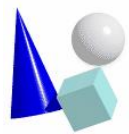
CHAIR OF INFORMATION SYSTEMS III
UNIVERSITY OF MANNHEIM
GERMANY

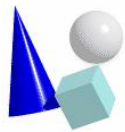


UNIVERSITY OF MANNHEIM

Using JDOSecure to Introduce Role-Based Permissions to Java Data Objects-based Applications

17th International Conference on Database and Expert Systems Applications (DEXA 2006)

September 4-8, 2006, Krakow, Poland



Agenda

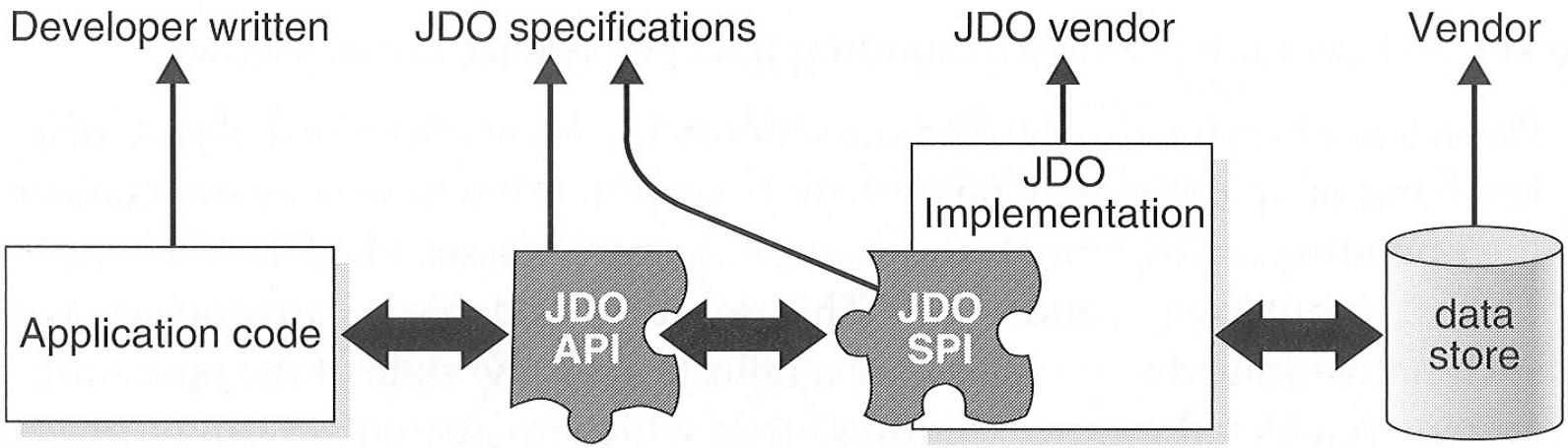
1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Novel Security Approach JDOSecure
4. Using JDOSecure in Context of a JDO Based Application
5. Conclusion and Further Work

The Java Data Objects-Specification

- Java Data Objects (JDO) is a Java API that enables application developers to deal with persistent objects in a transparent fashion (*transparent Persistence*)
- Provides a data store independent abstraction layer and enables the mapping of Java objects to any type of data store (RDBMS, OODBMS, file system, etc.)
- JDO was developed by an initiative of Sun Microsystems under the auspices of the Java Community Process
- The first version of JDO was introduced in May 2003
- Current version from May 2006: "Java Data Objects 2.0"

