PSE Ausarbeitung

Phillip Wo
Benjamin Moser
Daniel Sommer

March 2, 2019

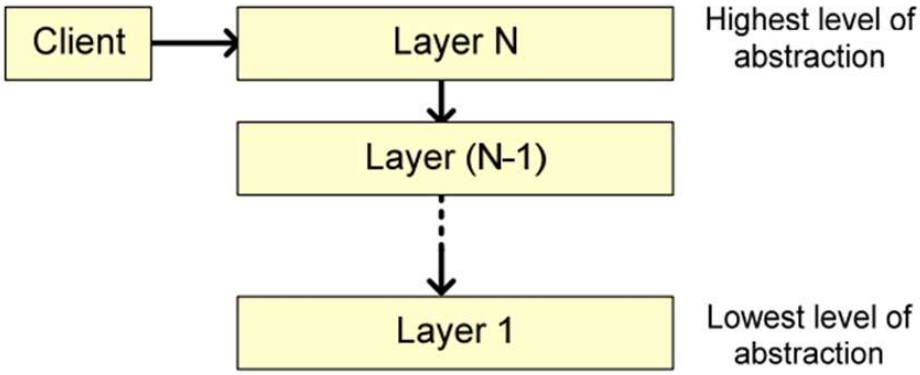
Contents

Part I
Pattern im Projekt

1 Layers Pattern

1.1 Erkläre die Funktionsweise + Skizze

- Client schickt eine Anfrage an Layer N
- Layer N reicht da er nicht vollständig alleine beantworten kann, Anfragen an darunterliegenden Layer weiter
- Eine Anfrage kann bei Bedarf auch in mehrere Anfragen an darunterliegende Layer geteilt werden
- dies wird immer weiter fortgesetzt bis Layer 1 erreicht ist
- dabei gehen Abhängigkeiten nur von oben nach unten

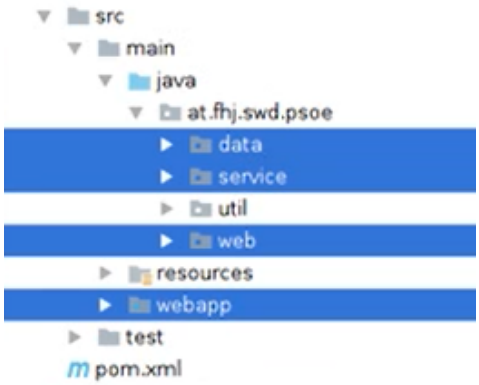


1.1.1 3 Schichten Architektur:

- Data Source Layer (data): Zugriff auf Daten, kümmert sich um Kommunikation mit anderen Systemen (z.B.: Datenbank)
  - beinhaltet DAO und DAOImpl » DocumentDAO, DocumentlibraryDAO
- Domain Layer(service): enthält Business Logik (Berechnungen, Datenvalidierung, ...)
  - beinhaltet
    - **Service Layer Pattern** (aka Session Fassade - siehe 3)
    - DTO » DocumentDTO
    - Mapper » DocumentMapper

```
public static Document toEntity(DocumentDTO documentDTO, Document
↪  document){};
public static DocumentDTO toDTO(Document document){};
```

- Presentation Layer(web): serverseitig, kümmert sich um Benutzerinteraktion
  - Controller (ViewHelper) » DocumentController, DocumentListController
  - View (WebApp)

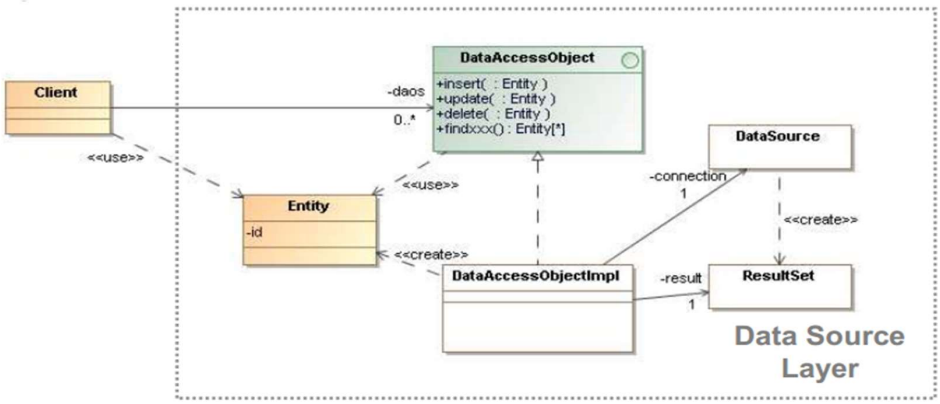


2 Data Access Object (DAO) Pattern

Befindet sich im Projekt in data und damit innerhalb des Data Layer.

