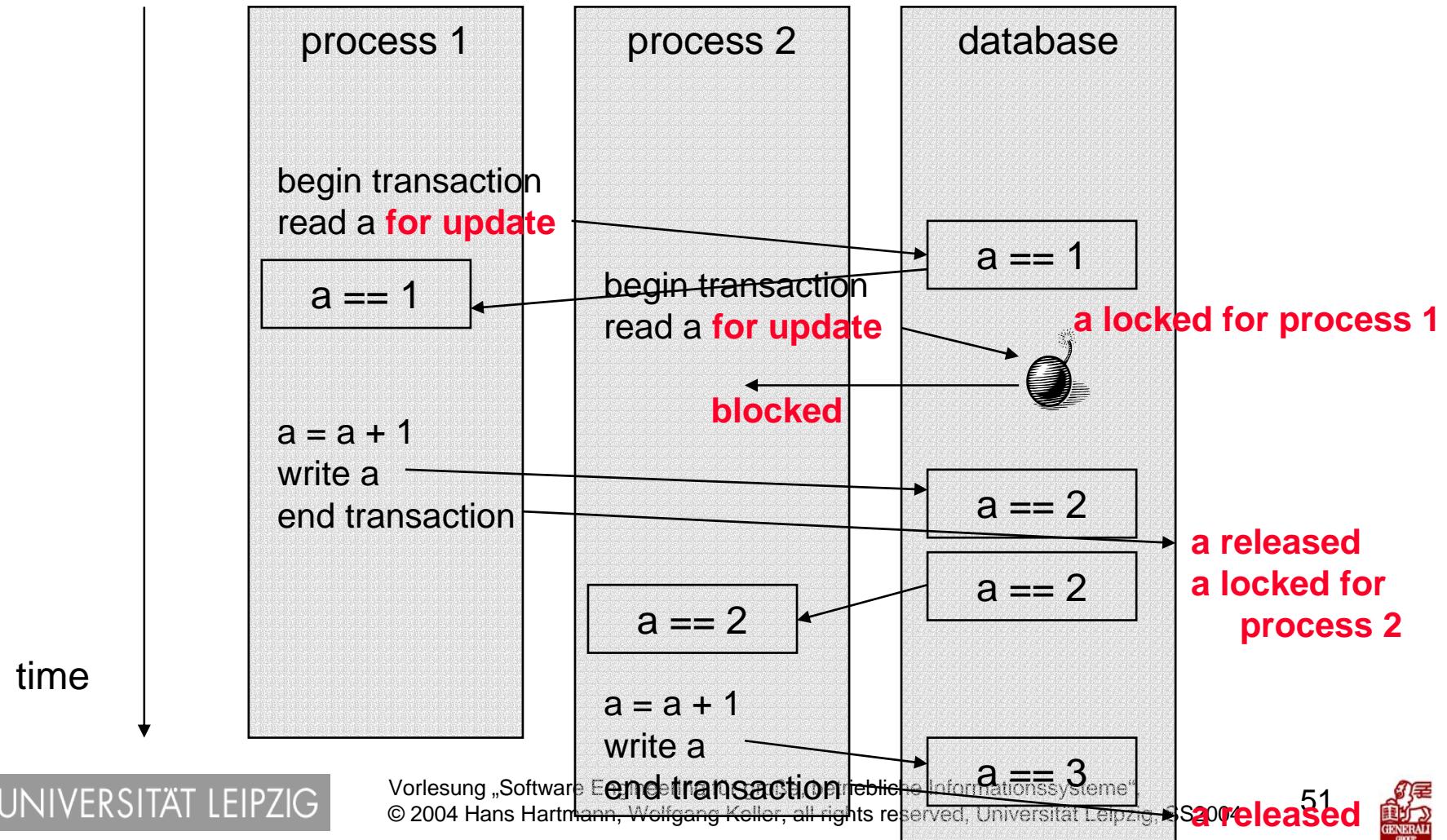
The Lost Update Problem or $2 + 1 + 1$ may end up to be 3