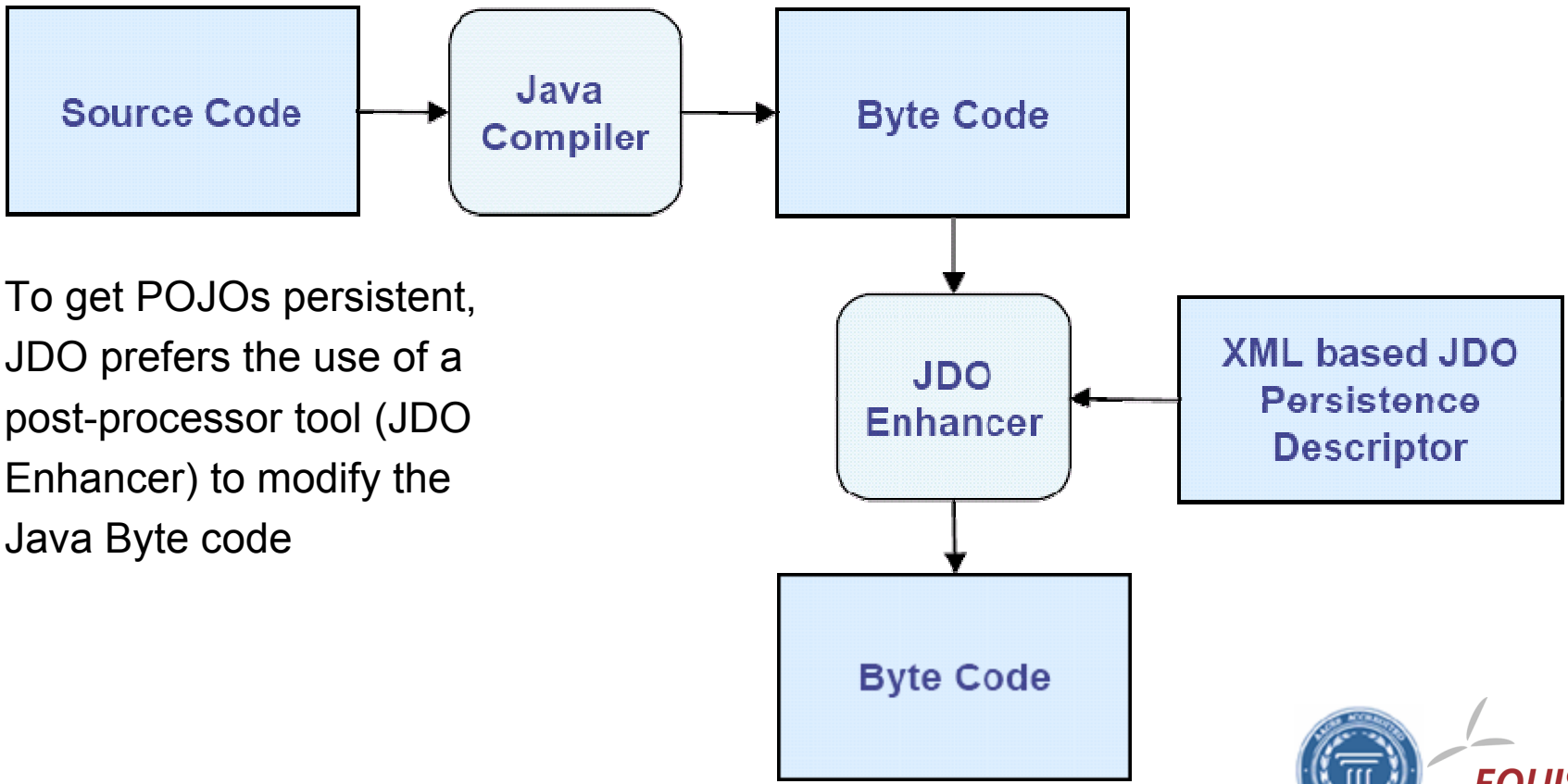
The Java Data Objects-Specification

- The JDO specification includes two Packages:
  - JDO Application Programming Interface (JDO API)
  - JDO Service Provider Interface (JDO SPI)