2.1 Erkläre die Funktion + Skizze

- Client erstellt ein DAO Object und kann nach Entitäten suchen, einfügen, löschen, etc.
- das DAO selbst soll keine spezifischen Elemente enthalten (Entity Manager, SQL Exception -> stattdessen DAOException)
- dadurch entsteht eine Kapselung bei der die DAOImpl ohne den Client zu verändern ausgetauscht werden kann



```
@ApplicationScoped
public class DocumentDAOImpl implements DocumentDAO, Serializable {
    private static final long serialVersionUID = 1L;
    private static final Logger logger =
     ↪  LoggerFactory.getLogger(DocumentDAOImpl.class);
    @PersistenceContext
    private EntityManager entityMangaer;

    @Override
    public List<Document> findByCommunity(Community community) {...}

    @Override
    public List<Document> findByUser(User user) {...}

    @Override
    public void insert(Document document) {...}

    @Override
    public void delete(Document document) {...}

    @Override
    public Document findById(Long id) {...}
}
```

2.2 Nenne die Konsequenzen der Anwendung

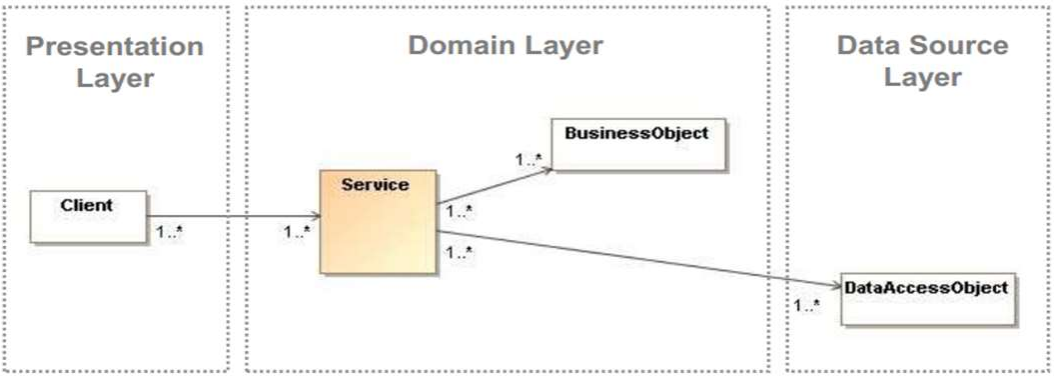
- Zugriff auf persistenten Speicher wird abstrahiert
- Details des Speichers werden versteckt
- ermöglicht einheitlichen Zugriff auf Daten
- entkoppelt Implementierung von Persistierung (Datenbank,...)
- ermöglicht Objektorientierte Ansicht des Speichers

3 Service Layer Pattern (auch Session Fassade - in unserem Projekt im Domain Layer

3.1 Erkläre die Funktion + Skizze

- Der Service Layer (Ordner "'service"' im Projekt) delegiert auf die Business Logik (Zeile 68 community.setDocumentlibrary) und zum DAO (z.B. Zeile 66)
- Bei wenig Logik wird zumindest Transaktions (Zeile 40), Error (ab Zeile 42) und Validierungshandling (ab Zeile 23) im Service erledigt

```
1       @Local(DocumentService.class)
2       @Remote(DocumentServiceRemote.class)
3       @Stateless
```