Transactions

Same Use Case

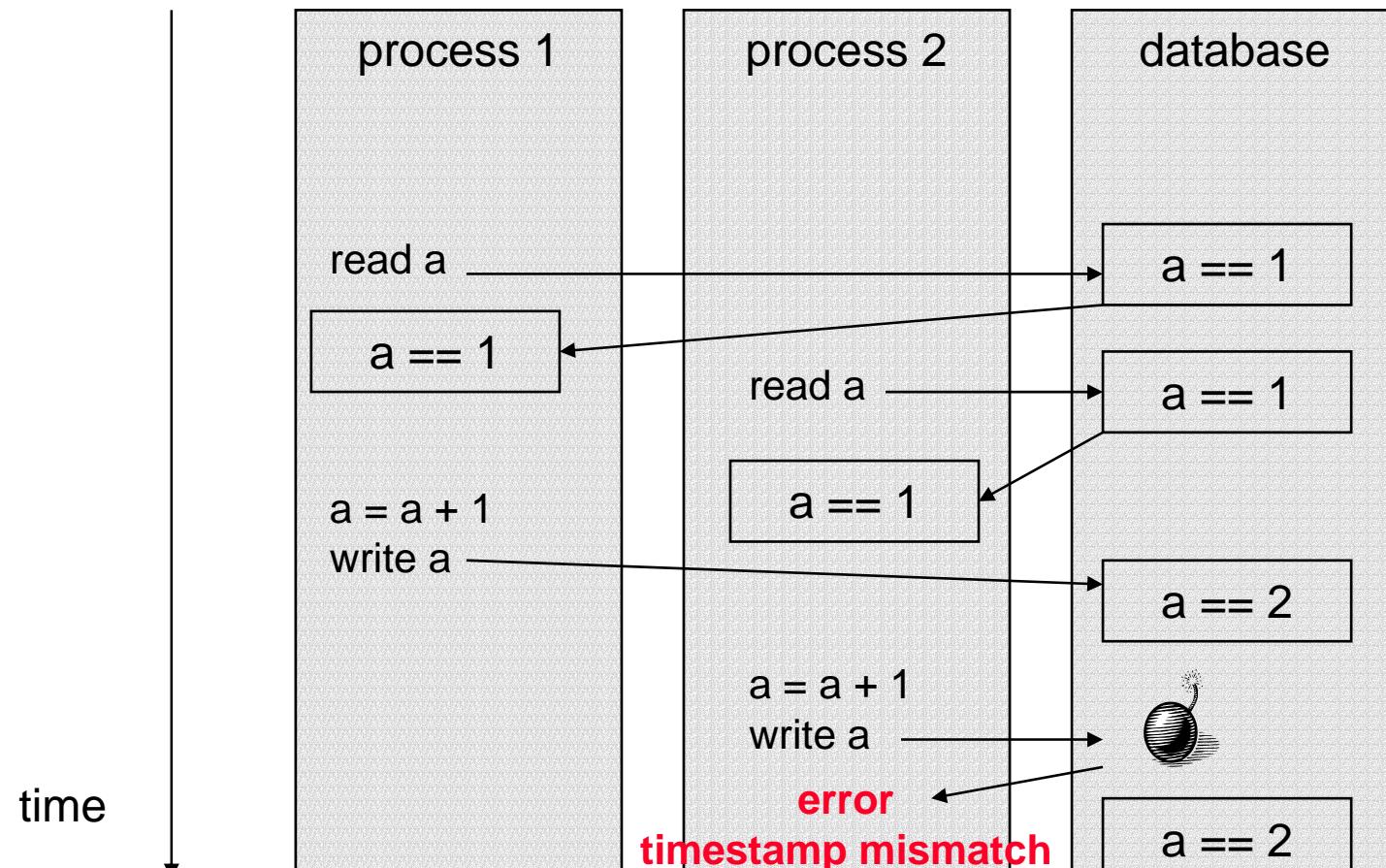
Use of Pessimistic Locking



Transactions

Same Use Case

Use of Optimistic Locking



Pessimistic vs. Optimistic Locking

- pessimistic locking is the standard way to prevent the lost update problem provided by relational databases
 - it is appropriate if the application above guarantees that locks are held only for short periods of time
 - it is fatal if somebody holds locks for minutes - even if somebody holds locks for seconds, it pulls down performance
 - pessimistic locking can lead to deadlock situations - which are resolved by the database manager which rolls back transactions
- optimistic locking is offered by most o/r access layers
 - timestamp errors need to be handled by the application - the reaction is domain specific
 - it is appropriate for long transactions
 - with a low likelihood of collisions

Implementing Optimistic Locking

- you need a special transaction object that knows which objects need to be written to a database
 - objects need to be registered with this transaction object
 - this can happen automatically „under the surface“
- you need to add a timestamp field in each table in the physical database design
 - this will also be read upon read
 - and will be compared when updating
- still needs to be based on pessimistic locks, while changes are written
- as people have thought of all this before, e.g. JDO offers this mechanism as a ready to use option