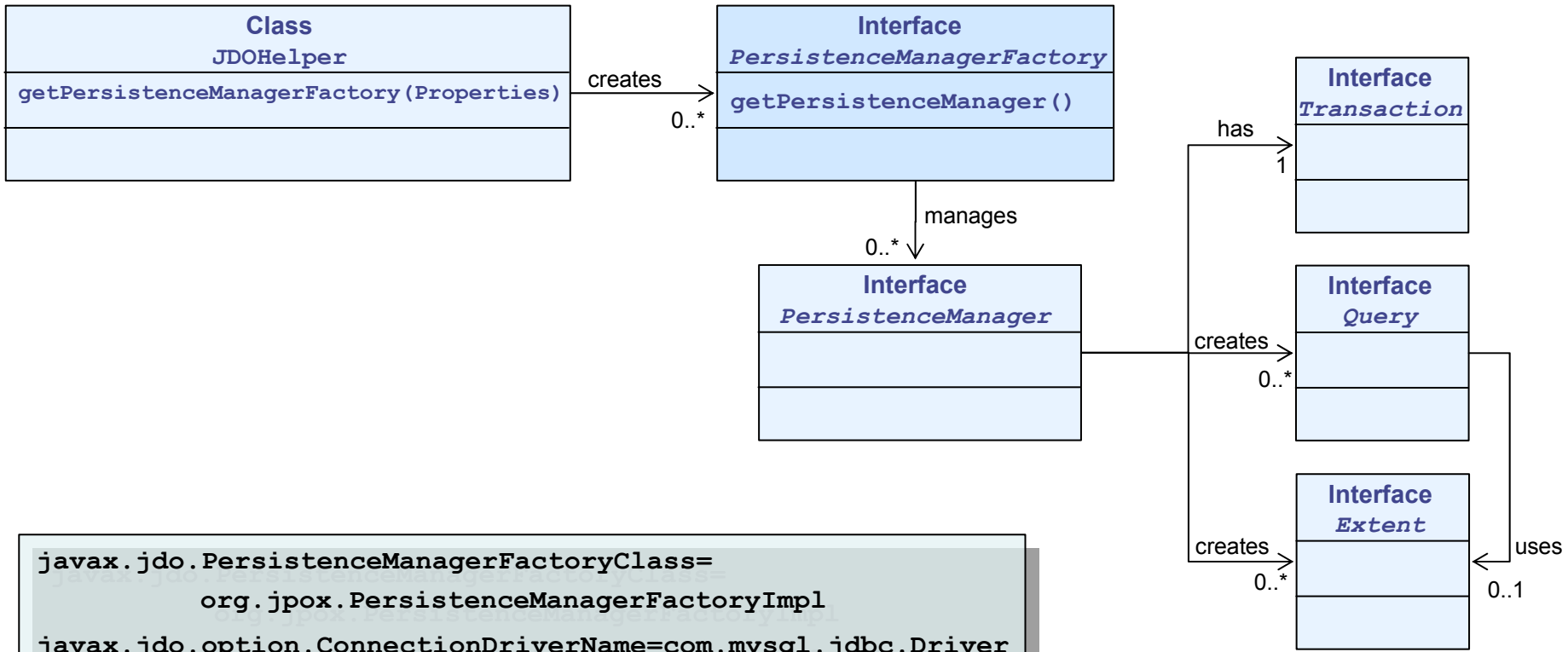
Source: Core Java Data Objects

The Java Data Objects-Specification



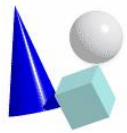
To get POJOs persistent, JDO prefers the use of a post-processor tool (JDO Enhancer) to modify the Java Byte code

The Java Data Objects-Specification



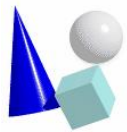

```
javax.jdo.PersistenceManagerFactoryClass=
          org.jpox.PersistenceManagerFactoryImpl
javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
javax.jdo.option.ConnectionURL=jdbc:mysql://localhost/jpox
javax.jdo.option.ConnectionUserName=merz
javax.jdo.option.ConnectionPassword=********
```

EQUIS ACCREDITED



Agenda

1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Novel Security Approach JDOSecure
4. Using JDOSecure in Context of a JDO Based Application
5. Conclusion and Further Work



JDO Security Deficits

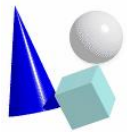
- JDO is a lightweight persistence approach without distributed access functions, multi-address-space communication or role-based security
- Consequently, the JDO persistence layer does not provide any methods for user authentication or authorization
- When the JDO database connection is established, everyone has full access privileges to store, query, update and delete persistent objects
  - **`getObjectById()`** allows to receive any persistent instance
  - **`deletePersistent()`** enables a user to delete each persistent object from the data store

JDO Security Deficits

- Slight improvement: Setting up individual user IDs at the level of the data store. This would allow the construction of different `PersistenceManagerFactory` instances
- But if all users should have access to a common database, individual user IDs and permissions have to be defined inside the data store
- E.g. when using a relational database, the permissions has to be configured based on the object-relational mapping scheme and the structure of the tables
- Strong dependency between the user application and the data store. A later replacement of the currently preferred data store leads to a time consuming and expensive migration.
- The strong binding of security permissions to a specific data store would contradict the intention of JDO, which is to provide application programmers a data store independent persistence abstraction layer

Agenda

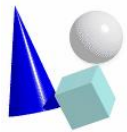
1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Novel Security Approach JDOSecure
4. Using JDOSecure in Context of a JDO Based Application
5. Conclusion and Further Work



The Novel Security Approach JDOSecure

Design Goals:

- The aim of JDOSecure is not a complete redesign of JDO, but to extend the existing specification with additional security functionality. Therefore, we put emphasis on the following aspects:
  - **Standard Compliance:** Security enhancements were developed as a JDO “add-on” to ensure standard compliance and portability of existing applications
  - **Portability:** JDOSecure-based applications should be able to run on a multitude of different JDO implementations
  - **Use of Existing Java Specifications (e.g. JAAS):** Could reduce development time/costs and minimizes security risks