- dto
- impl
- mapper
- rest
- ActivitystreamService
- ActivitystreamServiceRemote
- AuthenticationException
- BusinessTripService
- BusinessTripServiceRemote
- CommunityService
- CommunityServiceRemote
- CurrentUserProvider
- DepartmentHierarchyService
- DepartmentService
- DocumentService

```
4   public class DocumentServiceImpl implements DocumentService,
    ↪ DocumentServiceRemote, Serializable {
5       private static final long serialVersionUID = -1L;
6       private static final Logger logger =
        ↪ LoggerFactory.getLogger(DocumentServiceImpl.class);
7
8       @Inject
9       private DocumentDAO documentDAO;
10      @Inject
11      private DocumentlibraryDAO documentlibraryDAO;
12      @Inject
13      private CommunityDAO communityDAO;
14      @Inject
15      private UserDAO userDAO;
16      @Inject
17      private MessageDAO messageDAO;
18      @Override
19      public DocumentDTO addDocument(Long communityID, String userID, byte[]
        ↪ data, String filename) {
20          Document addedDocument;
21          User user;
22
23          // Validierungshandling gefolgt von Error Handling
24          try {
25              if (communityID <= 0) throw new
                ↪ IllegalArgumentException("community must not be empty");
26              if (userID == null) throw new IllegalArgumentException("user must
                ↪ not be empty");
27              if (data == null) throw new IllegalArgumentException("uploaded
                ↪ file must not be empty");
28              if (filename == null) throw new
                ↪ IllegalArgumentException("filename must not be empty");
29
30              Documentlibrary documentlibrary =
                ↪ documentlibraryDAO.findByCommunityId(communityID);
31
32              //create a document library, if there isn't already one in the
                ↪ database
33              if (documentlibrary == null) {
34                  documentlibrary = addDocumentlibrary(communityID);
35              }
36
37              user = userDAO.getByUserId(userID);
38
39              addedDocument = new Document(documentlibrary, user, filename,
                ↪ data);
40              documentDAO.insert(addedDocument); // Transaktionshandling
41              logger.info(String.format("Document %s saved in database",
                ↪ filename));
42          // Error Handling
43          } catch (IllegalArgumentException iaex) {
44              String errorMsg = "Uploading file failed (illegal argument)";
45              logger.error(errorMsg, iaex);
```

```
46                  throw new ServiceException(errorMsg);
47
48              } catch (Exception ex) {
49                  String errorMsg = String.format("Uploading file %s failed.",
                        ↪  filename);
50                  logger.error(errorMsg, ex);
51                  throw new ServiceException(errorMsg);
52              }
53
54              String msgText = "Uploaded Document " + filename + " by user " +
                    ↪  user.getUserId();
55              addMessageToStream(communityID, user, msgText, addedDocument);
56              return DocumentMapper.toDTO(addedDocument);
57          }
58
59
60          private void addMessageToStream(Long communityID, User user, String text,
                ↪  Document document) {...}
61
62          private Documentlibrary addDocumentlibrary(Long communityID) {
63              logger.info("Create missing documentlibrary");
64              Community community;
65              Documentlibrary documentlibrary = new Documentlibrary();
66              documentlibraryDAO.insert(documentlibrary); // Delegation zum DAO
67              community = communityDAO.findById(communityID); // Delegation zum
                    ↪  DAO
68              community.setDocumentlibrary(documentlibrary); // Delegation zur
                    ↪  Business Logik (Entity)
69              communityDAO.update(community); // Delegation zum DAO
70              return documentlibrary;
71          }
72
73          @Override
74          public List<DocumentDTO> getDocumentsFromCommunity(Long communityID)
                ↪  {...}
75
76          @Override
77          public List<DocumentDTO> getDocumentsFromUser(String userID) {...}
78
79          @Override
80          public void removeDocument(Long documentID) {...}
81
82          @Override
83          public DocumentDTO getDocumentById(Long documentID) {...}
84      }
```

3.2 Nenne die Konsequenzen der Anwendung

- Reduzierung der Abhängigkeiten zwischen Presentation und Domain Layer
- Zentralisiertes Sicherheits und Transaktionshandling
- verbirgt vor Client Komplexität der Business Logik

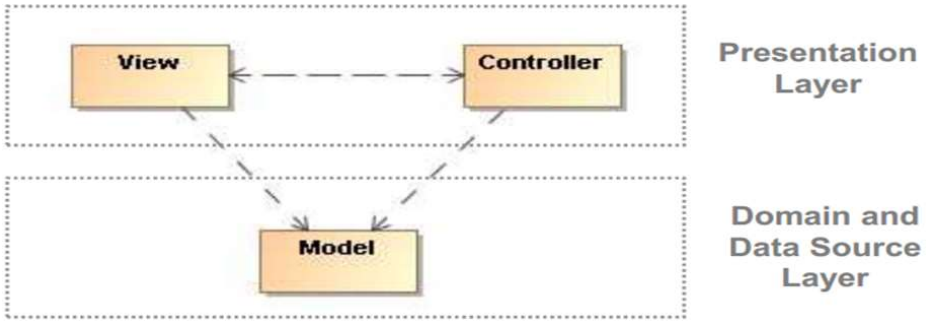
- stellt Client ein grobkörniges Interface zur Verfügung
- gut für Remote Aufrufe geeignet (weniger Aufrufe)

4 Model-View-Controller (MVC) Pattern

4.1 Erkläre die Funktion + Skizze

MVC unterteilt eine interaktive Applikation in drei Teile: Model, View und Controller.

