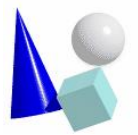
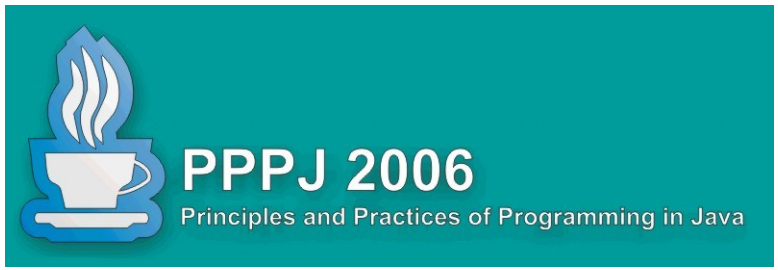
CHAIR OF INFORMATION SYSTEMS III
UNIVERSITY OF MANNHEIM
GERMANY



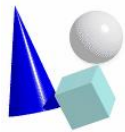
UNIVERSITY OF MANNHEIM

The Management of Users, Roles, and Permissions in JDOSecure



August 30 – September 1,

Mannheim, Germany



Agenda

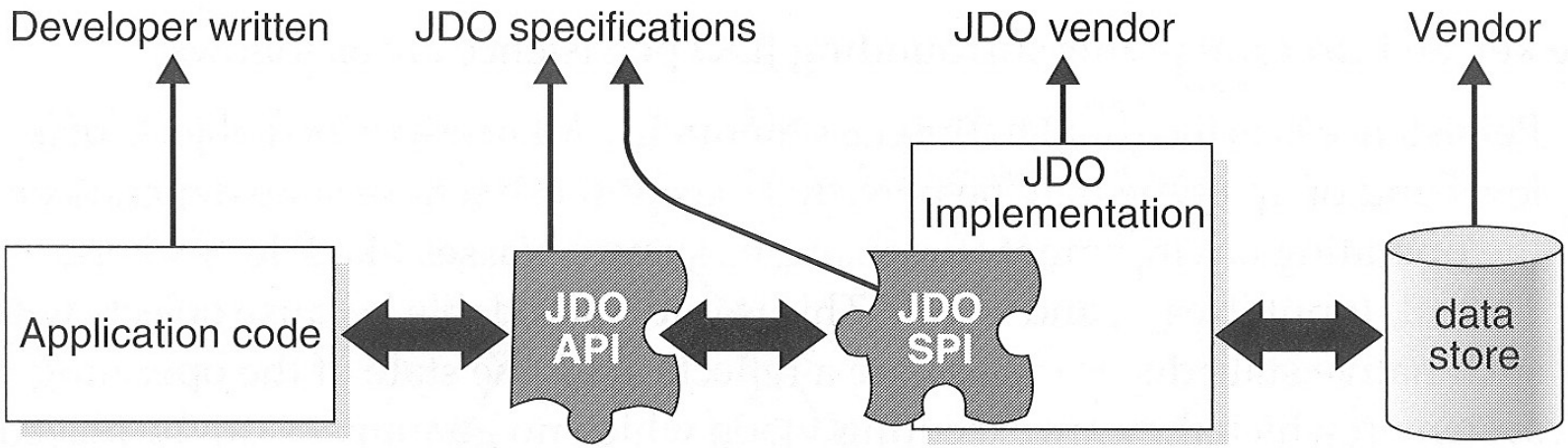
1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Architecture of JDOSecure
4. The Management of Users, Roles, and Permissions in JDOSecure
5. Conclusion and Further Work

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*)
- 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"
- 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.)