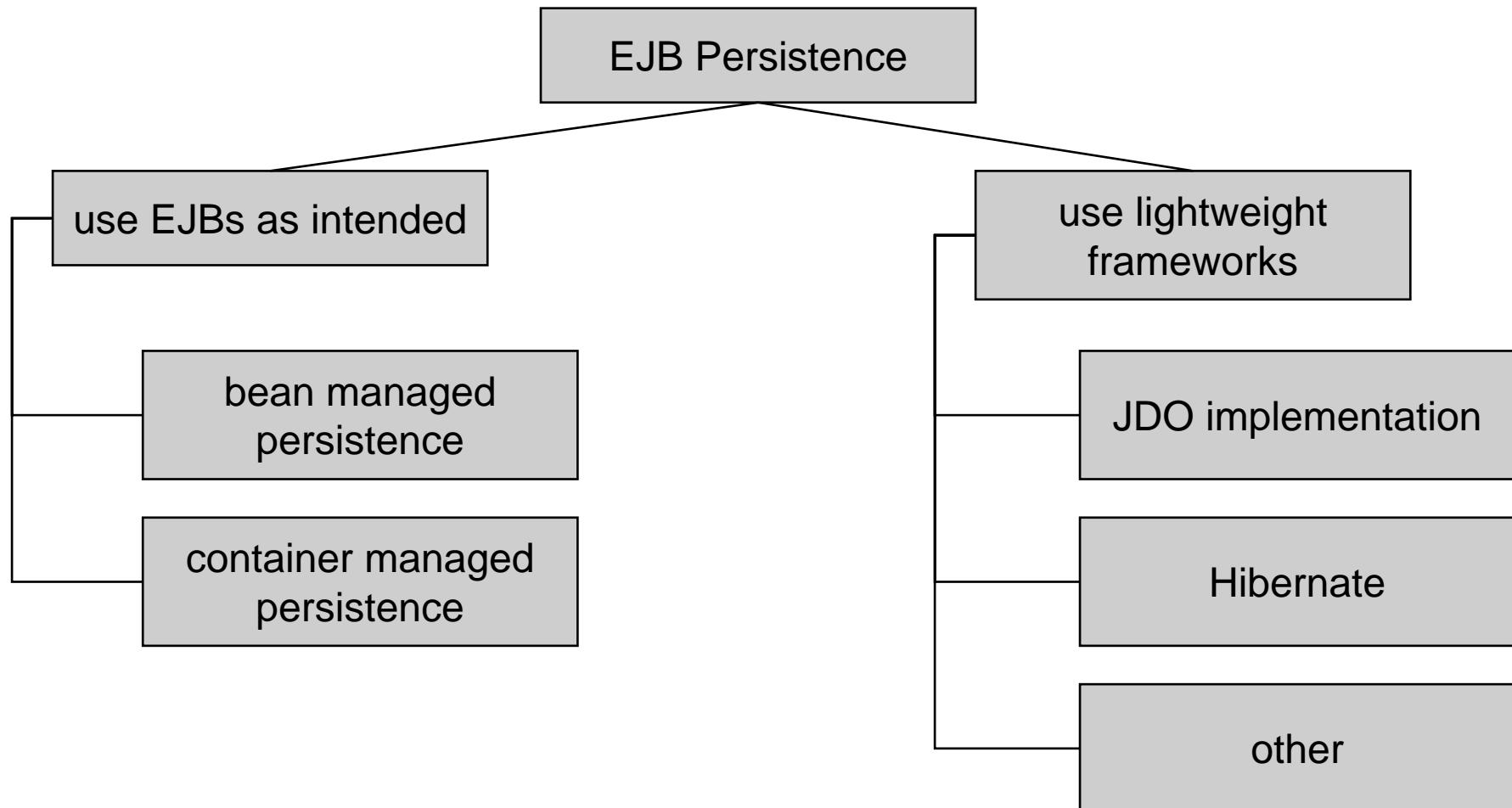
HOW

Overview

- what is persistence anyway?
 - persistence defined
 - the concept of transparent persistence
 - persistence interfaces
- application styles
 - when to use o/r mapping and when to use other options
- o/r mappers explained (**how to ...**) from the primitive to the complex
 - the basics of mapping
 - the basics of implementing o/r mapper features
 - oid, inheritance, relations, transactions,
- persistence in EJBs
- a few remarks on the state-of-the-art in .NET
- summary

Persistence in EJBs

A short outline ...



Persistence in EJBs

A short outline ...

- First of all EJB containers distinguish between
 - Session Beans
 - these are „transient“
 - Entity Beans
 - these are the potentially persistent objects that store business objects
- For Entity Beans there are two persistence mechanisms
 - **Bean Managed Persistence (BMP)**
 - this is „write your own layer“ persistence
 - might be used in complex mapping cases or performance critical apps
 - **Container Managed Persistence (CMP)**
 - automatic, using normed protocols
 - vendor specific
 - need not be mapped to a database - may be mapped to a database
 - Clear Advantage: Query Language

seen that before

CMP: Pros and Cons

- Pro:** Query Language with the power of OQL comes for free
- Pro:** For simple, straight cases you need not know too much about access layers
- Caveat:** Have a look at how your container vendor implements CMP.
He might implement it without a database just based on indexed files. The standard does alone does not prevent it.
- Con:** Lots and lots of deployment information needs to be written
- Con:** Does not cover really complex data type to database mapping (vendor specific)

For a detailed comparison with e.g. JDO - another mapping layer see