- Controller und View befinden sich im Presentation Layer und haben gegenseitig Abhängigkeiten
- Das Model darf keine Abhängigkeiten haben (Controller und View hängen vom Model ab)



4.1.1 Model

- Es befinden sich Teile im Domain und Data Source Layer.
- Das Model enthält die Kernfunktionalität und Daten. (z.B.: Datenbankzugriff)
- Im Projekt wird dies durch die Ordner *service* und *data* repräsentiert

4.1.2 View

- Im Projekt im Ordner *webapp* zu finden.
- Enthält im Projekt xhtml Dateien zur Darstellung und User Interaktion

4.1.3 Controller

- Im Projekt sind Controllerklassen im Ordner *web* zu finden.
- Sie enthalten die Logik und behandeln Benutzereingaben

5 Front Controller

5.1 Erkläre die Funktion + Skizze

- Client schickt Request an Front Controller
- FC erfasst nur Infos die er für die weiter Delegation braucht
- FC gibt Request an entsprechenden ConcreteCommand oder View weiter
- es gibt zwei Implementierungsvarianten des Controller
  - Servlet
  - ConcreteCommand

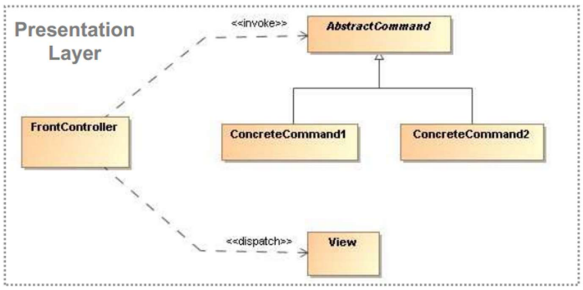
5.2 Servlet

- Im Projekt wurde der Front Controller in Form eines Servlet realisiert,
- die Einbindung erfolgt in der Konfigurationsdatei *src/main/webapp/WEB-INF/web.xml*,
- Servlet ist eine Java-API, welche auf einem Server betrieben wird,
- die Verarbeitung von Requests und Responses wird ermöglicht,
- JSF und JSP können darauf aufsetzen, in unserem Projekt wurde JSF verwendet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ↪  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   ↪  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   ↪  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
3  ...
4  <servlet>
5  <servlet-name>Faces Servlet</servlet-name>
6  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
7  <load-on-startup>1</load-on-startup>
8  </servlet>
9  <servlet-mapping>
10 <servlet-name>Faces Servlet</servlet-name>
11 <url-pattern>*.xhtml</url-pattern>
12 </servlet-mapping>
```

5.2.1 Java Server Faces (JSF)

- JSF basiert auf dem MVC-Pattern
- JSF-View-Code ist im Projekt im Ordner *src/main/webapp/** zu finden
- JSF-Logik befindet sich in den Java-Beans (im Projekt */src/main/java/at/fhj/swd/psoe/web/**)
- in unserem Projekt gibt es zu jeder xhtml-View eine eigene Controller-Klasse, welche dem ViewHelper-Pattern entspricht



- in unserem Projekt kommt PrimeFaces zum Einsatz (eine konkrete Implementierungsart von JSF => Einbindung in pom.xml)