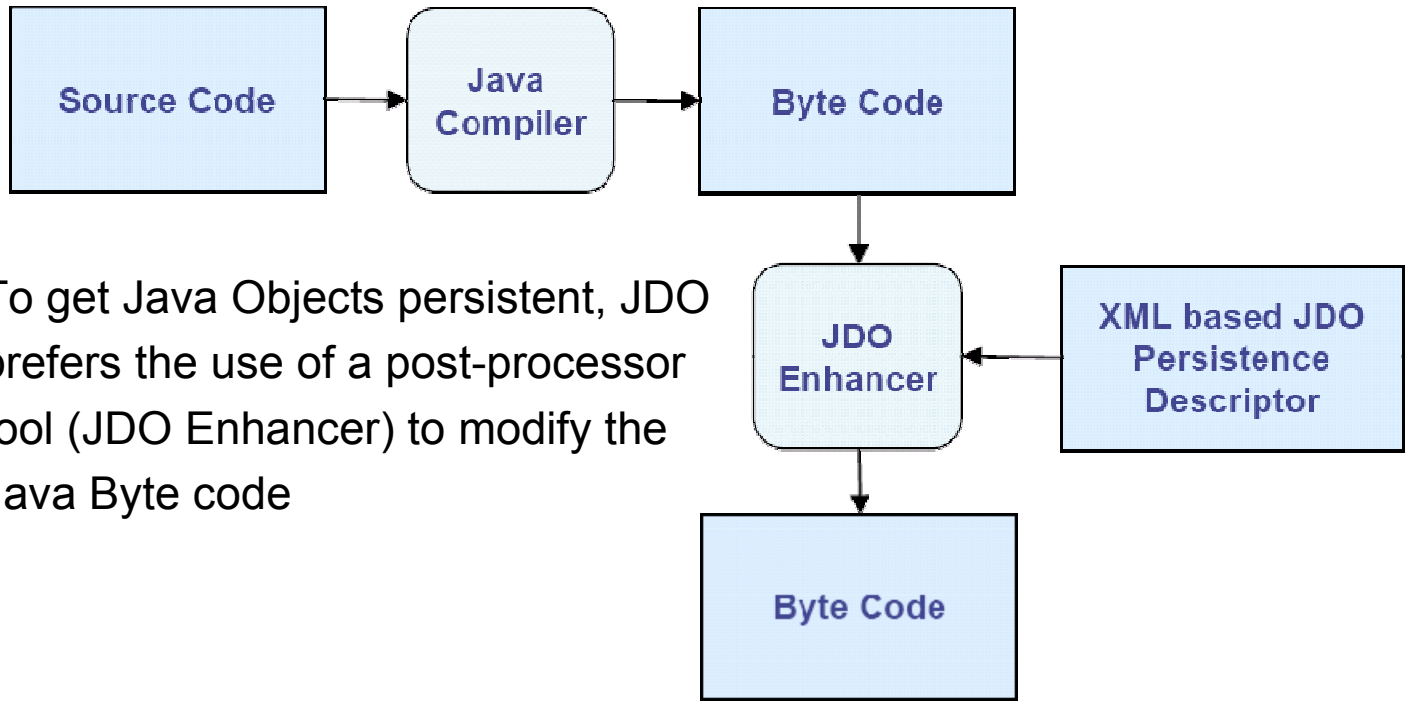
The Java Data Objects-Specification

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



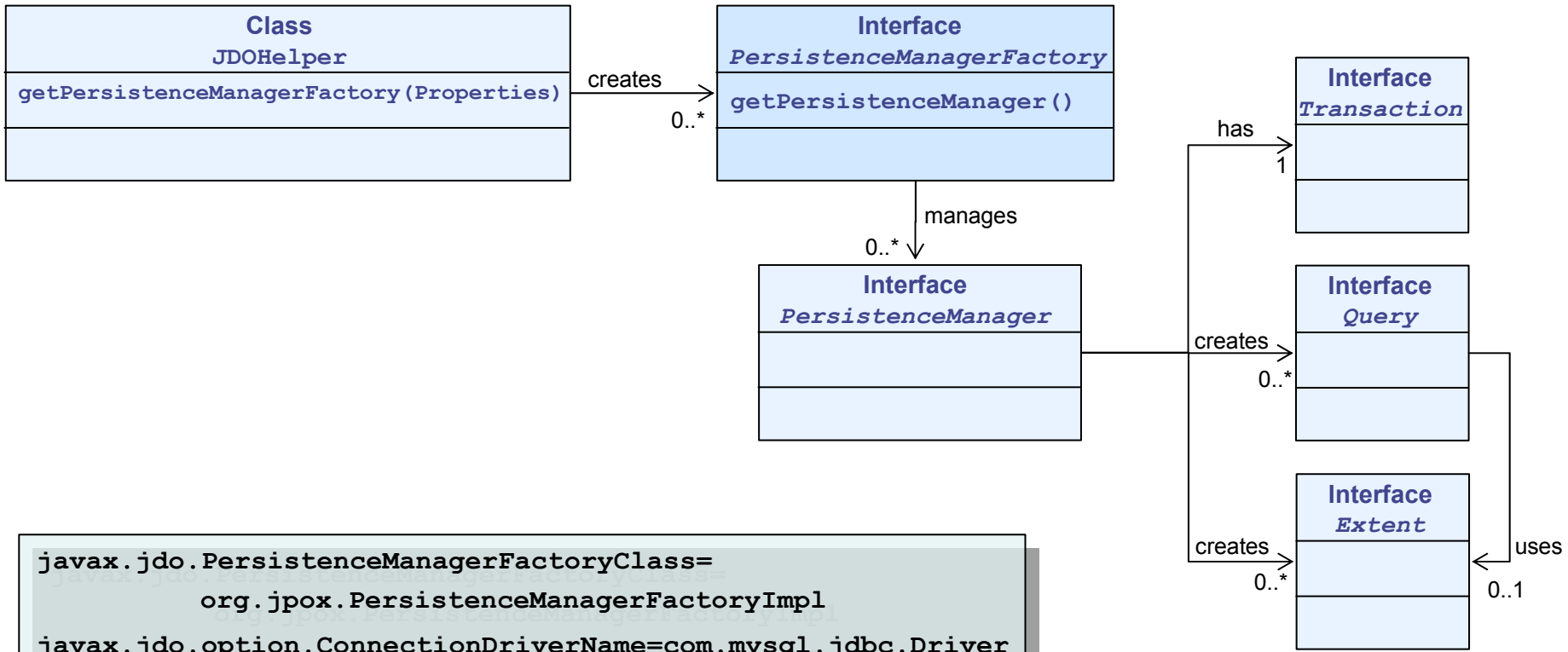
Source: Core Java Data Objects

The Java Data Objects-Specification

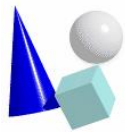


To get Java Objects persistent, JDO prefers the use of a post-processor tool (JDO Enhancer) to modify the Java Byte code

