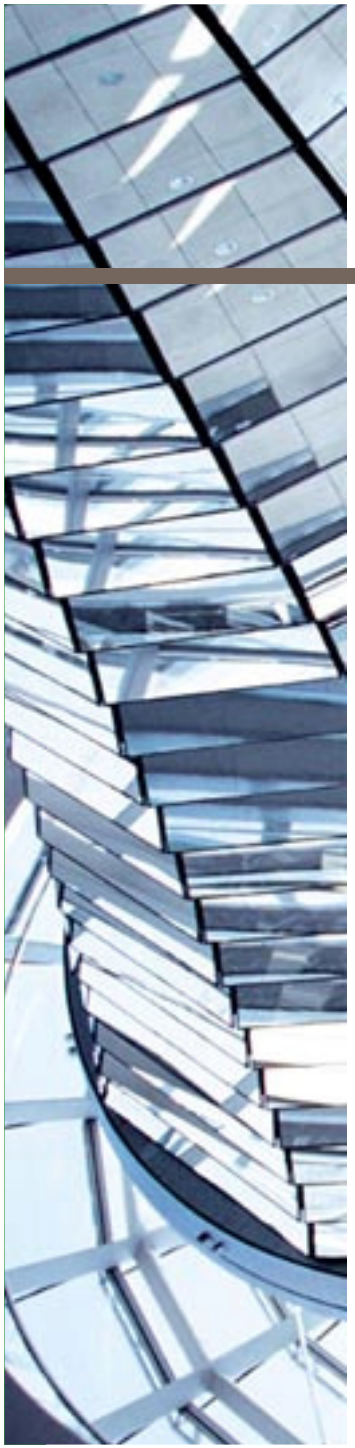


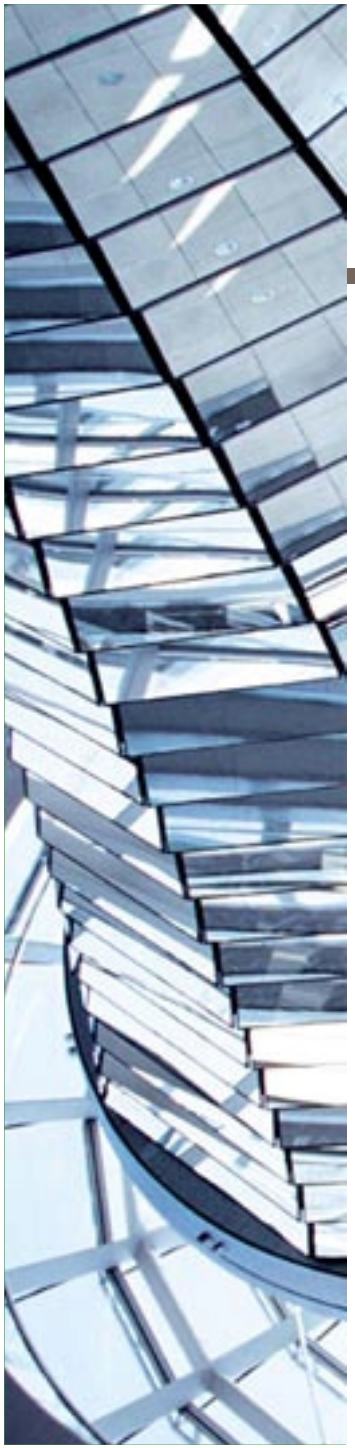
5. Authorization
Reference Architectures and Patterns

Winter Semester 2008 / 2009
Prof. Dr. Bernhard Humm
Darmstadt University of Applied Sciences
Department of Computer Science

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

The lecture in the context of the entire course

1. Introduction
2. A reference architecture for business information systems
3. Application kernel
4. Persistence and transaction
5. Authorization
6. Client architecture
7. Other reference architectures: SOA, BI, systems integration, …