```
1  <!-- Pfad: /src/main/webapp/community/documentManagement.xhtml -->
2
3  <?xml version='1.0' encoding='UTF-8' ?>
4  <!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   ↪  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5  <ui:composition xmlns="http://www.w3.org/1999/xhtml"
6  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
7  xmlns:h="http://xmlns.jcp.org/jsf/html"
8  xmlns:p="http://primefaces.org/ui"
9  xmlns:f="http://xmlns.jcp.org/jsf/core"
10 template="communityTemplate.xhtml">
11 <ui:define name="communityContent">
12 <h1>#{msg.document_manage_title}</h1>
13 <f:metadata>
14 <f:viewAction action="#{documentListController.loadDocumentsFromCommunity()}" />
15 </f:metadata>
16
17 <h:form id="doclistform">
18 <p:commandButton value="Refresh list"
   ↪  actionListener="#{documentListController.loadDocumentsFromCommunity()}"
   ↪  update="@form doclistform"></p:commandButton>
19 <p:dataTable id="doclisttable" value="#{documentListController.communityDocuments}"
   ↪  var="docs">
20 <p:column class="documenttimecolumn"
   ↪  headerText="#{msg.document_uploaded}">#{docs.createdTimestamp}</p:column>
21 <p:column class="documenttimecolumn"
   ↪  headerText="#{msg.label_userid}">#{docs.user.userId}</p:column>
22 <p:column headerText="#{msg.label_filename}">#{docs.filename}</p:column>
23 <p:column headerText="" class="documentbuttoncolumn">
24 <p:commandButton value="#{msg.button_download}" ajax="false"
25 onclick="PrimeFaces.monitorDownload(start, stop);">
26 <p:fileDownload value="#{documentController.downloadDocument(docs.id)}"/>
27 </p:commandButton>
28 </p:column>
29 <p:column headerText="" class="documentbuttoncolumn">
30 <p:commandButton id="btnDel" value="#{msg.button_delete}"
31 actionListener="#{documentController.removeDocument(docs.id)}"
32 update="@form doclistform">
```

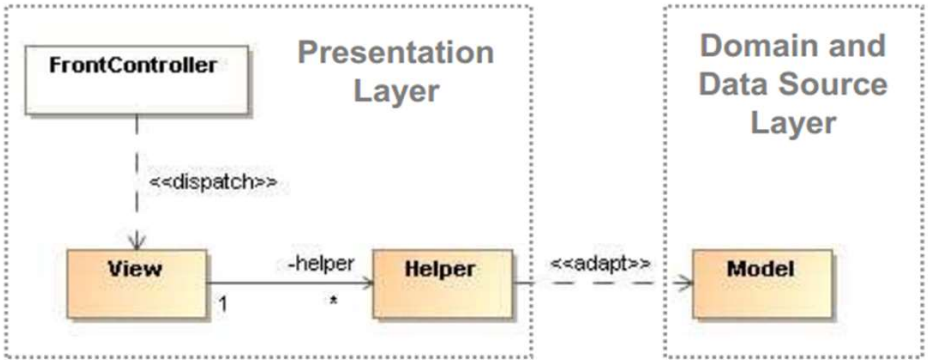
```
</p:commandButton>
</p:column>
</p:dataTable>
</h:form>
<h:form id="formdocupload" enctype="multipart/form-data">
<p:fileUpload id="fileupload"
dragDropSupport="false"
update="@form doclistform"
fileUploadListener="#{documentController.uploadDocument}"
allowTypes="/(\.|\/)(pdf|jpe?g|docx)\$/" sizeLimit="5000000"
mode="advanced" label="Add document (.pdf .jpg .docx)">
</p:fileUpload>
</h:form>
<p:messages id="feedbackBox" severity="info,error" showDetail="true"
    showSummary="false">
<p:autoUpdate/>
</p:messages>
</ui:define>
</ui:composition>
```

5.3 Nenne die Konsequenzen der Anwendung

- es muss nur EIN (Front) Controller konfiguriert werden
- da bei jedem Request ein neues Command Objekt erzeugt wird ist Thread-Safety nicht notwendig
- da nur EIN Controller sind auch Erweiterungen durch z.B.: Decorator einfach (auch zur Laufzeit)

6 View Helper (*/src/main/java/at/fhj/swd/psoe/web/**)

6.1 Erkläre die Funktion + Skizze



- View (xhtml-Dateien im Ordner */src/main/webapp/**) delegiert Aufgaben an Helper (z.B. DocumentController im Ordner web)

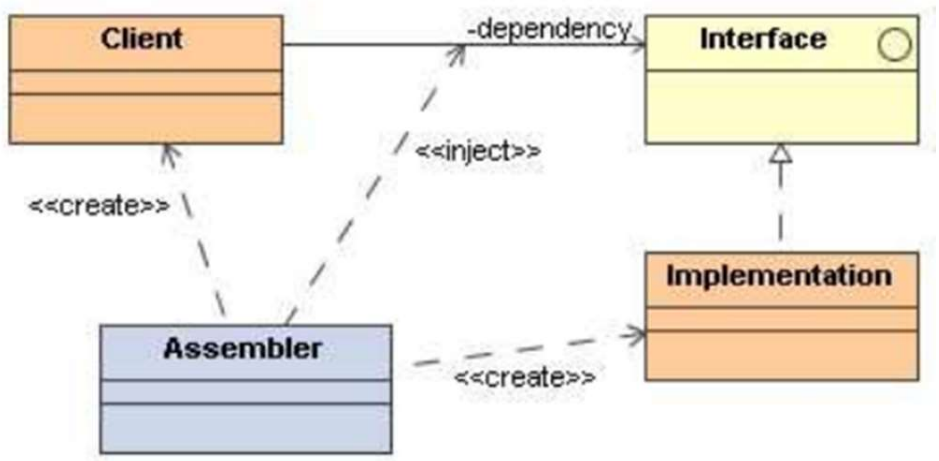
- Helper adaptieren View zu Model (Klassen in den Ordnern *src/main/java/at/fhj/swd/psoe/service/** und *src/main/java/at/fhj/swd/psoe/data/**)
- in View befindet sich HTML Code im ViewHelper Java Code zur Aufbereitung der Daten (+ wenig HTML)

6.2 Nenne die Konsequenzen der Anwendung

- kapselt Design-Code in View und View-Processing-Code Logik in Helper
- steigert Wiederverwendbarkeit, Wartbarkeit und Strukturierungsqualität der Anwendung
- vereinfacht Tests (Helperfunktionen ohne View)
- bessere Trennung zwischen
  - Presentation und Data Source Layer
  - Entwickler und Designer

7 Dependency Injection (CDI-Framework in pom.xml im Projekt)

7.1 Erkläre die Funktion + Skizze



- Grundidee sind loose gekoppelte Objekte
- Objekte werden mittels externem Assembler verknüpft
- Abhängigkeiten bestehen nur auf Interfaces
- Assembler Objekt (Framework) erzeugt die Interface-Implementierungen (z.B.: durch Factory)

- Es wird zwischen Constructor Injection und Setter Injection unterschiedlichen