The Java Data Objects-Specification

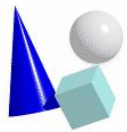



```
javax.jdo.PersistenceManagerFactoryClass=
          org.jpox.PersistenceManagerFactoryImpl
javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
javax.jdo.option.ConnectionURL=jdbc:mysql://localhost/jpox
javax.jdo.option.ConnectionUserName=merz
javax.jdo.option.ConnectionPassword=********
```



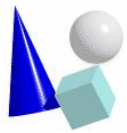
Agenda

1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Architecture of JDOSecure
4. The Management of Users, Roles, and Permissions in JDOSecure
5. Conclusion and Further Work



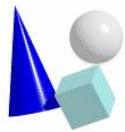
JDO Security Deficits

- 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



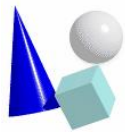
JDO Security Deficits

- Slight improvement: Using several database accounts to construct different **`PersistenceManagerFactory`** instances
- However, if the access to a common JDO resource is required, the permissions have to be defined inside the data store
- Highly complex and inflexible
  - If e.g. JDO uses a RDBMS, the permission have to be defined according to the object-relational mapping scheme and table structure
- Strong dependency between application and data store
  - A necessary replacement of the data store leads to a time consuming and expensive migration
  - Contradicts the intention of JDO, which is to provide a data store independent persistence abstraction layer



Agenda

1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Architecture of JDOSecure
4. The Management of Users, Roles, and Permissions in JDOSecure
5. Conclusion and Further Work



The Architecture of JDOSecure

Challenges for developing a security extension to JDO:

Providing sufficient access-privileges with regard to

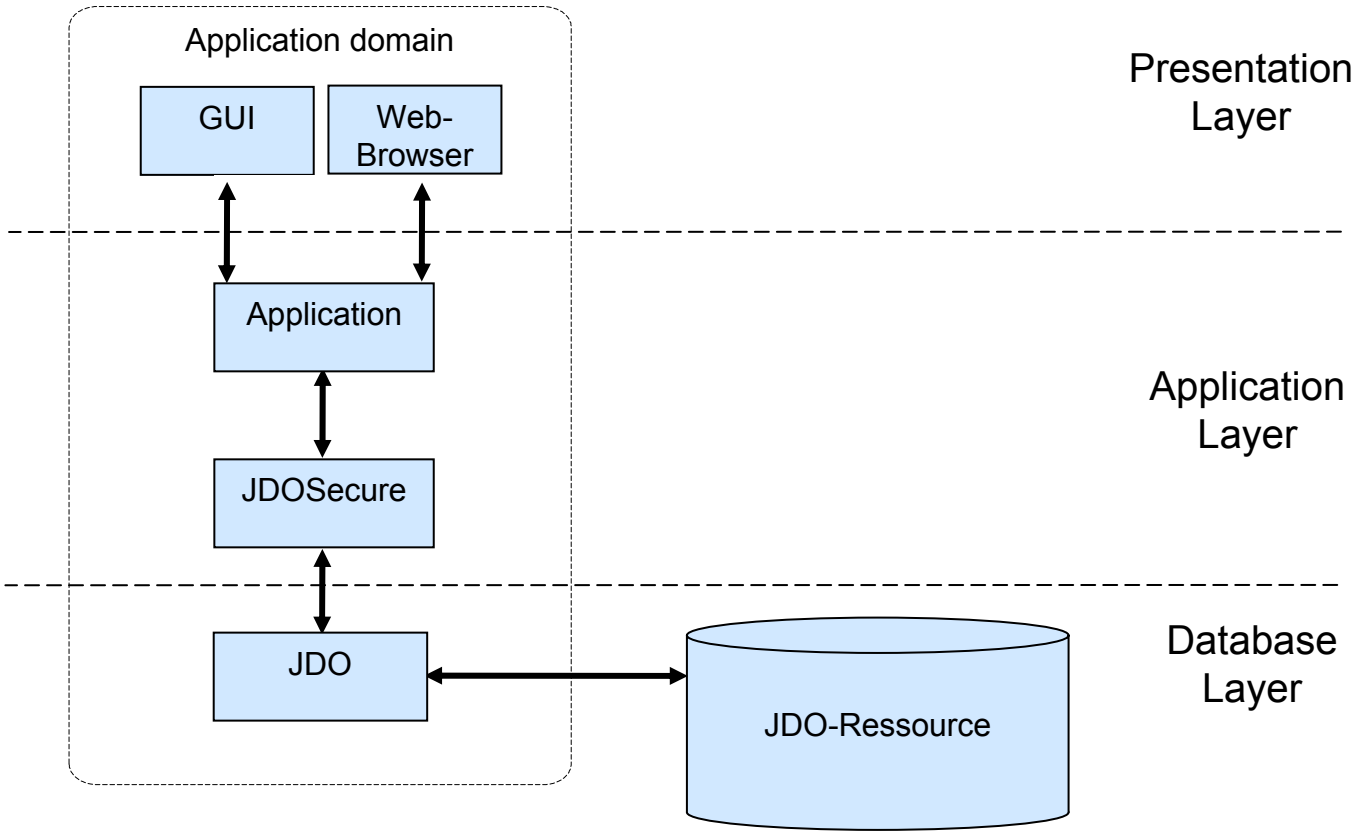
- Users and their roles
- Specific packages, classes and objects
- CRUD-Operations (Create, Retrieve, Update, Delete)