Agenda

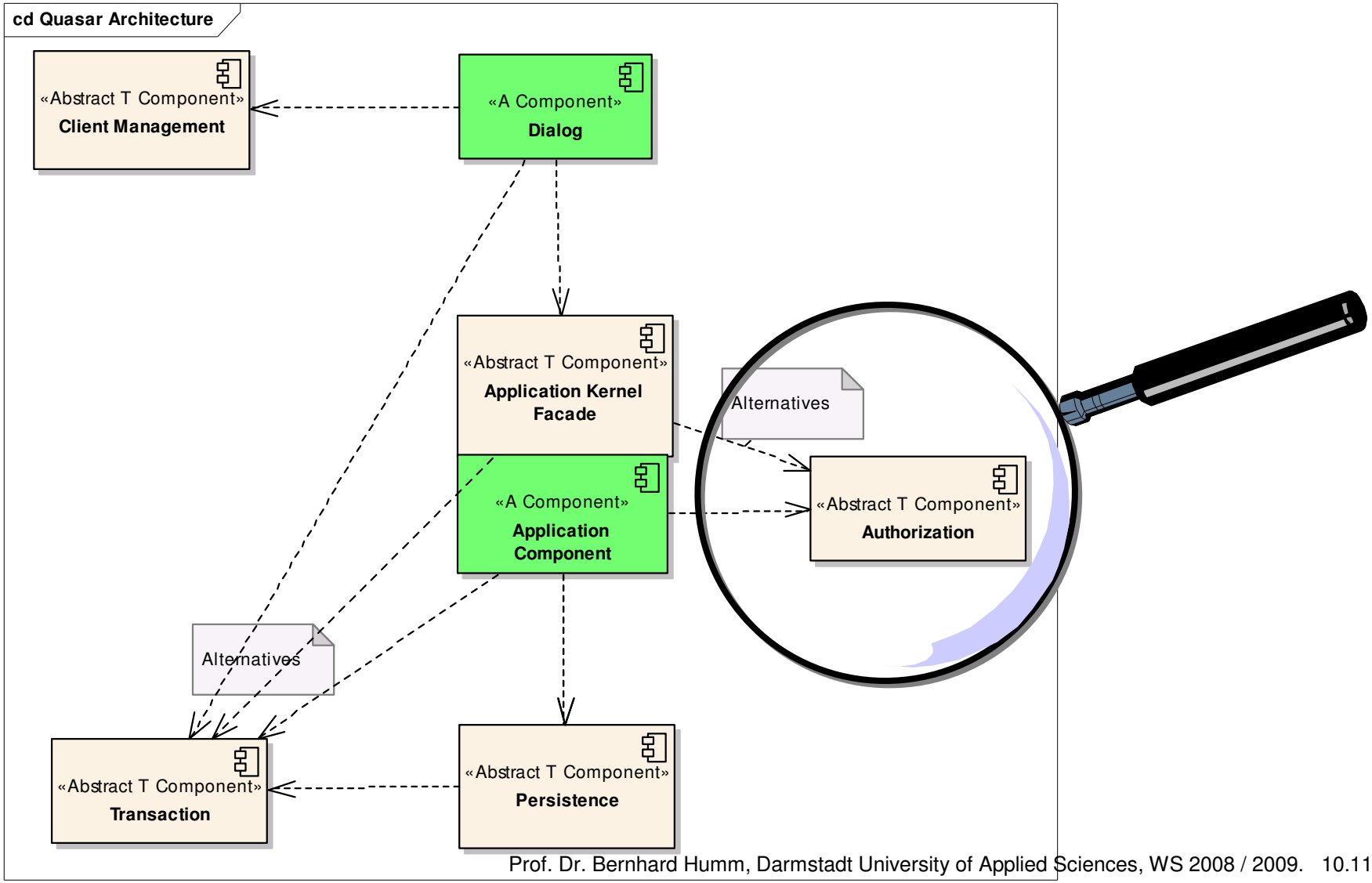
➔ **Reference architecture**

Authentication

Authorization

Sequences

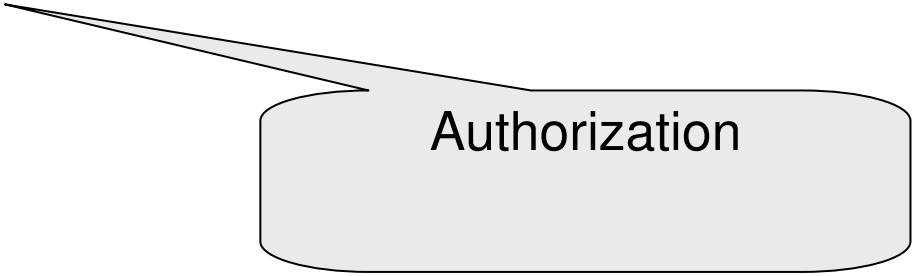
Reference architecture for business information systems Quasar (Quality Software Architecture)



What is Authorization all about?

- Hello! Do we know each other?

- Hey, what do you want to do?
  Are you allowed to do that?



Tasks of the Authorization

- Administration of …
  - User data:
    - "Joe Bloggs (login name 'bloggs') works in the finance department."
  - Permission data:
    - "Finance department staff may read account data."
- Performing …
  - Authentication:
    - "Is this really Joe Blocks who ist trying to log in to the system?"
  - Authorization:
    - „May Joe Bloggs read Mary Poppin's account data?"