```
1           // Constructor Injection
2       puplic class Client
3       {
4           private Interface iface;
5       public Client(Interface iface)
6       {
7           this.iface = iface;
8       }}
9
10          // Setter Injection
11      puplic class Client
12      {
13          private Interface iface;
14      public setIface(Interface iface)
15      {
16          this.iface = iface;
17      }}
```

- Im Spring Context:
  - Dependency Injection mit XML-Datei
  - alle Beans sind dort gelistet und werden verknüpft
  - Context wird geladen damit alles verknüpft ist
  - erspart Factories

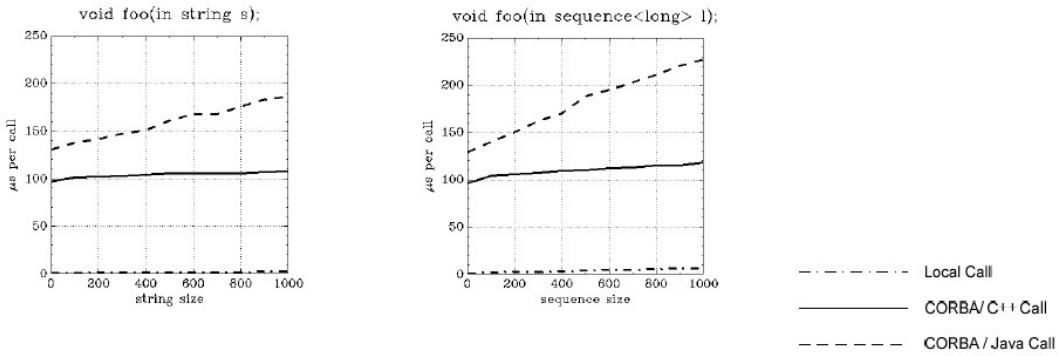
7.2 Nenne die Konsequenzen der Anwendung

- loose gekoppelte Objekte
- Referenzen nur noch auf Interfaces
- hohe Flexibilität (Strategy, Proxy,..)
- bessere Erweiterbarkeit und Testbarkeit
- bei Spring kann Dependency Injection mittels XML oder Annotation erfolgen
  - Vorteil Annotation: Typ-Sicherheit (Tippfehler passieren schnell im XML)
  - Nachteil Annotation: nicht so flexibel wie XML

8 Data Transfer Object (DTO) Pattern

8.1 Erkläre die Funktion (Skizze - ein Grund für DTO)

- Transportiert Daten zwischen Prozessen um Remote Methodenaufrufe zu minimieren
- besteht aus Fields, Getter und Setter
- fasst Daten verschiedener Objekte zusammen die vom Remote Objekt benötigt werden
- ev. Map, Record Set, ...



8.2 Verwendung im Projekt in *src/main/java/at/fhj/swd/psoe/service/dto/**

```
1  package at.fhj.swd.psoe.service.dto;
2
3  import java.io.Serializable;
4  import java.util.Date;
5
6  public class DocumentDTO implements Serializable {
7
8          private static final long serialVersionUID = 4016557982897997689L;
9
10         private Long id;
11         private Long documentlibraryID;
12         private String filename;
13         private UserDTO user;
14         private byte[] data;
15         private Date createdTimestamp;
16
17         public DocumentDTO() {}
18
19         public Long getId() {
20                 return id;
21         }
22
23         public void setId(Long id) {
24                 this.id = id;
25         }
26
27         public Long getDocumentlibraryID() {
28                 return documentlibraryID;
29         }
30
31         public void setDocumentlibraryID(Long documentlibraryID) {
32                 this.documentlibraryID = documentlibraryID;
33         }
34
35         public String getFilename() {
36                 return filename;
37         }
38
```