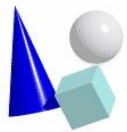
Additional requirements:

- Keep the solution JDO compatible
- Independency from a concrete JDO implementation
- No "work around" to compromise the security approach

The Architecture of JDOSecure

Interposing access control in the application domain



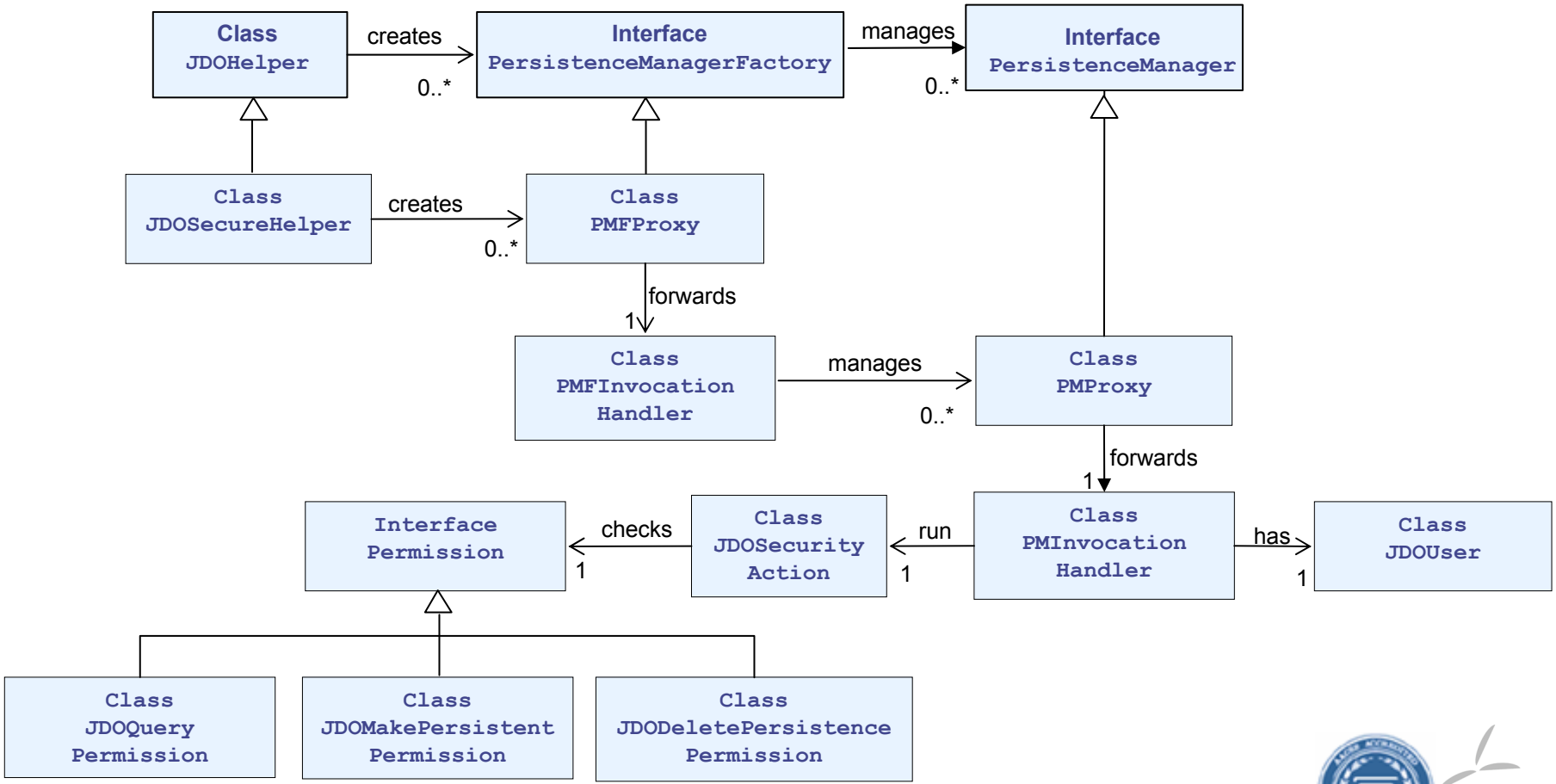


The Architecture of JDOSecure

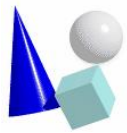
The basic idea

- A **`JDOSecureHelper`** class, derived from the original **`JDOHelper`**, overrides the **`getPersistenceManagerFactory()`** method
- It returns a dynamic proxy instance instead the original **`PersistenceManager`**
- Method invocations are only delegated from the proxy to the original **`PersistenceManager`**, if the authenticated user has appropriate permissions

The Architecture of JDOSecure



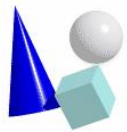
Context between JAAS-Authorization and JDOSecure



The Architecture of 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 now uses this **`Properties`** object to authenticate the user on the level of JDOSecure