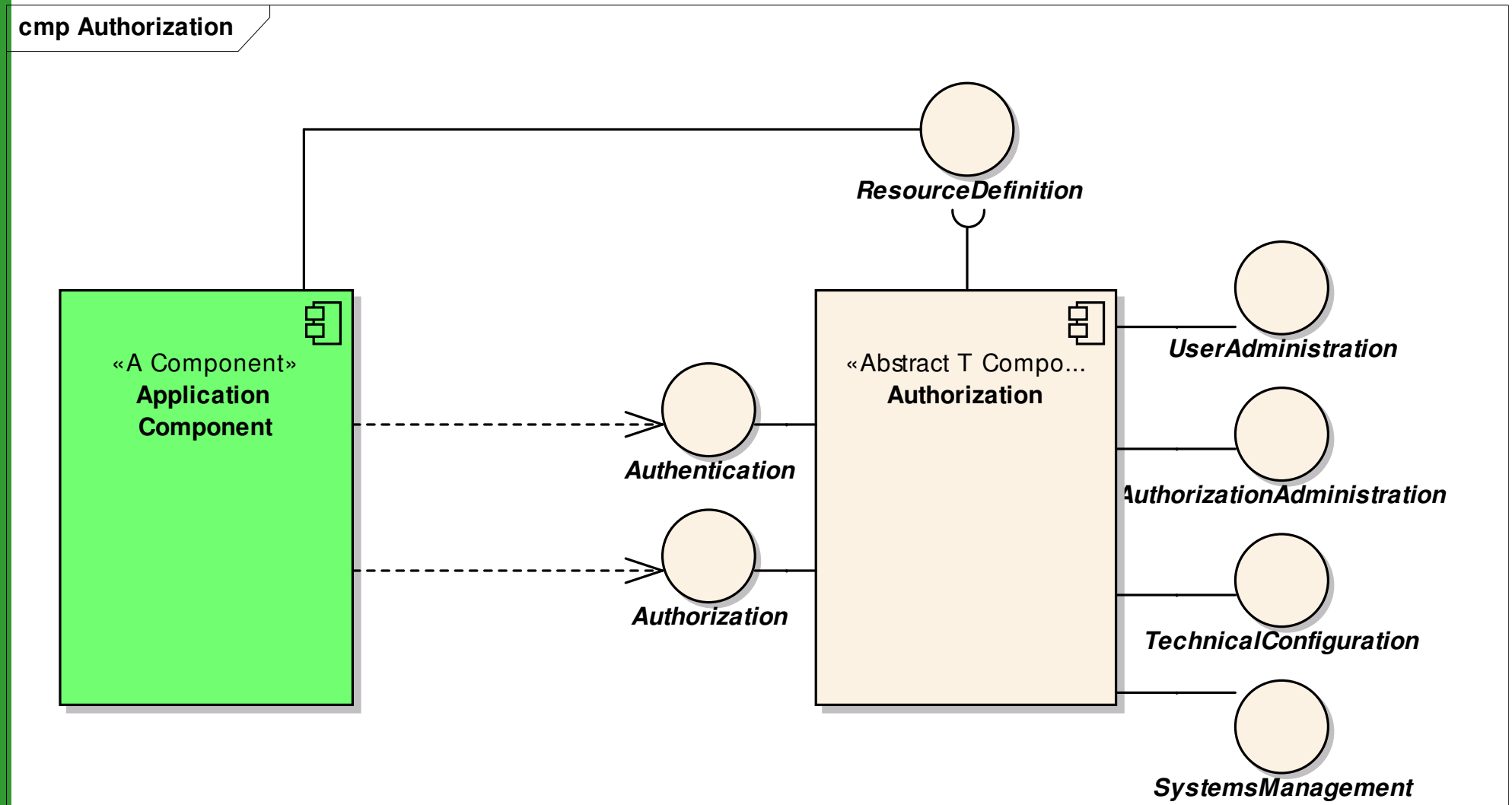
User Administration

Authorization Administration

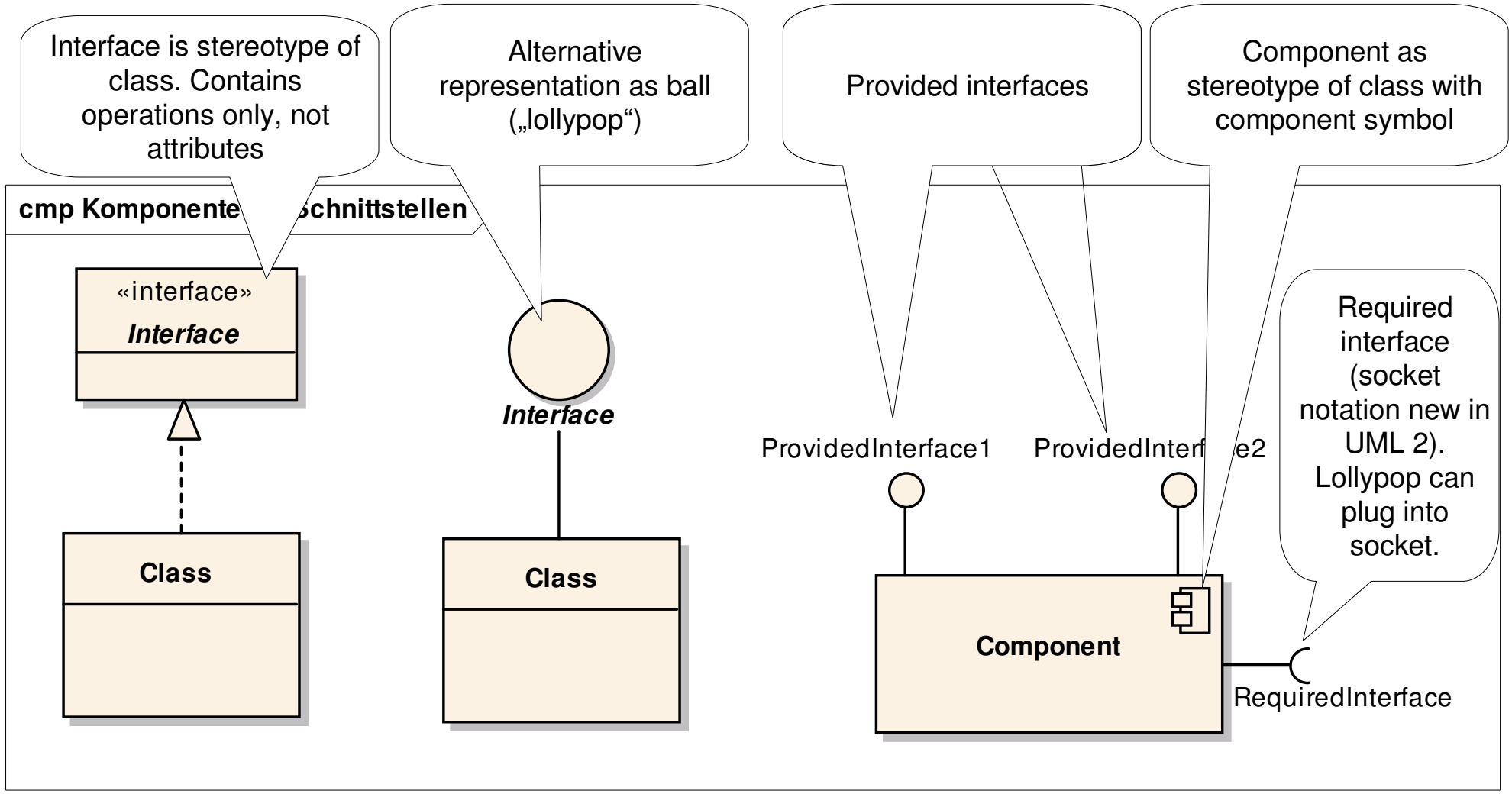
Authentication

Authorization

Authorization reference interfaces

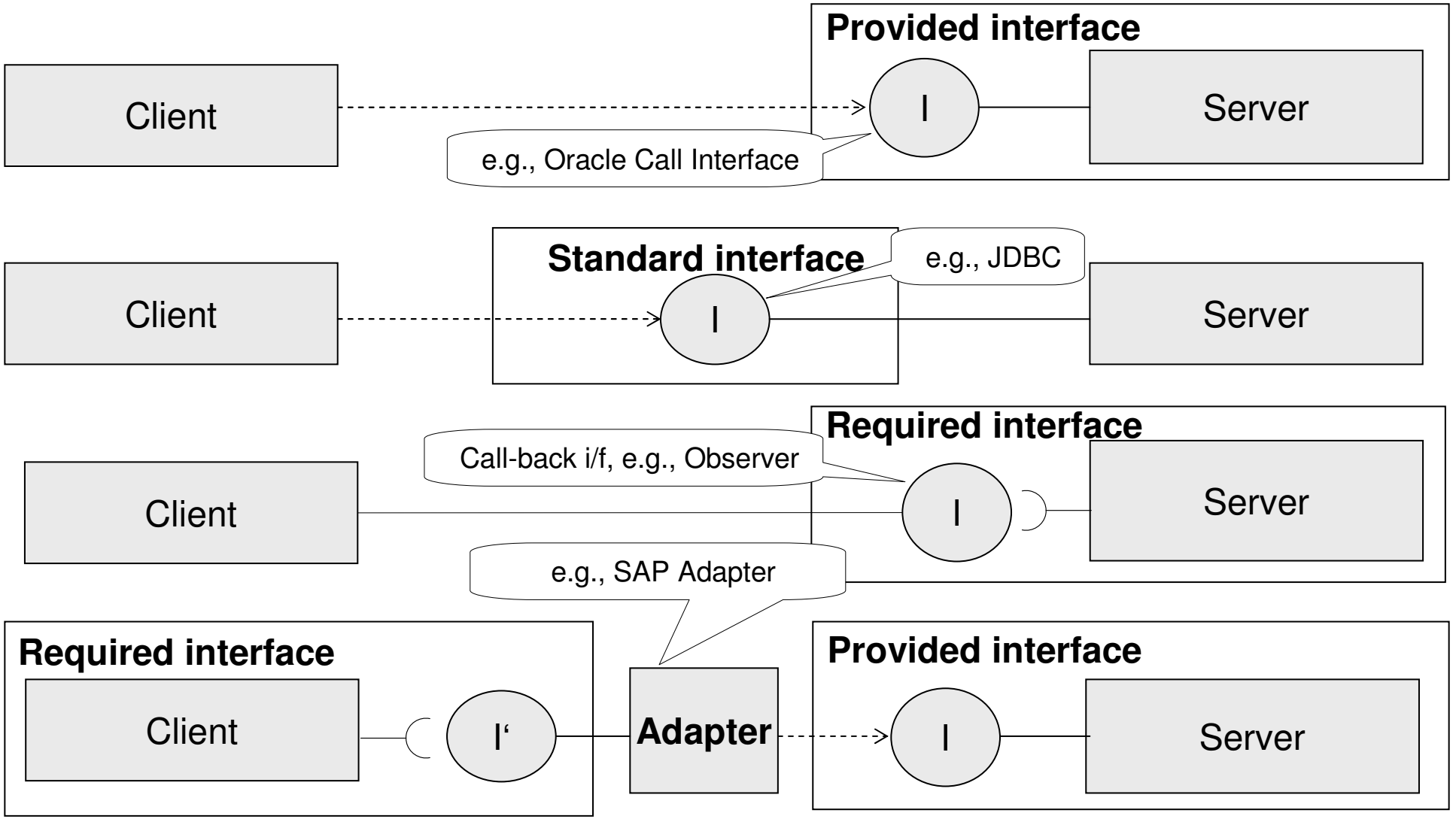


Excursion: UML-Notations for Interfaces

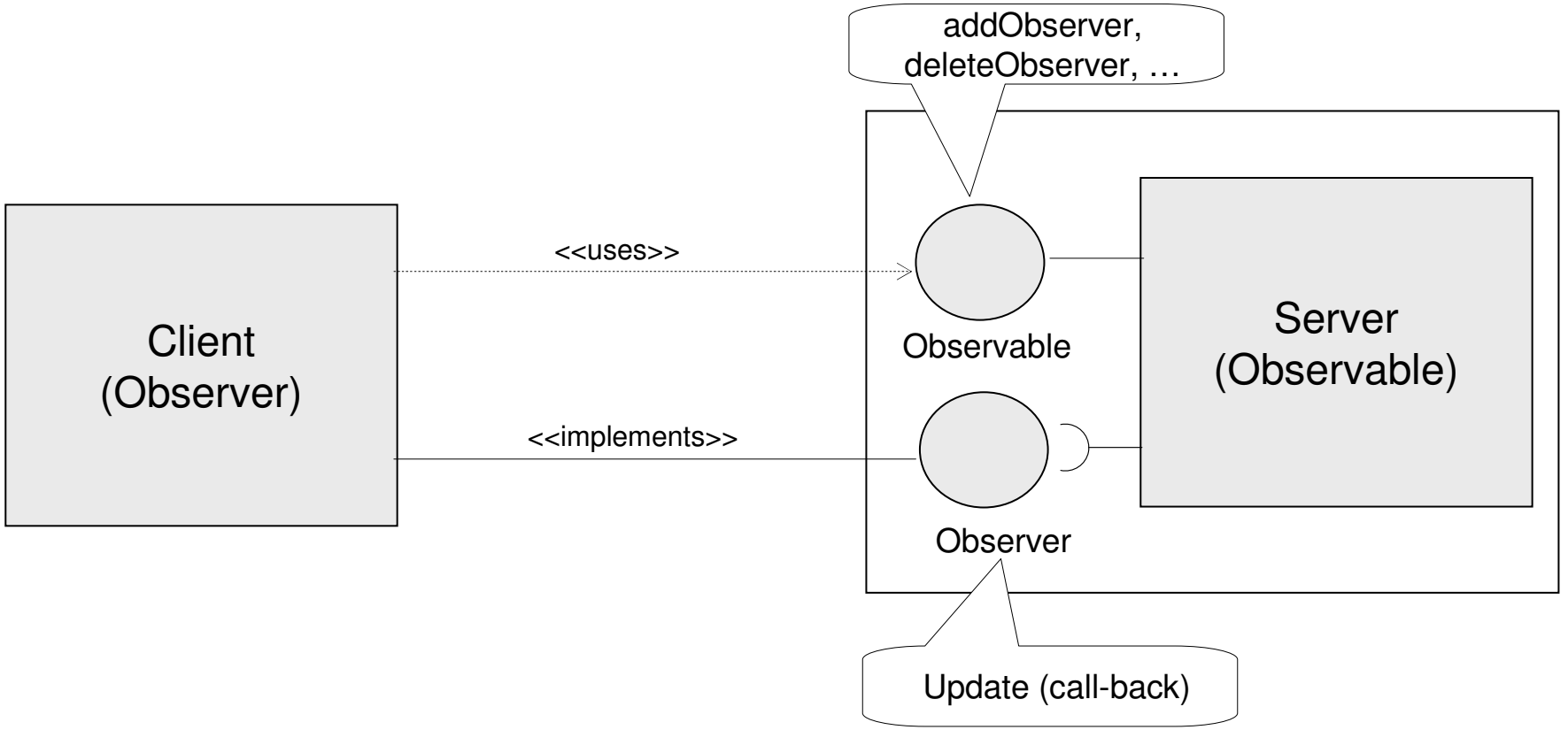


Ownership of interfaces (dependency: client → server)

Convention for this lecture: use of socket notation only if required interface is owned by component



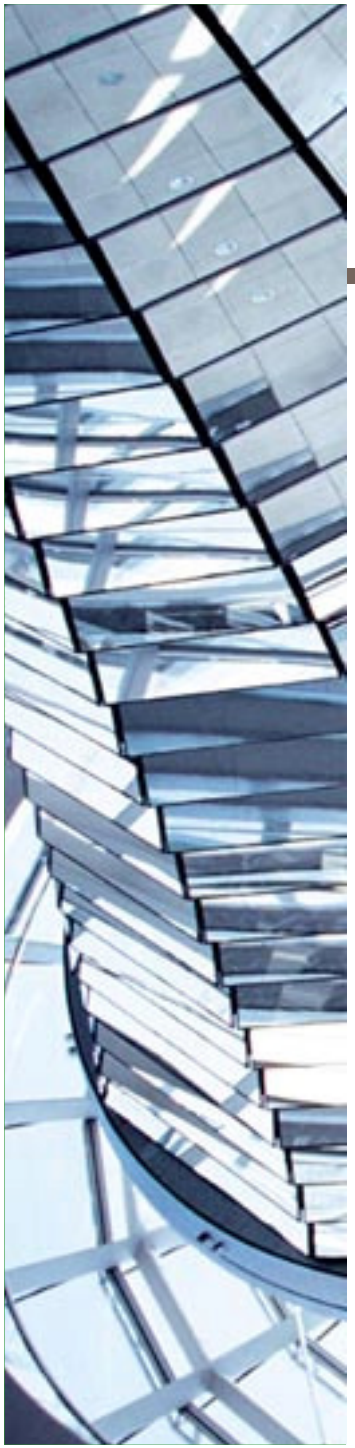
Example: Observer pattern



Principle inversion of control

Server (observable) exports provided as well as required interface

→ Bidirectional invocation but unidirectional dependency (observer → observable)



Agenda

Reference architecture

➔ **Authentication**

Authorization

Sequences

Interface Authentication

quasar.authorization

Interface Authentication

```
public interface Authentication
```

[export, operation] Call interface for user authentication.

```
[uses
     User    // interface for an object to denote an authenticated user
]
```

Method Summary	
User	**authenticate**(java.lang.String username, java.lang.String password) [basicQuery] Authenticates username with password and returns User object if authentication is successful