The Novel Security Approach JDOSecure

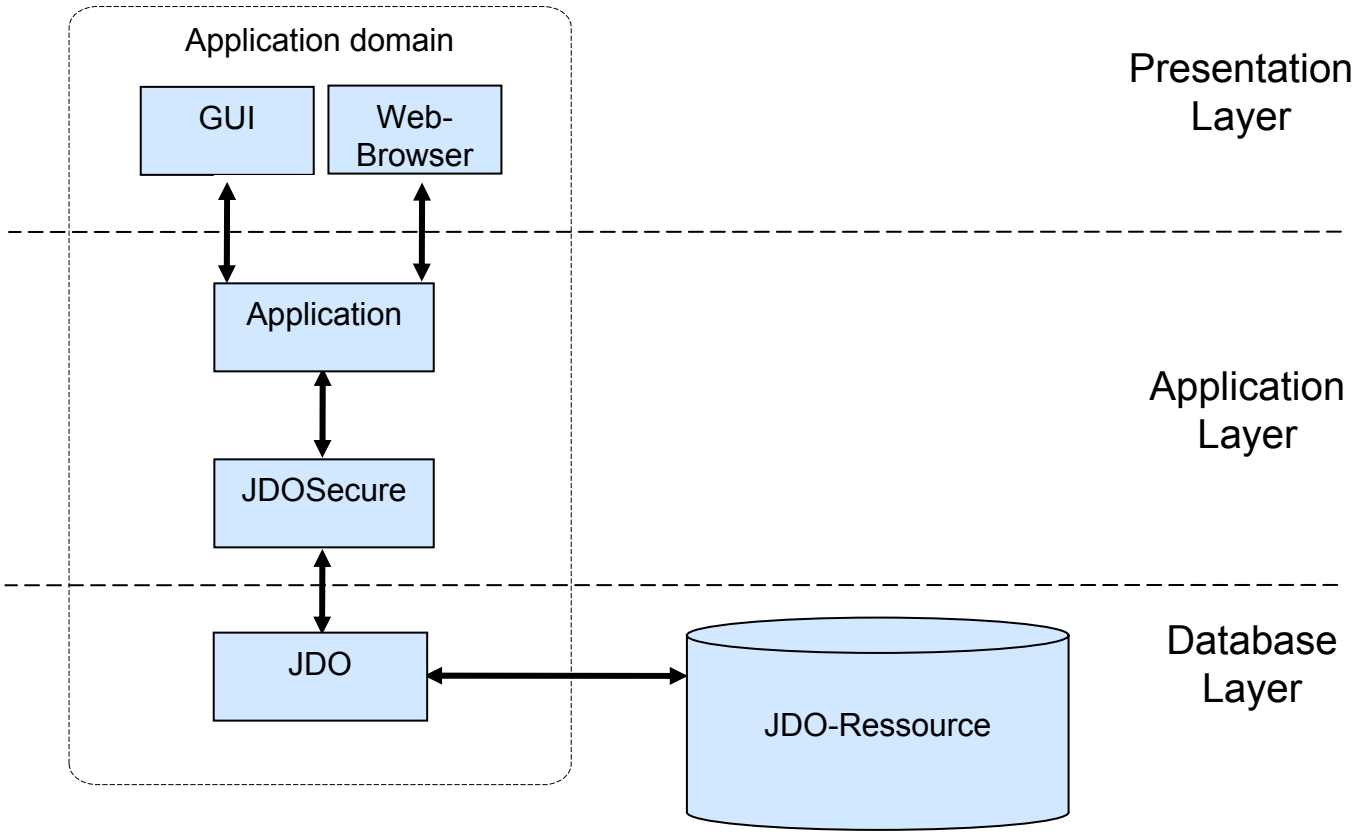
Challenges for developing a security extension to JDO:

Providing sufficient access-privileges with regard to

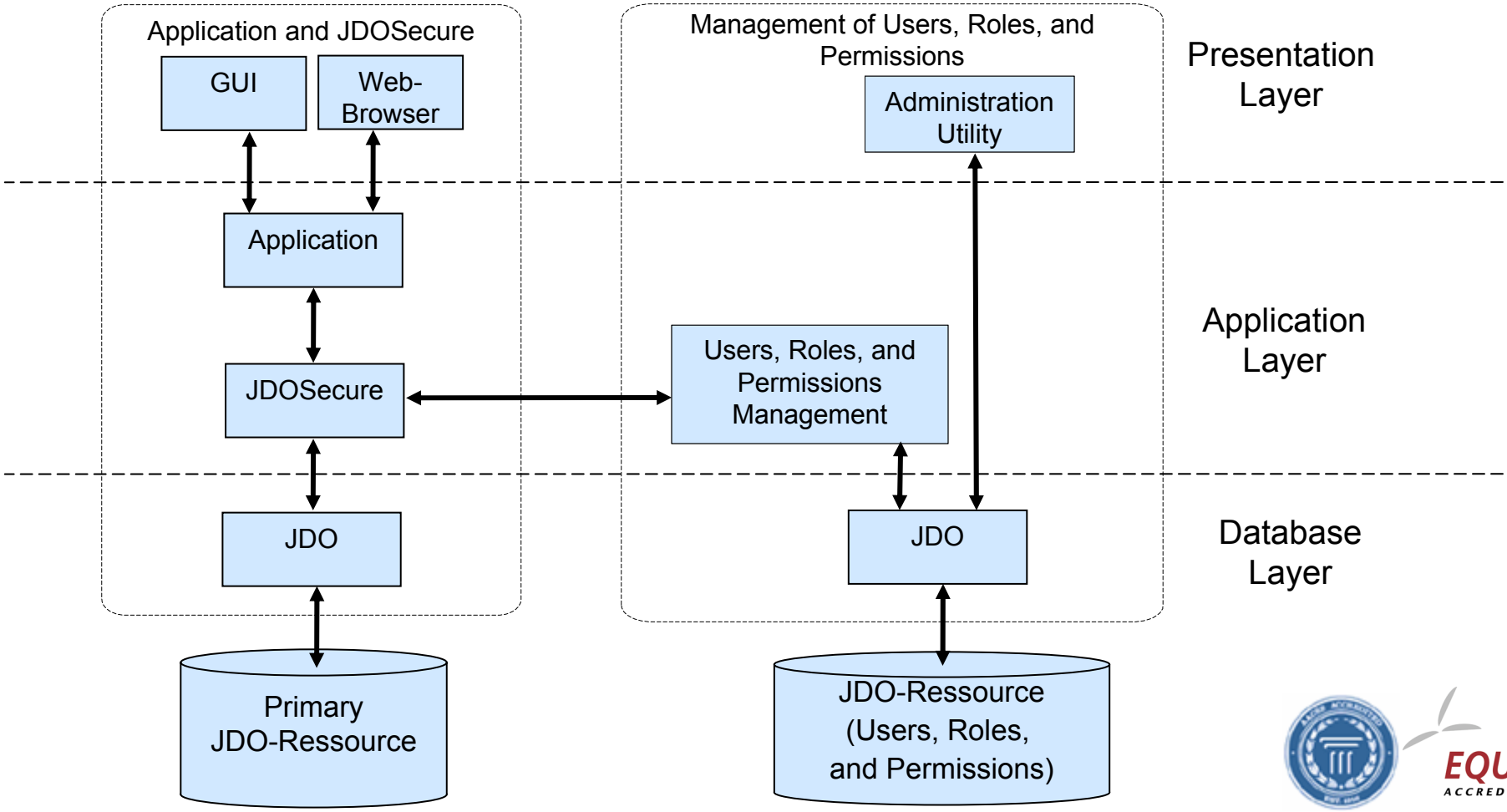
- Users and their roles
- Specific packages, classes and objects
- CRUD-Operations (Create, Retrieve, Update, Delete)

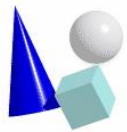
The Novel Security Approach JDOSecure

Interposing access control in the application domain



The Management of Users, Roles, and Permissions





The Novel Security Approach JDOSecure

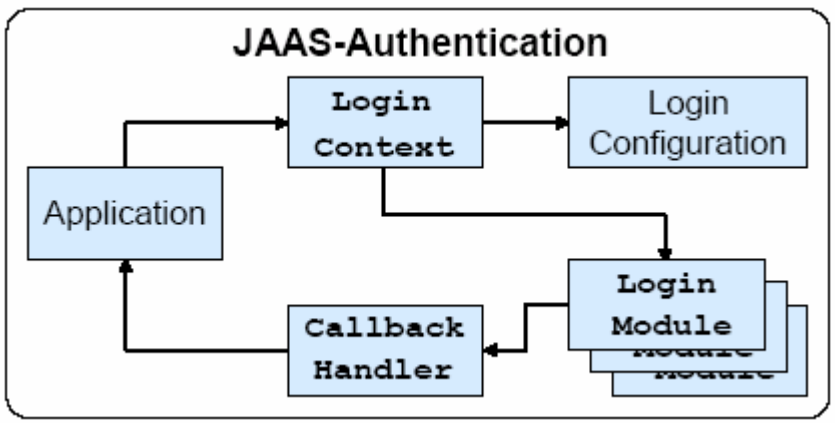
How to create an entry point for applications?