- If successful, the user ID and the password in the **`Properties`** object are replaced by JDOSecure before the database connection is established
- The replacement is necessary to prevent a direct user connection to the database

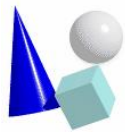


The Architecture of JDOSecure

Authorization

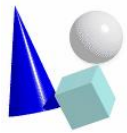
- Inside the **`PMInvocationHandler`**, the access-control is delegated to Java **`AccessController`** as part of JAAS
- The permissions has previously be defined in a local policy-file
- E.g. to invoke **`makePersistent()`** for objects of the package "org.sample" and a user ”sample” could be defined in a JAAS policy file:

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



Agenda

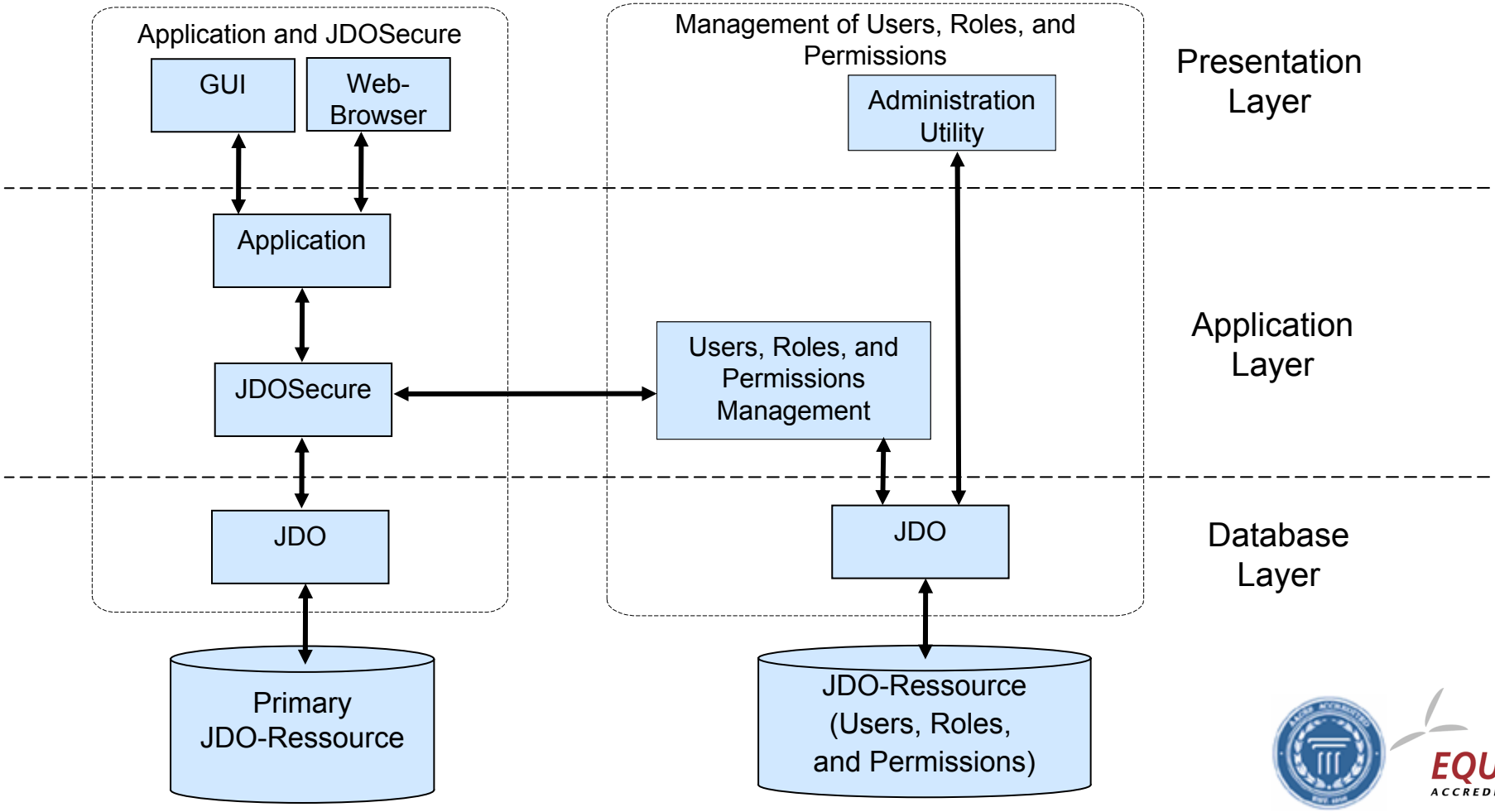
1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Architecture of JDOSecure
4. The Management of Users, Roles, and Permissions in JDOSecure
5. Conclusion and Further Work