backup slides

Persistence with EJB Containers

You find the following main options

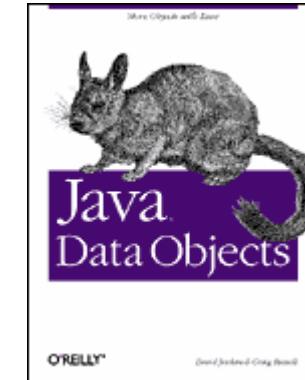
- Use, what the container provides
 - CMP and BMP
- Lately people use a „lighter“ weight persistence framework and skip the Entity Beans
 - either Hibernate
 - or a JDO implementation
 - or anything else from the market

EJB-Container Managed Persistence (1) versus for example JDO

the table can be found at www.jdocentral.com or in David Jordan's and Craig Russel's highly recommendable book on JDO

{PRIVATE}Character	CMP beans	JDO persistent classes
Environmental		
Portability of applications	Few portability unknowns	Documented portability rules
Operating environment	Application server	One-tier, two-tier, web server, application server
Independence of persistent classes from environment	Low: beans must implement EJB interfaces and execute in server container	High: persistent classes are usable with no special interface requirements and execute in many environments
Metadata		
Mark persistent classes	Deployment descriptor identifies all persistent classes	Metadata identifies all persistent classes
Mark persistent fields	Deployment descriptor identifies all persistent fields and relationships	Metadata defaults persistent fields and relationships
Modeling		
Domain-class modeling object	CMP bean (abstract schema)	Persistent class
Inheritance of domain-class modeling objects	Not supported	Fully supported
Field access	Abstract get/set methods	Any valid field access, including get/set methods
Collection, Set	Supported	Supported
List, Array, Map	Not supported	Optional features
Relationships	Expressed as references to CMP local interfaces	Expressed as references to JDO persistent classes or interfaces
Polymorphic references	Not supported	Supported

see:



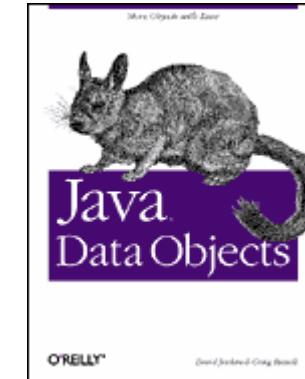
Backup

EJB-Container Managed Persistence (2) versus for example JDO

the table can be found at www.jdocentral.com or in David Jordan's and Craig Russel's highly recommendable book on JDO

{PRIVATE}Characteri	CMP beans	JDO persistent classes
Programming		
Query language	EJBQL modeled after SQL	JDOQL modeled after Java Boolean expressions
Remote method invocation	Supported	Not supported
Required lifecycle methods	<code>setEntityContext</code> , <code>unsetEntityContext</code> , <code>ejbActivate</code> , <code>ejbPassivate</code> , <code>ejbLoad</code> , <code>ejbStore</code> , <code>ejbRemove</code>	no-arg constructor (may be private)
Optional lifecycle callback methods	<code>ejbCreate</code> , <code>ejbPostCreate</code> , <code>ejbFind</code>	<code>jdoPostLoad</code> , <code>jdoPreStore</code> , <code>jdoPreClear</code> , <code>jdoPreDelete</code>
Mapping to relational datastores	Vendor-specific	Vendor-specific
Method security policy	Supported	Not supported
Method transaction policy	Supported	Not supported
Nontransactional access	Not standard	Supported
Required classes/interfaces	<code>EJBLocalHome</code> , local interface (if local interface supported); <code>EJBHome</code> , remote interface (if remote interface supported); Abstract beans must implement <code>EJBEntityBean</code> ; Identity class (if nonprimitive identity)	Persistent class; <code>objectid</code> class (only for application identity)
Transaction synchronization callbacks	Not supported	Supported

see:



Backup

HOW

Overview

- what is persistence anyway?
 - persistence defined
 - the concept of transparent persistence
 - persistence interfaces
- application styles
 - when to use o/r mapping and when to use other options
- o/r mappers explained (**how to ...**) from the primitive to the complex
 - the basics of mapping
 - the basics of implementing o/r mapper features
 - oid, inheritance, relations, transactions,
- persistence in EJBs
- a few remarks on the state-of-the-art in .NET
- summary

a very short visit of persistence in .NET

let's assume C#

- products have appeared somewhat later than similar products in the Java community ...
- There are two levels ...
 - database APIs similar to what JDBC offers .. in .NET speech the equivalent is called ADO.NET
 - full blown persistence layers similar to the one's we have seen so far are there but in smaller numbers. Some are also „descendants“ from a previous Java version.

ADO.NET, C# code sample

from <http://samples.gotdotnet.com/quickstart/howto/doc/adoplus/updatedatafromdb.aspx>

```
// Create a new Connection and SqlDataAdapter

SqlConnection myConnection = new
    SqlConnection("server=(local)\\VSdotNET;Trusted_Connection=yes;database=northwind");
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter("Select * from Customers",
    myConnection);
DataSet myDataSet = new DataSet();
DataRow myDataRow;

// Create command builder. This line automatically generates
// the update commands for you, so you don't
// have to provide or create your own.
SqlCommandBuilder mySqlCommandBuilder = new SqlCommandBuilder(mySqlDataAdapter);

// Set the MissingSchemaAction property to AddWithKey because Fill will not cause primary
// key & unique key information to be retrieved unless AddWithKey is specified.
mySqlDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

mySqlDataAdapter.Fill(myDataSet, "Customers");

myDataRow = myDataSet.Tables["Customers"].NewRow();
myDataRow["CustomerId"] = "NewID";
myDataRow["ContactName"] = "New Name";
myDataRow["CompanyName"] = "New Company Name";

myDataSet.Tables["Customers"].Rows.Add(myDataRow);
```