Authentication.authenticate

Method Detail

Semi-formal notation for the specification of interfaces, inspired by QSL (Quasar Specification Language)

authenticate

```
User authenticate(java.lang.String username,
                  java.lang.String password)
```

Signature

Classification basicQuery, derivedQuery, command

[basicQuery] Authenticates username with password and returns User object if authentication is successful

Description of funtional and non-functional properties in prose

Parameters:

`username` - [in] account name of the user

`password` - [in] password to be checked

Parameter type in, out, inout

Returns:

user object [out] to be used for authorization

```
[
  post
    // valid userName and correct password
  error  authenticationFailed
    // Unknown username or wrong password
]
```

Formal or textual specification of preconditions,postconditions, and invariants

Specification of errors

Example implementation JAAS: javax.security.auth.login.LoginContext .login()

login

```
public void login()
           throws LoginException
```

Perform the authentication.

This method invokes the `login` method for each LoginModule configured for the *name* specified to the `LoginContext` constructor, as determined by the login `Configuration`. Each `LoginModule` then performs its respective type of authentication (username/password, smart card pin verification, etc.).

This method completes a 2-phase authentication process by calling each configured LoginModule's `commit` method if the overall authentication succeeded (the relevant REQUIRED, REQUISITE, SUFFICIENT, and OPTIONAL LoginModules succeeded), or by calling each configured LoginModule's `abort` method if the overall authentication failed. If authentication succeeded, each successful LoginModule's `commit` method associates the relevant Principals and Credentials with the `Subject`. If authentication failed, each LoginModule's `abort` method removes/destroys any previously stored state.

If the `commit` phase of the authentication process fails, then the overall authentication fails and this method invokes the `abort` method for each configured `LoginModule`.

If the `abort` phase fails for any reason, then this method propagates the original exception thrown either during the `login` phase or the `commit` phase. In either case, the overall authentication fails.