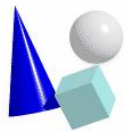


The Management of Users, Roles, and Permissions

- Defining appropriate permissions in policy text-files becomes more and more complex with an increasing number of different users and roles
- Thus, JDOSecure comprises a management solution for users, roles, and permissions
- It allows to store the authentication and authorization data in any arbitrary JDO resource
- A Java-based administration utility (GUI) simplifies the maintenance of security privileges and permissions

The Management of Users, Roles, and Permissions





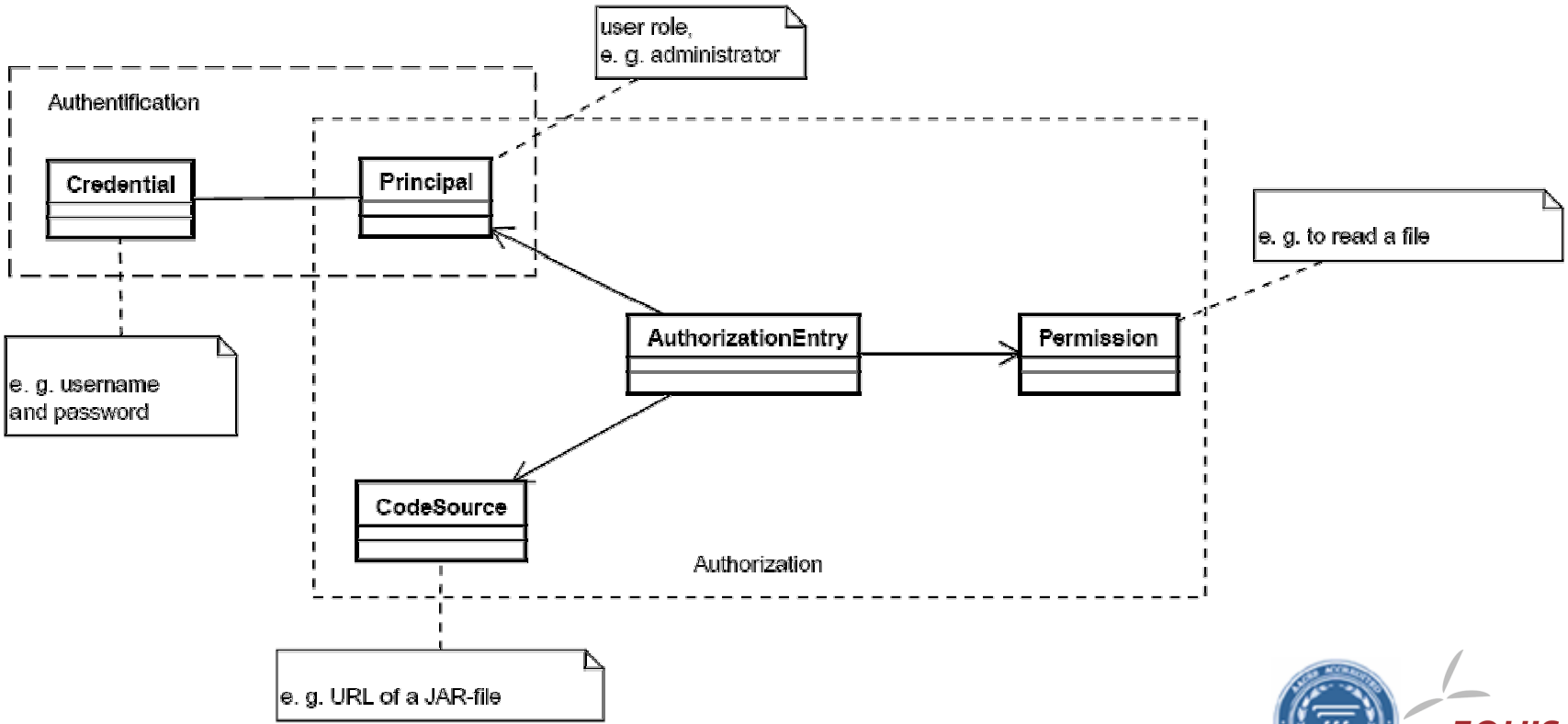
The Management of Users, Roles, and Permissions