Backup

a few places where you can find information on .NET persistence

- some relevant groups and places
 - usenet: microsoft.public.objectspaces
 - <http://groups.msn.com/DotNetPersistence>
 - <http://dotnetguru.org/> section on persistence
- full blown persistence products: just some examples ...
 - Pragmatier for VB.NET and C#
<http://www.pragmatier.com>
 - FastObjects for C#
<http://www.fastobjects.com/>
 - db4o
<http://www.db4o.com>

HOW

Overview

- what is persistence anyway?
 - persistence defined
 - the concept of transparent persistence
 - persistence interfaces
- application styles
 - when to use o/r mapping and when to use other options
- o/r mappers explained (**how to ...**) from the primitive to the complex
 - the basics of mapping
 - the basics of implementing o/r mapper features
 - oid, inheritance, relations, transactions,
- persistence in EJBs
- a few remarks on the state-of-the-art in .NET
- summary

Summary

The 4 Key Messages revisited

Sooooow:-)

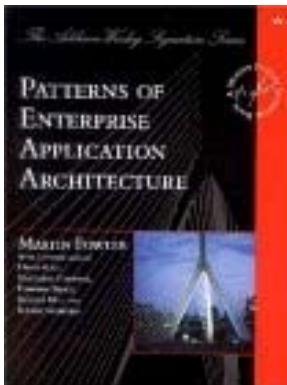
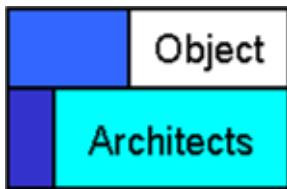
- know your application style before you decide for a certain way to implement persistence
- know the concept of transparent persistence
- don't develop your own green-field persistence layer unless you do it for fun. That made sense 10 years ago but in the presence of plenty of commercial and open source software for the area it is nowadays too expensive in most cases
- In case you run into problems, know where to find the patterns and explanations on the mechanics of persistence

Thanks



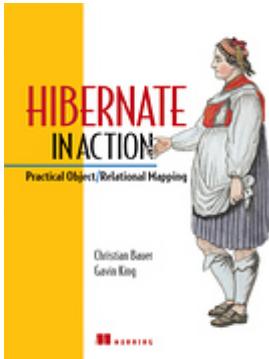
Questions

Pointers to Additional Material



- Find many of the patterns free of charge at <http://www.objectarchitects.de/>
- Find similar patterns in Martin Fowler's book „Patterns of Enterprise Application Architecture“ (ISBN 0-321-12742-0)
- Find distribution schemes and information on EJB mapping at <http://www.service-architecture.com/>
 - site by Doug Barry - contains a lot on persistence architectures, ODMG, O/R mappers, ...
- Find good code examples for JDO (Java Data Objects) as one possible Java persistence layer at www.JDOcentral.com. Also highly recommended book „Java Data Objects“ by David Jordan and Craig Russell (ISBN 0-596-0026-9)

Pointers to Additional Material



- Christian Bauer and Gavin King: Their book “Hibernate in Action” coming up. State December 2004: In public review at <http://theserverside.com/resources/HibernateReview.jsp>

Pointers to Additional Material



- have a look at one of Scott Ambler's web sites
<http://www.agiledata.org> - you'll find ample high quality papers for free

Credits

- thanks for some slides from Jens Coldewey, Coldewey Consulting. Please visit Jens' web-site at <http://www.coldewey.com/>
- Thanks to the JBoss people for the Crime Portal idea
- Thanks to David Jordan and Craig Russel for their excellent JDO book (ISBN 0-596-0026-9)