```
39          public void setFilename(String filename) {
40                  this.filename = filename;
41          }
42
43          public UserDTO getUser() {
44                  return user;
45          }
46
47          public void setUser(UserDTO user) {
48                  this.user = user;
49          }
50
51          public byte[] getData() {
52                  return data;
53          }
54
55          public void setData(byte[] data) {
56                  this.data = data;
57          }
58
59          public Date getCreatedTimestamp() {
60                  return createdTimestamp;
61          }
62
63          public void setCreatedTimestamp(Date createdTimestamp) {
64                  this.createdTimestamp = createdTimestamp;
65          }
66
67          @Override
68          public String toString() {
69                  return "DocumentDTO{" +
70                          "id=" + id +
71                          ", documentlibraryID=" + documentlibraryID +
72                          ", filename='" + filename + '\'' +
73                          '}';
74          }
75
76          @Override
77          public boolean equals(Object o) {
78                  if (this == o) return true;
79                  if (!(o instanceof DocumentDTO)) return false;
80
81                  DocumentDTO that = (DocumentDTO) o;
82
83                  return id.equals(that.id);
84          }
85
86          @Override
87          public int hashCode() {
88                  return id.hashCode();
89          }
90  }
```

8.3 Nenne die Konsequenzen der Anwendung

- kapselt und versteckt
- nimmt Komplexität
- steigert Effizienz da weniger Aufrufe über Remotegrenze

9 Page-Object-Pattern

PageObjectPattern
HTML – wrappen mit JavaCode, um es zu manipulieren
GUI-Test-Klasse und Page-Object
Über die Page-Object-Klasse manipuliere ich das HTML-Dokument
Bei HTML-Änderung muss ich nur Page-Objekt ändern und ansonsten nichts angreifen (Verkapselung)

10 Remote

11 Beschreibe die Unterschiede zwischen lokalem und Remote Interface Design

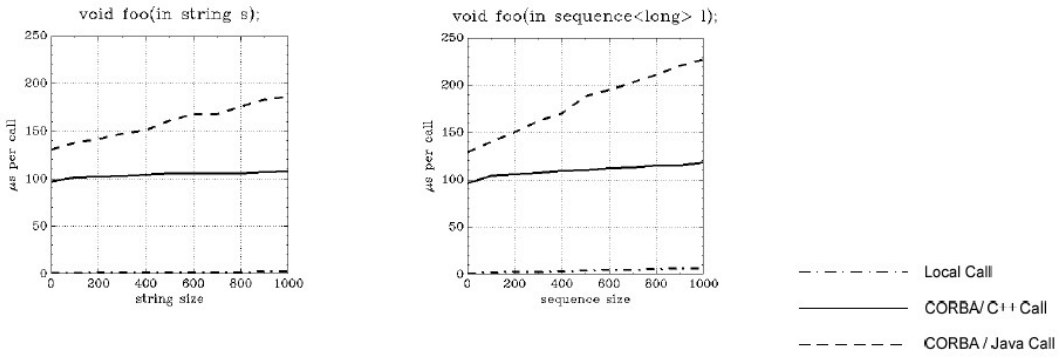
- Aufrufe innerhalb des Prozesses sind schneller als über Prozessgrenzen
- lokale Interfaces sind möglichst fein-granular während Remote Interfaces grob-granular sein müssen (weniger Aufrufe - Effizienz)
- viele kleine Aufrufe mit wenigen Daten sind "teuer" (Latenz durch Verbindungsherstellung)

12 Beschreibe drei Situationen wo Multiple Prozesse in Applikationen verwendet werden müssen

- Trennung zwischen Clients und Servern in Business Software
- Trennung zwischen server-basierter Applikationssoftware und Datenbank (SQL ist als Remote Interface designed, daher sind hier schnelle Abfragen möglich)
- Trennung wegen unterschiedlichen Anbietern oder Programmiersprachen

13 Beschreibe das folgende Diagramm. Was können wir daraus für das Design von Remote Interfaces folgern?

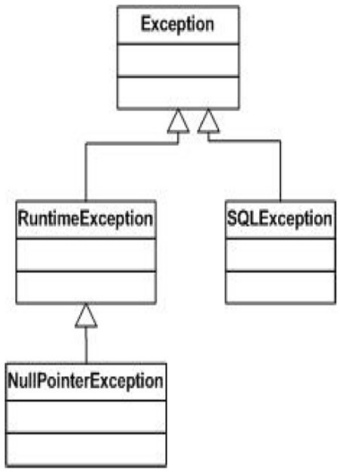
- speziell bei "teuren" Remote Calls ist es empfehlenswert weniger Calls mit großen Datenmengen anstatt vielen Calls mit wenigen Daten zu machen
- dieser Gedanke befürwortet auch den Einsatz von DTO um Calls und Daten zu bündeln



14 Exception Handling

15 Beschreibe den Unterschied zwischen Checked und Runtime Exceptions in Java (inkl. Klassendiagramm)

- Checked Exceptions (z.B. SQL-Exception) leiten von Exception Klasse ab und müssen behandelt werden (trows - catch)
  - Verwendung für Probleme die durch User behoben werden können (alternative Aktion)