To enable JDOSecure to use JDO to manage JAAS authentication and authorization data, a LoginModule has to be implemented and Sun's default Policy implementation has to be adopted:

- **`JDOLoginModule`**
  - Uses a **`CallbackHandler`** to get receive the username and password from the **`Properties`** Object passed to the **`JDOSecureHelper`** instance
  - Establishes a JDO connection to query for user and password information
  - The password is stored as SHA-1 hash in the JDO resource
- **`JDOPolicy:`**
  - Extends Sun's default Java Policy implementation (**`sun.security.provider.PolicyFile`**)
  - Uses a JDO resource to receive authorization data

The Management of Users, Roles, and Permissions

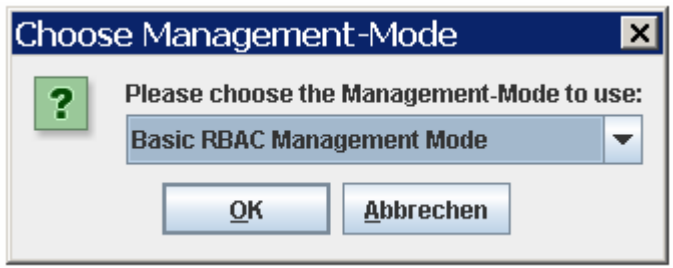
Modeling the Users, Roles, and Permissions Management Domain:



The Management of Users, Roles, and Permissions

Administration Utility:

- Simplifies the setup of authentication and authorization data
- Establishes a direct JDO connection (even from a remote computer)
- Provides two management modes:
  - A basic RBAC management mode:
    - The **`Permissions`** are bound to roles (**`Principals`**), and a user is assigned to one or more roles

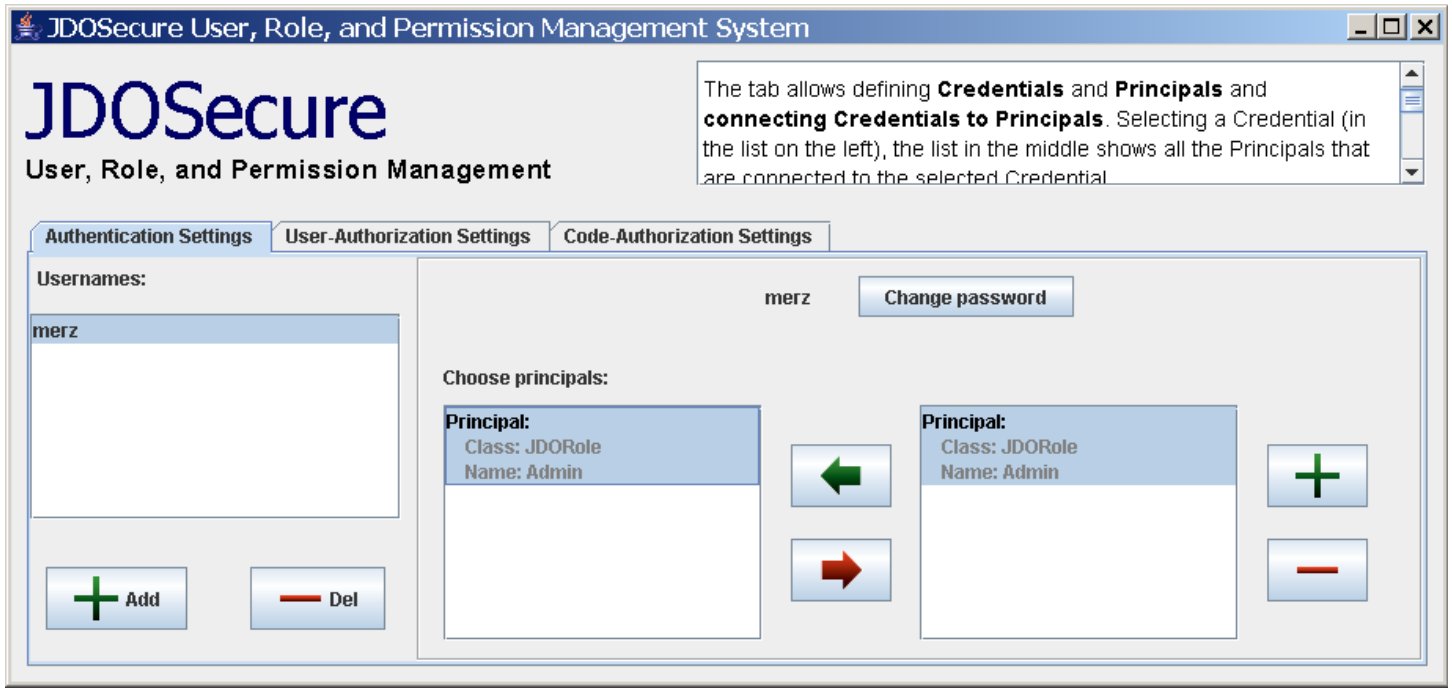


  - An extensive management mode:
    - Operates directly on the presented domain model
    - **`AuthorizationEntry`** objects could be directly constructed and bound to **`Principal`**, **`Permission`** and **`CodeSource`**
    - Highly flexible, but complex handling

