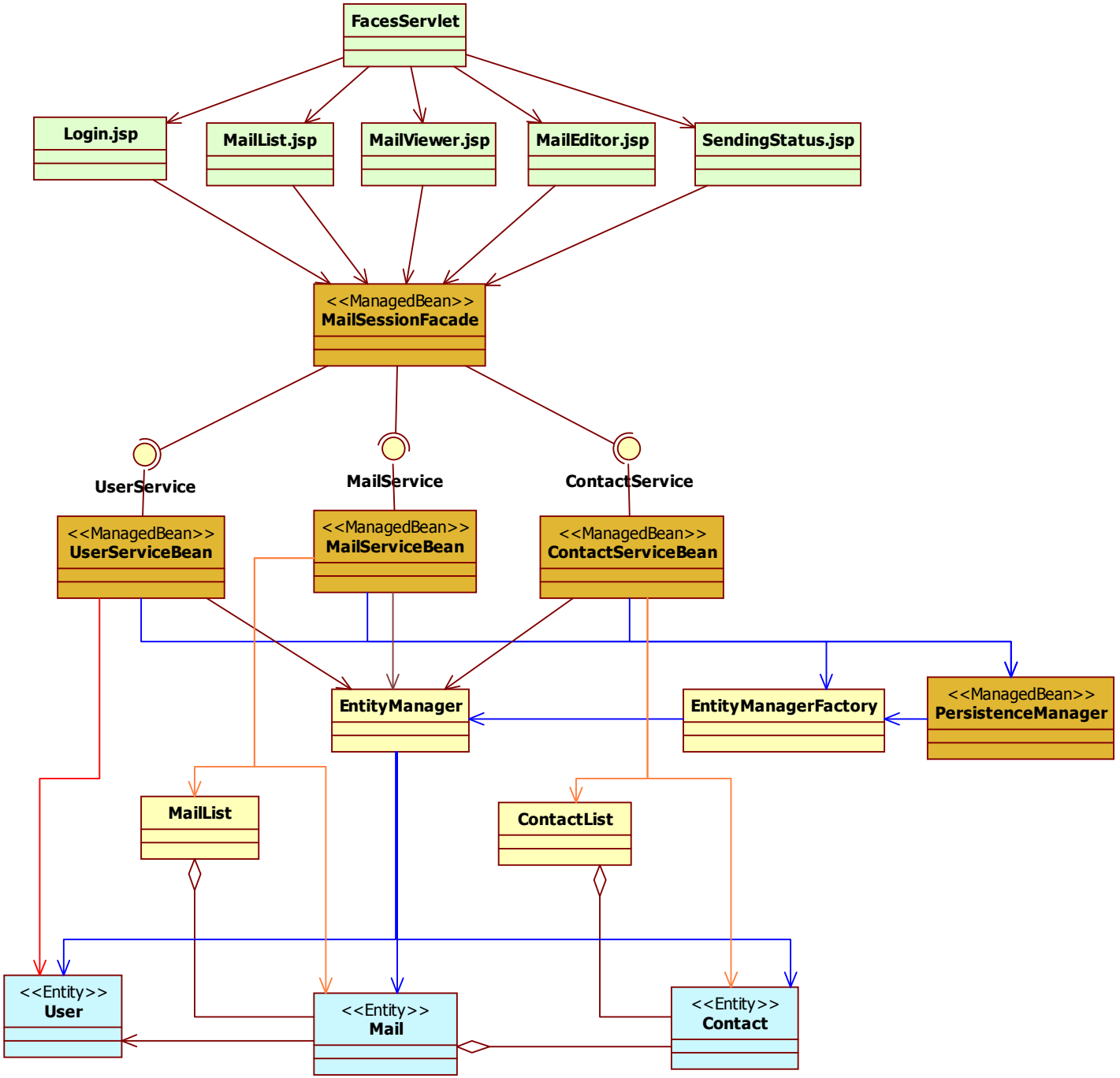
Authentifizierung einer JSF-Anwendung

Gesamt-Architektu



Authentifizierung

- Login - Mindestabsicherung gegen Missbrauch
  - Eigenschaft der Session
  - Verwaltung durch den Container
- Rechteverwaltung:
  - Benutzerrollen
  - Zugriffsrechte auf Resourcen rollenabhängig
  - Einfachste Form: Resource = Seite
  - Detaillierung bis zum einzelnen Interaktionselement Datenbankspalte möglich
- Verschiedene Techniken
  - Basis-Authentifizierung des Seitenzugriffs durch den Container
  - Frei programmierbare Überprüfung durch JSF-PhaseListener
  - Explizite Programmierung in allen entsprechenden Aktionen
    - nicht zu empfehlen!