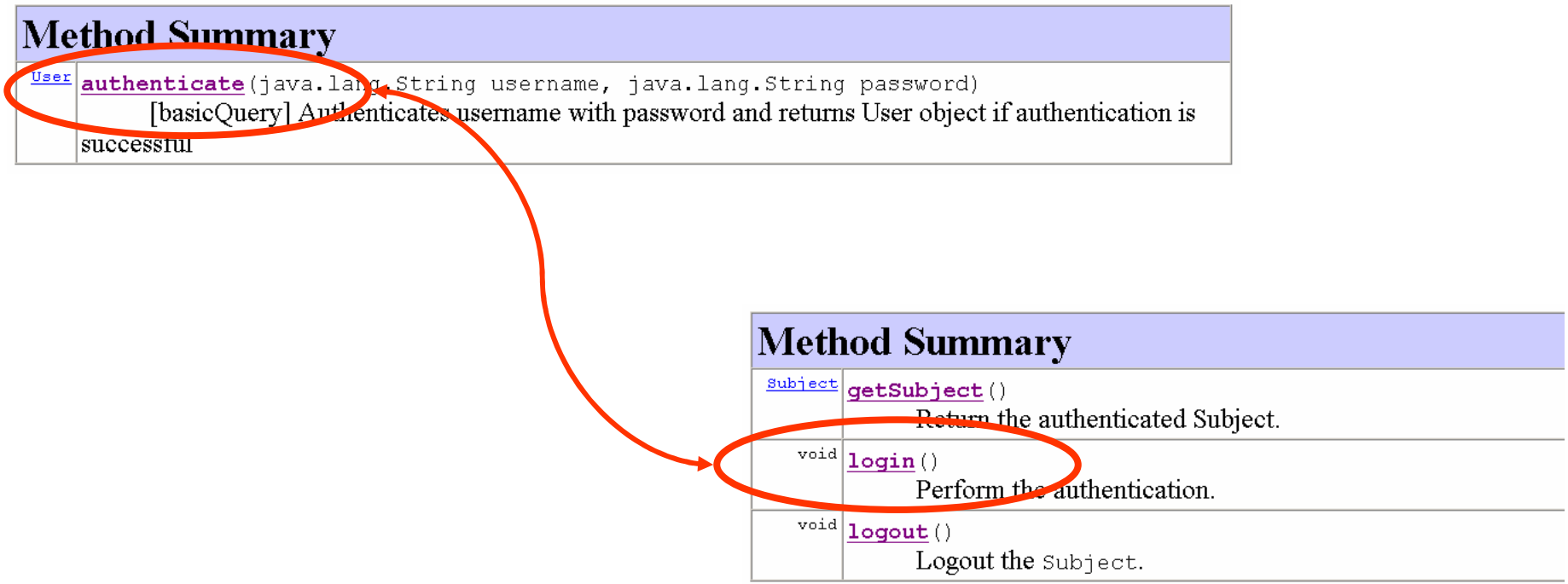
In the case where multiple LoginModules fail, this method propagates the exception raised by the first `LoginModule` which failed.

Note that if this method enters the `abort` phase (either the `login` or `commit` phase failed), this method invokes all LoginModules configured for the application regardless of their respective `Configuration` flag parameters. Essentially this means that `Requisite` and `Sufficient` semantics are ignored during the `abort` phase. This guarantees that proper cleanup and state restoration can take place.

Throws:

`LoginException` - if the authentication fails.

Correlation between reference interface and realization

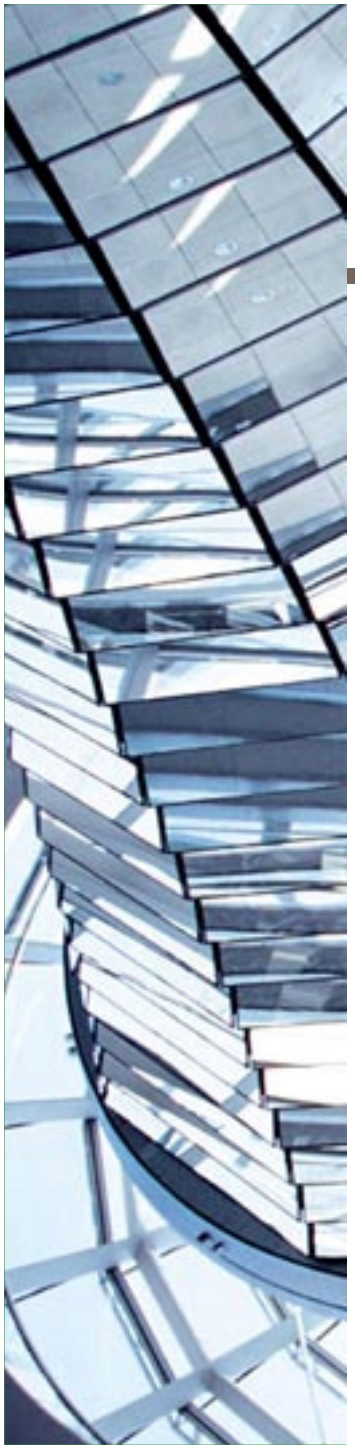


Method Summary

User	**authenticate**(java.lang.String username, java.lang.String password) [basicQuery] Authenticates username with password and returns User object if authentication is successful

Method Summary

Subject	**getSubject**() Return the authenticated Subject.
void	**login**() Perform the authentication.
void	**logout**() Logout the Subject.