EQUIS ACCREDITED

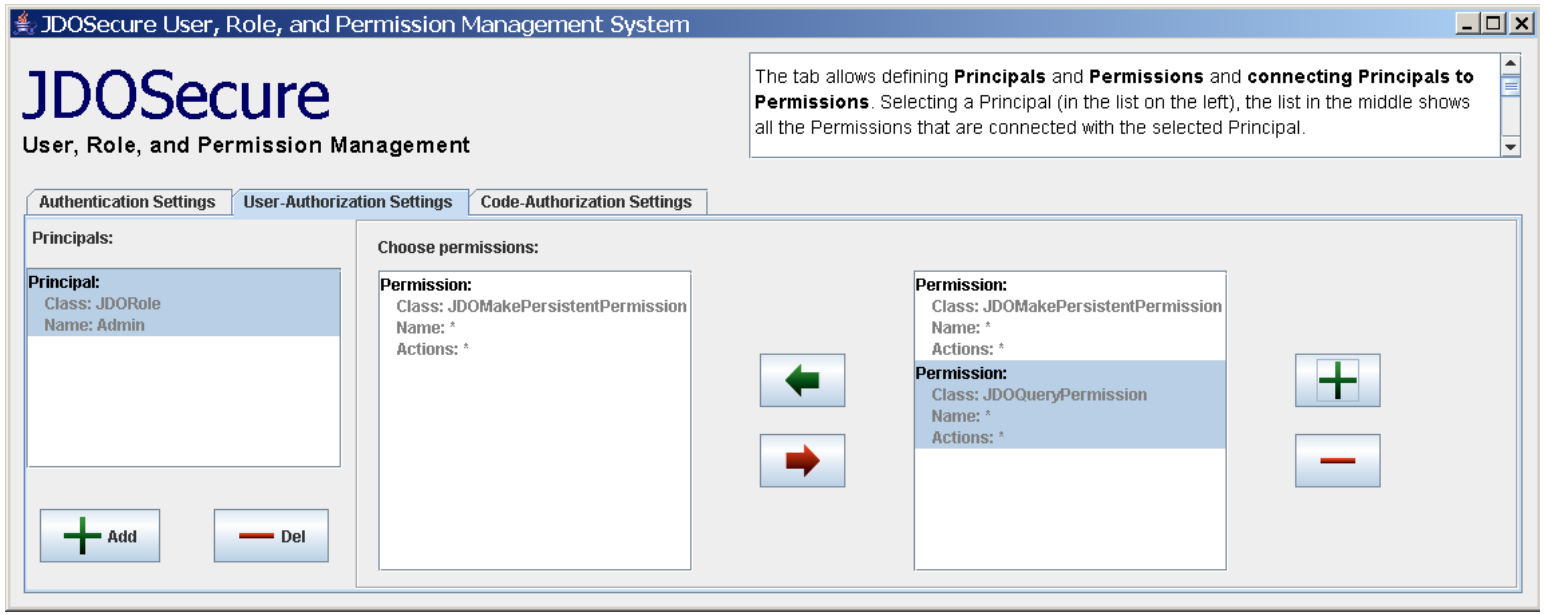
The Management of Users, Roles, and Permissions

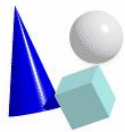
Administration Utility (Basic RBAC Management Mode):



The Management of Users, Roles, and Permissions

Administration Utility (Basic RBAC Management Mode):



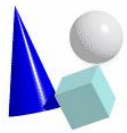


Agenda

1. The Java Data Objects Specification
2. JDO Security Deficits
3. The Architecture of JDOSecure
4. The Management of Users, Roles, and Permissions in JDOSecure
5. Conclusion and Further Work

Concluding Remarks

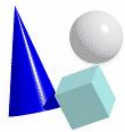
- JDOSecure introduces a fine grained access control mechanism to JDO
- The permissions can be defined individually
  - for every user/role
  - with regards to certain operations (create, delete, update, or query)
  - and for a specific object, class, or package
- To reduce possible inconsistency and potential typos in policy files,
  - the authentication and authorization data are stored in a separate JDO resource
  - a Java-based administration utility (GUI) simplifies the maintenance of security privileges and permissions



Concluding Remarks

Performance Aspects

- Even JDOSecure could improve the security of JDO applications, it becomes obvious that the management of permissions and the access control mechanism has negative performance effects
- Even worse, the dynamic proxy approach including a huge number of indirections between the constructed instances and their proxies leads to a further deterioration of performance
- By doing a basic CRUD performance test, it turns out that the performance of JDO applications will be reduces at about 15-20% on an average
- In the future, we aim to extend our performance tests and are confident of achieving a slightly better performance for JDOSecure



Concluding Remarks

Thank you for your attention!

More information are available at:

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