Container-basierte Authentifizierung

- Container sieht vor:
  - Definition von Benutzern mit Name und Passwort
  - Zuordnung von Benutzern zu Rollen
  - Rollenabhängige Freigabe von Ressourcen
- Mögliche Ressourcen:
  - Verzeichnisse (Pfade)
  - Dateien
- Zur Laufzeit:
  - Login-Anforderung und –Prüfung bei Erstanforderung einer gesicherten Ressource
  - Login-Prüfung bei weiteren Anforderungen

Benutzer- und Rollen-Spezifikationen (Tomcat)

- In Tomcat Spezifikation von Benutzern und Rollen in der Datei tomcat-users.xml (unverschlüsselt)

```
<tomcat-users>
      <role rolename="tomcat"/>
      <role rolename="role1"/>
      <role rolename="manager"/>
      <role rolename="admin"/>
      <user username="ide" password="(generated password)"
                                    roles="manager,admin"/>
      <user username="tomcat" password="tomcat" roles="tomcat"/>
      <user username="role1" password="tomcat" roles="role1"/>
      <user username="both" password="tomcat" roles="tomcat,role1"/>
</tomcat-users>
```

Benutzerspezifikation (Glassfish)

- Eingabe von Usernamen und Passwörtern über die Admin-Konsole des Servers:

Rollen-Spezifikationen (Glassfish)

- Zuordnung von Usernamen zu Rollen in sun-web.xml (auch über einen Wizard)

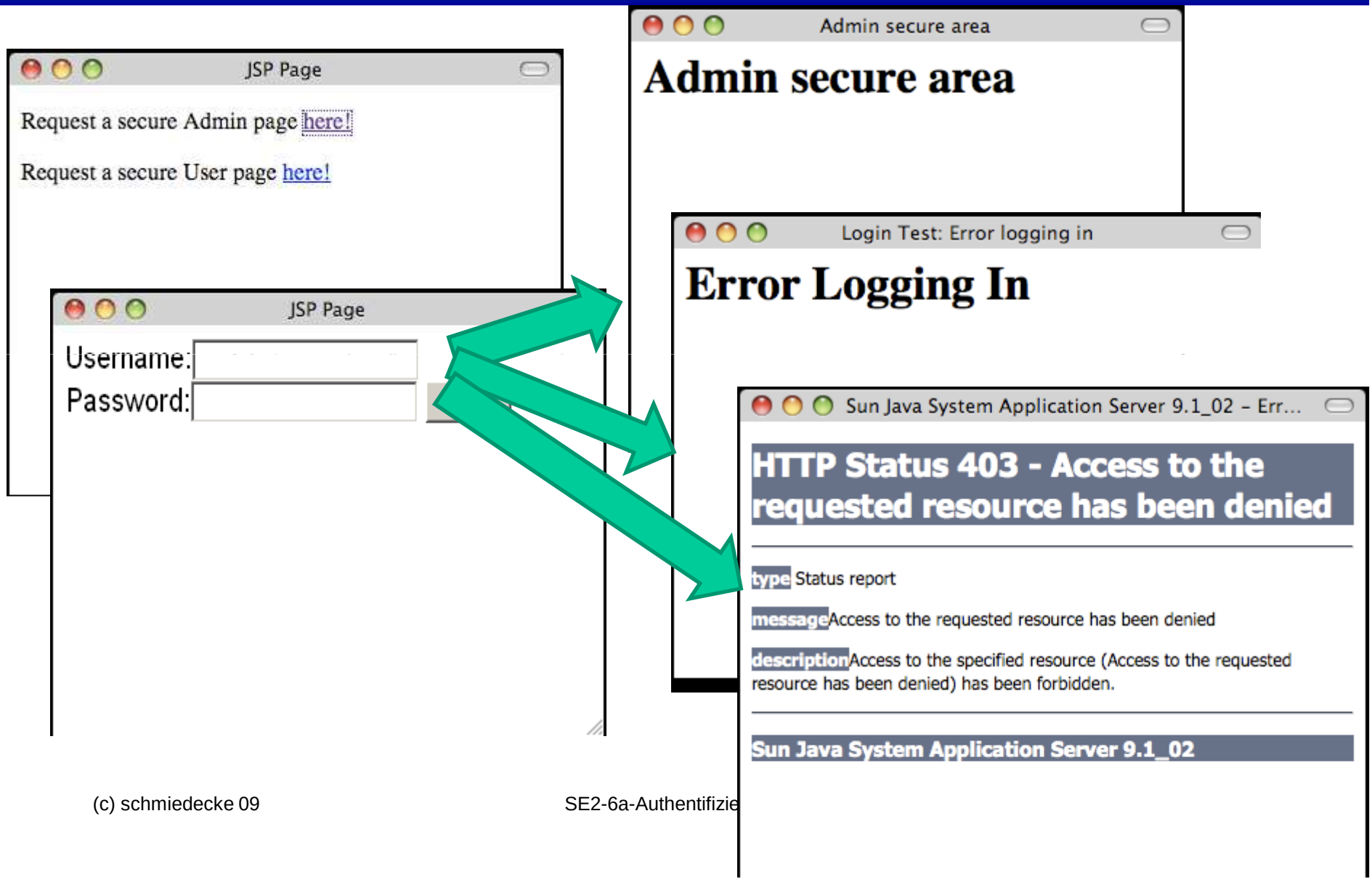
```
<security-role-mapping>
   <role-name>Admin</role-name>
   <principal-name>admin</principal-name>
</security-role-mapping>
```