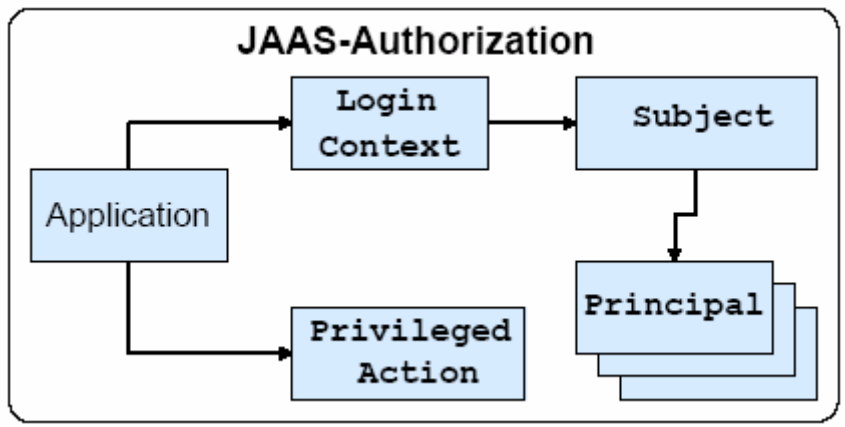
- A **`JDOSecureHelper`** class, derived from the original **`JDOHelper`**, overrides the **`getPersistenceManagerFactory()`** method
- It returns a dynamic proxy instance instead the real **`PersistenceManager`**
- This allows to introduce access-control capabilities in the associated **`PMInvocationHandler`**
- The dynamic proxy approach enables the collaboration with any JDO implementation, without source code modification

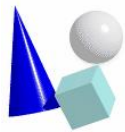
The Novel Security Approach JDOSecure

How to implement access-control?

- Java Authentication and Authorization Service (JAAS) allows to restrict the access to resources depending on the currently logged on user