Agenda

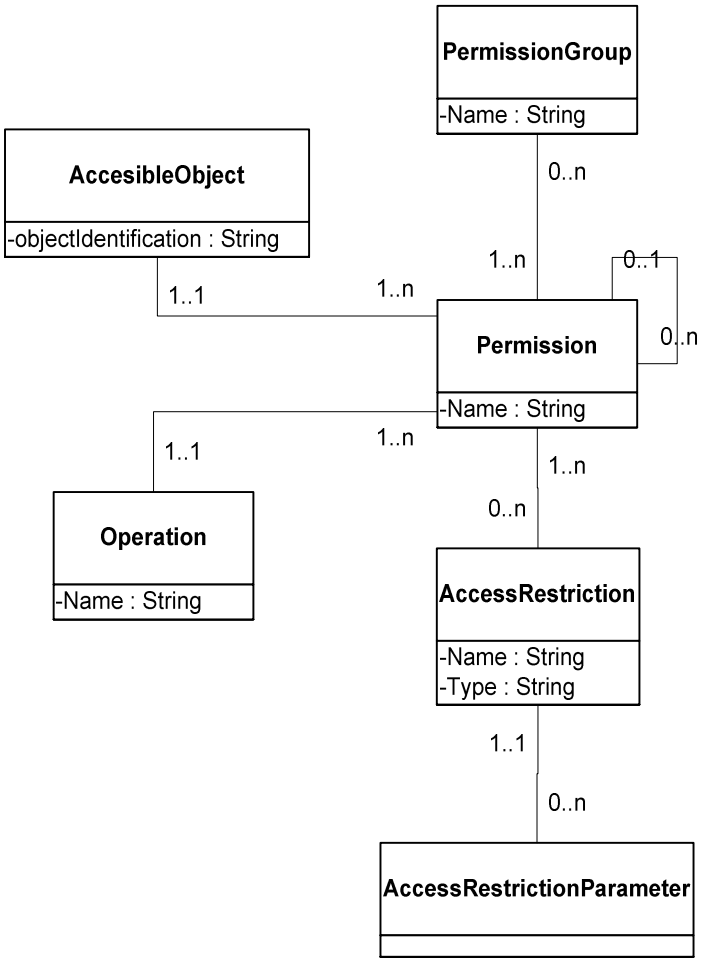
Reference architecture

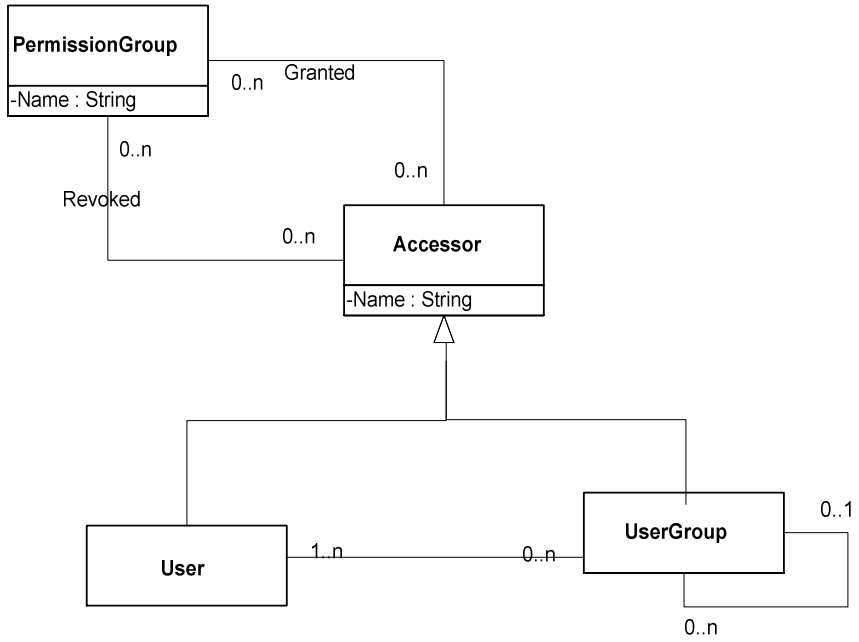
Authentication

➔ **Authorization**

Sequences

Authorization logic can be extremely complex





Interface Authorization: extremely simple

quasar.authorization

Interface Authorization

```
public interface Authorization
```

[export, operation] Call interface for authorization proper, i.e., checking access rights.

```
[uses
     User         // interface for objects to denote authenticated users
     Operation    // enumeration of allowed operations on a resource
]
```

Method Summary

boolean	**mayPerform**(User user, java.lang.Object resource, Operation operation) [basicQuery] Returns true if user is allowed to perform operation on resource, false otherwise

Authorization.mayPerform

Method Detail

mayPerform

```
boolean mayPerform(User user,
                   java.lang.Object resource,
                   Operation operation)
```

[basicQuery] Returns true if user is allowed to perform operation on resource, false otherwise

Parameters:

`user` - [in] user to perform the operation on the resource
`resource` - [in] resource that is being accessed
`operation` - [in] operation to be performed on the resource

Returns:

flag [out] to denote whether access is permitted

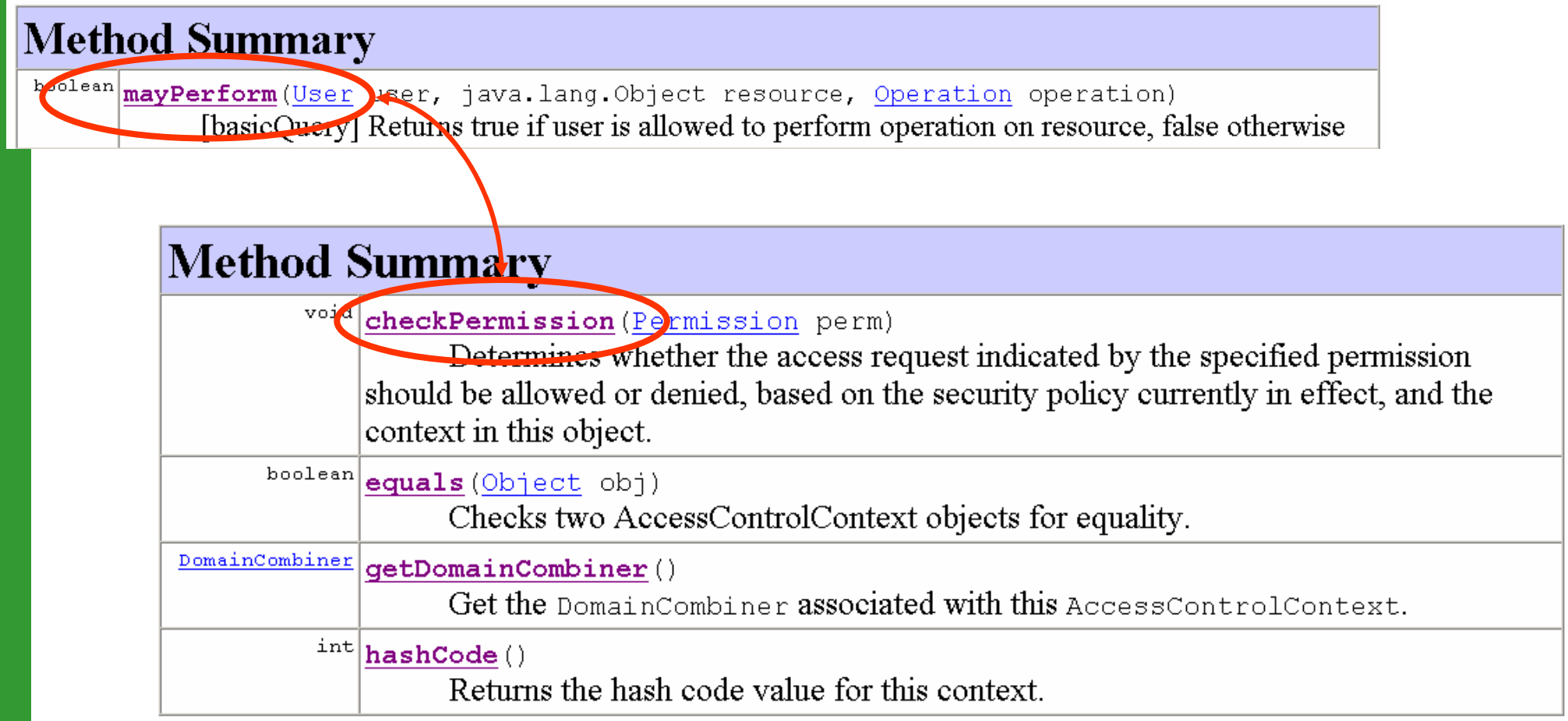
```
[
pre
  exists String userName, String password, Time t : t < now :
     invoke(this.getInterface(Authentication).authenticate(userName, password), t) == user
     // user must be authenticated, i.e., must have been the return value of
     // Authentication.authenticate of this Authorization component

error  unmanagedResource
     // resource unknown to authorization component
     // or operation not applicable to resource
```

Example Implementation JAAS: java.security.AccessControlContext

Method Summary	
void	**checkPermission**(Permission perm) Determines whether the access request indicated by the specified permission should be allowed or denied, based on the security policy currently in effect, and the context in this object.
boolean	**equals**(Object obj) Checks two AccessControlContext objects for equality.
DomainCombiner	**getDomainCombiner**() Get the DomainCombiner associated with this AccessControlContext.
int	**hashCode**() Returns the hash code value for this context.

Correlation between reference interface and realization

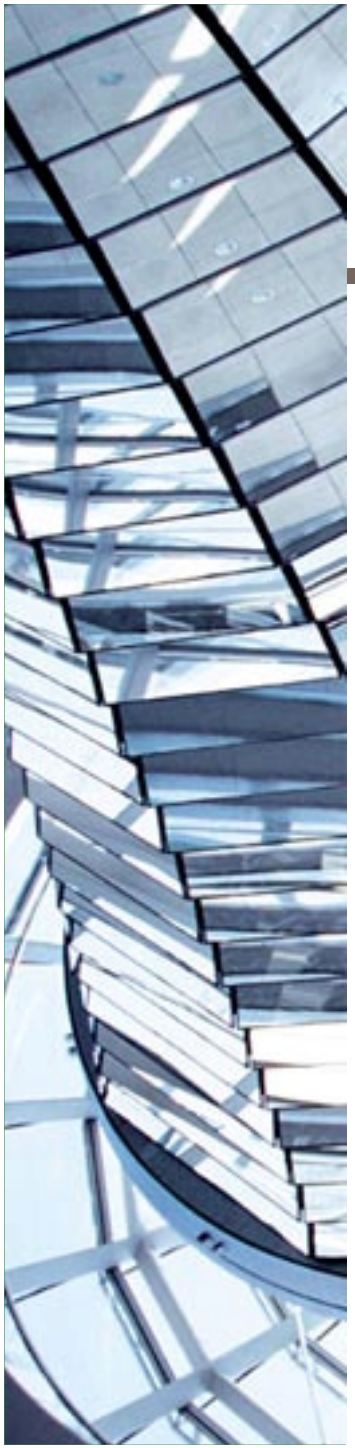


Method Summary

boolean	**mayPerform**(User user, java.lang.Object resource, Operation operation) [basicQuery] Returns true if user is allowed to perform operation on resource, false otherwise

Method Summary

void	**checkPermission**(Permission perm) Determines whether the access request indicated by the specified permission should be allowed or denied, based on the security policy currently in effect, and the context in this object.
boolean	**equals**(Object obj) Checks two AccessControlContext objects for equality.
DomainCombiner	**getDomainCombiner**() Get the DomainCombiner associated with this AccessControlContext.
int	**hashCode**() Returns the hash code value for this context.



Agenda

Reference architecture

Authentication

Authorization

➔ **Sequences**

Sequence: authenticate and authorize