Rollenbasierte Ressourcensicherung (beide Container)

- Zuordnung von Ressourcen zu Rollen in web.xml (Wizard)

```
<security-constraint>
    <display-name>AdminConstraint</display-name>
    <web-resource-collection>
      <web-resource-name>AdminResource</web-resource-name>
      <url-pattern>/secureAdmin/*</url-pattern>
      <http-method>GET</http-method>
      […]
      <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>Admin</role-name>
    </auth-constraint>
</security-constraint>
```

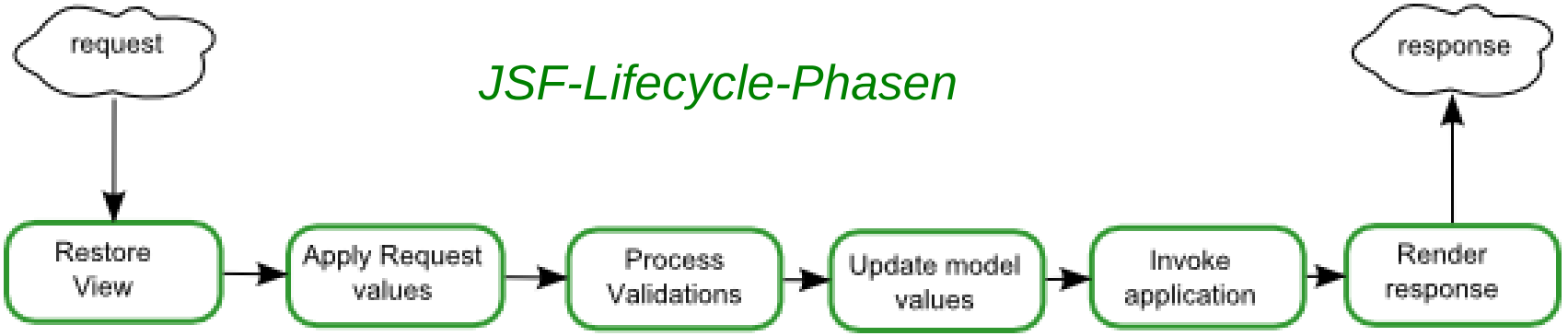
Anforderung einer gesicherten Seite



Benutzerdefinierte Authentifzierung mit PhaseListenern

- Authentifizierung ist eine „Querschnittsaufgabe“ – das klingt nach aspektorientierter Programmierung

- PhaseListener bieten ein verwandtes Konzept:
  - Der JSF-Lifecycle ist in abgegrenzte Phasen eingeteilt
  - Vor und nach jeder Phase wird ein Ereignis erzeugt, für das ein PhaseListener registriert werden kann
  - Damit werden PhaseListener querschnittsartig vor oder nach bestimmten Programmteilen ausgeführt



Phasengebundene Authentifizierung: Prinzip

- PhaseListener für die Authentifizierung:
  - Zu Beginn einer Seitenanfrage Login-Status-Prüfung
  - Typischerweise am Ende der ersten Phase:
    „after phase“ --- PhaseId.RESTORE_VIEW
  - In der Methode afterPhase(PhaseEvent ev) wird dann eine beliebige Überprüfung aufgerufen.
  - Bei Misserfolg wird zu einer Fehlerseite navigiert,
    entweder fest per „redirect“
    oder mit dem NavigationHandler entsprechend einer konfigurierten Navigation (flexibler)

Phasengebundene Authentifizierung: PhaseListener-Implementierung

```
import javax.faces.event.PhaseListener;
[…]
public class Authentificator implements PhaseListener {
   public PhaseId get PhaseId ()
    { return PhaseId.RESTORE_VIEW;   }

   public void afterPhase(PhaseEvent evt) {
      FacesContext ctx = evt.getFacesContext();
      if (! loginOK(ctx)) {
         NavigationHandler nav =
            ctx.getApplication().getNavigationHandler();
        nav.handleNavigation(ctx, null, „not authorized“);
      }
   }

   public boolean loginOK(FacesContext ctx) {….}
}
```

Phasengebundene Authentifizierung: Konfiguration in faces-config.xml

- Registrierung des PhaseListeners

```
<lifecycle>
    <phase-listener>
            com.example.event.Authenticator
    <phase-listener>
</lifecycle>
```

- Navigationsregel

```
<navigation-rule>
  <navigation-case>
      <from-outcome>not authorized</from-outcome>
       <to-view-id>/autherror.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Das war's (weitestgehend) in Sachen Systemarchitektur- und Entwurf