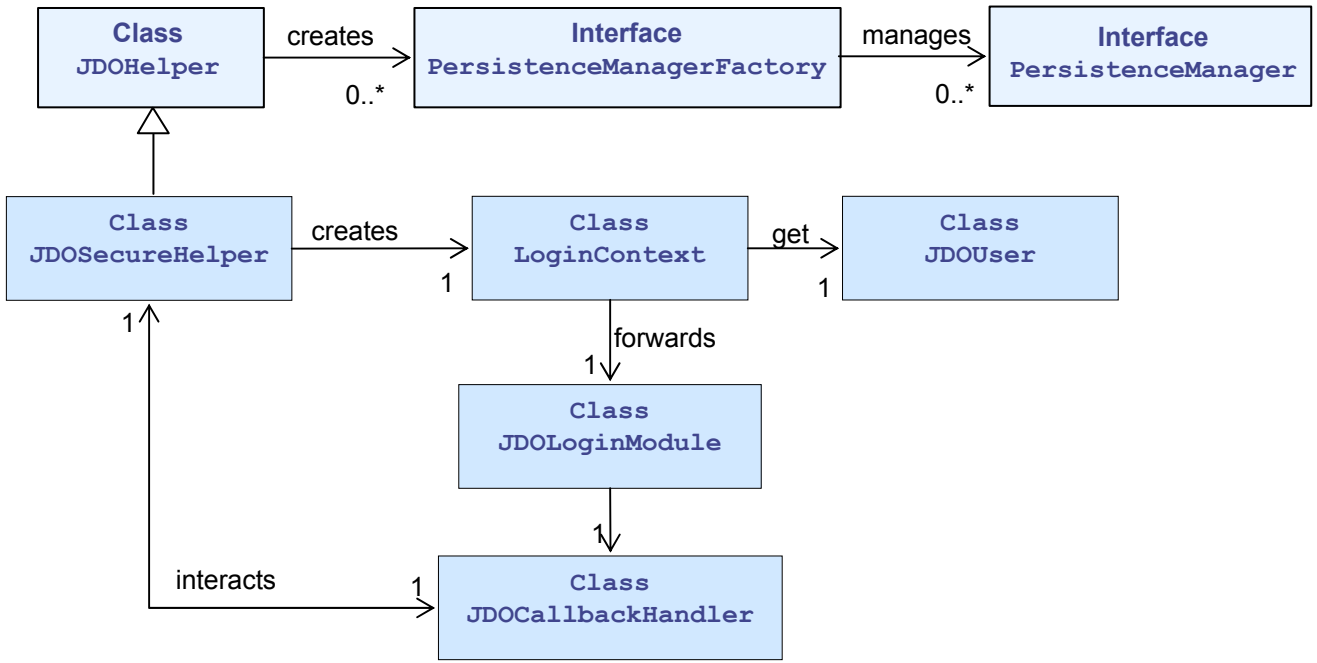


The Novel Security Approach JDOSecure

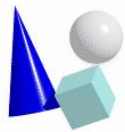
Authentication

- The **`Properties`** object passed to the **`JDOHelper`** class contains amongst others user ID and password to access a JDO resource
- The **`JDOSecureHelper`** class uses this information to authenticate the user
- If successful, the user ID and the password are replaced before the database connection is established
- The replacement is necessary to prevent a direct user connection to the database

The Novel Security Approach JDOSecure



Context between JAAS-Authentication and JDOSecure



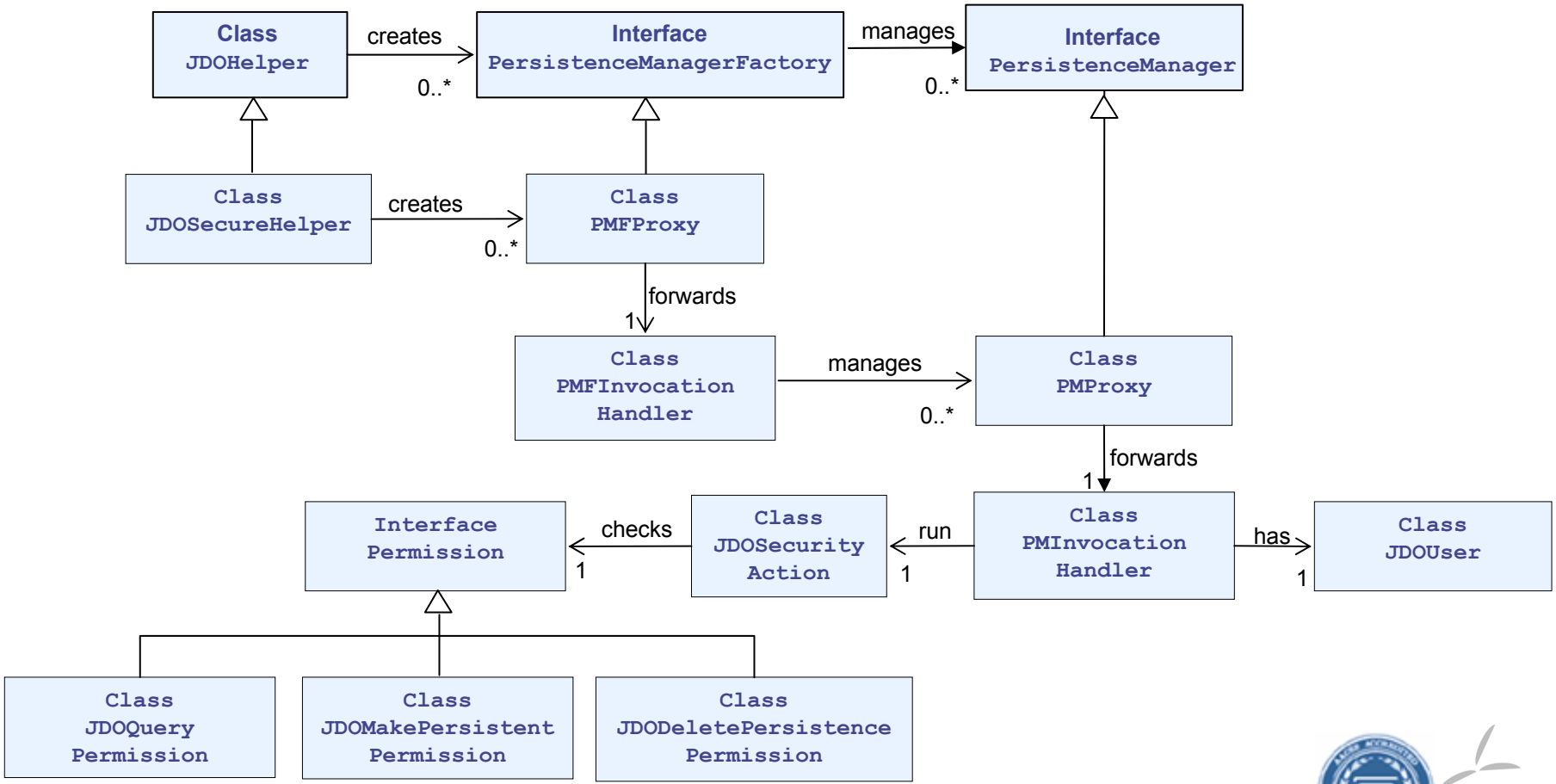
The Novel Security Approach JDOSecure

Authorization

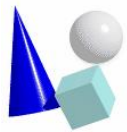
- Calls to the **`PersistenceManager`** proxy are only delegated, if the authenticated user has appropriate permissions
- Inside the **`PMInvocationHandler`**, the access-control is delegated to Java **`AccessController`** as part of JAAS
- E.g. the permission to invoke **`makePersistent()`** for objects of the package "org.sample" and a user ”sample” could be defined in a JAAS policy file:

```
grant Principal JDOUser "sample"{
    permission JDOMakePersistentPermission "org.sample.*";
}
```

The Novel Security Approach JDOSecure



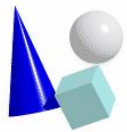
Context between JAAS-Authorization and JDOSecure



The Novel Security Approach JDOSecure

Update of object attributes:

- JDO doesn't provide any methods to update object attributes or flushing instances after an update to the data store ("transparent persistence")
- Within the JDO Enhancement process every persistence capable instance is assigned with **`StateManager`**
- This instance is responsible for flushing changes to the data store
- To enable JDOSecure to permit or disallow the update process in regard with the permissions auf a user, the **`StateManager`** is also replaced by a dynamic proxy instance



Agenda

1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Novel Security Approach JDOSecure
4. Using JDOSecure in Context of a JDO Based Application
5. Conclusion and Further Work

JDOSecure in Context of a JDO Based Application

Setting up JDOSecure:

- Configuring authentication data: User IDs and passwords of those, who are generally allowed to access the persistent instances of the data store
- Define the master-password to allow JDOSecure to access the underlying JDO resource (Necessary to replace the user ID and password to prevent a direct user connection to the database)
- The set-up of user/role permissions have to be configured in the JAAS policy-file.
- In a scenario where a “guest” should be allowed to query all instances of a package `sample`, the permission can be defined for this principal in a JAAS policy-file as follows:

```
grant Principal JDOUser "guest"{
     permission JDOQueryPermission "sample.*";
}
```

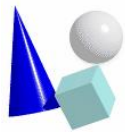
JDOSecure in Context of a JDO Based Application

Activate the SecurityManager

- As JDOSecure is based on JAAS, the Java **`SecurityManager`** has to be activated to participate of the security benefits, e.g. by using JVM command-line switches:

```
-Djava.security.manager
-Djava.security.policy=JDOAccessControl.policy
-Djava.security.auth.login.config=JDOSecure_jaas.config
```

- The first switch is responsible to activate the Java **`SecurityManager`**. The second defines the source, where the user permissions are located, and the last refers to a configuration file, which defines the **`JDOLoginModule`** to authenticate a user



JDOSecure in Context of a JDO Based Application

Application adjustments:

- Only one modification in the Java source code has to be made:
  The **`JDOSecureHelper`** class has to be used instead of the class **`JDOHelper`** to receive an instance of a **`PersistenceManagerFactory`**
- An application could be extended in additional by adding **`try`** and **`catch(AccessControlException ace)`** blocks to handle possible security exceptions at run-time

Conclusion:

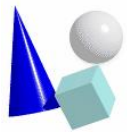
- The user and the application does not even be aware of the interception
- Within these slight modifications it gets possible to introduce rolebased permissions in every JDO-based application

Agenda

1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Novel Security Approach JDOSecure
4. Using JDOSecure in Context of a JDO Based Application
5. Conclusion and Further Work

Concluding Remarks

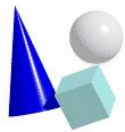
- JDOSecure introduces a fine grained access control mechanism to JDO
- It and allows the definition of role-based permissions
- The permissions can be defined individually
  - for every user/role
  - with regards to certain operations (create, delete, update, or query)
  - and for a specific object, class, or package
- The dynamic proxy approach enables JDOSecure to collaborate with any JDO implementation without source code modification
- The user and the application does not even be aware of the interception
- Within these slight modifications it gets possible to introduce rolebased permissions in every JDO-based application



Concluding Remarks

Further Research

- To reduce possible inconsistency and potential typos, we are developing a management solution for users, roles, and permissions, that
  - stores the authentication and authorization information in any arbitrary JDO resource
  - add an administration utility with a graphical user interface to simplify the maintenance of security privileges and permissions



Concluding Remarks

Thank you for your attention!

More information are available at:

`http://projekt-jdo.uni-mannheim.de/JDOSecure`