- Unchecked Exceptions (z.B. NullPointerException) leiten von RuntimeException ab
  - Verwendung für technische Probleme (User kann nichts machen außer neu starten)



16 Beschreibe einen Use Case für eine Checked Exceptions in Java

- eine Netzwerkübertragung schlägt fehl - es ist vorgesehen, dass der Applikations-User dies neu anstoßen kann

17 Beschreibe einen Use Case für eine Runtime Exceptions in Java

- Die Datenbank ist beschädigt - die Exception geht durch alle Layer erst mit Implementierungsspezifischer Exception später mit Runtime ohne Stacktrace (Sicherheit) bis zum User.

18 Beschreibe 5 Best Practice Beispiele beim Einsatz von Exceptions

- Exceptions nicht für Programmflusskontrolle verwenden (schlechte Performance)
- offene Ressourcen schließen (try-with-resources bzw. close im finally)
- selbst erstellte Exceptions auch mit nützlichen Infos ausstatten
- Implementierungsspezifische Exceptions nicht bis zum User durchwerfen (stattdessen catch + trow RuntimeException)
- dokumentieren mit @trows im DOC, testen mit JUnit

19 Beschreibe 5 Exception Handling Anti Pattern

- Log and Trow (nie beides: entweder, oder)
- Trowing Exception bzw. catch Exception (spezifischere anstatt Basisklasse verwenden)
- Destructive Wrapping (wenn bei catch + trow = wrapping nicht die Original Exception weitergegeben wird)
- Log and return Null (provoziert an einer anderen Stelle eine NullPointerException)
- Catch and Ignore
- Unsupported Operation return Null (besser UnsupportedOperationException)

20 Logging

21 Nenne die Nachteile von debugging mit printf() sowie die Vorteile die Logging Frameworks wie log4j bieten

21.1 Nachteile printf()

- Produktiv-Code wird überfüllt -> erschwert Lesbarkeit
- Consolenausgabe wird bei vielen prints auch schnell unübersichtlich
- im Falle eines vorzeitigen Absturzes können Ausgabedaten verloren gehen
- Performance bei vielen Logprints

21.2 Vorteile Logging mittels Framework (z.B.: log4j)

- Nutzt ein einheitliches Format / Konventionen
- logging kann optional an und ausgeschalten werden
- durch verschiedene Log-level können Logs gefiltert erstellt werden
- Layout für Ausgabe kann zentral definiert/geändert werden

Part II
Project Structure

22 Annotationen

22.1 @MappedSuperclass

- ist im Hybernate Framework eine Klasse durch die gemeinsame Felder definiert werden.
- definiert eine abstrakte Superklasse

@Produces – kommt während deployment, markiert Factory Method damit man nicht direkt auf die Klasse zugreifen muss @Typed – zeigt die Vererbung Wieso bei uns allein stehend? @Named – Zeigt bei Mehrdeutigkeit das richtige Objekt mit dem Namen @Resource – fast wie Dependency Injection @Stateless – speichert den Client Status nicht @Entity – Data Access Layer @Table – Tabellenname im SQL @Column – SQL-Spalten nullable=false @OneToMany Beziehung @JoinColums – welche Spalten zusammen gehören FK @OneToMany FK auf anderen Seite @ApplicationScoped – lebt die ganze Applikation lang, wird einmal gemacht. @PersistenceContext – persistance.xml auslesen für Treiber und andere JPA Geschichten + Data Source. Entity Manager Injection. @Id – das ist die id @GeneratedValue – Wert kommt aus der DB @Local – Klasse für lokale Aufrufe. @Remote – interprozessaufrufe. RMI @ApplicationException – Rollback wenn so eine Exception kommt, Nachricht zum Client.

23 Patterns in Practice

Data Access Layer Entity – Java Repräsentation vom DB Entity DAO damit man auf die Entities zugreifen kann. DB abstrahieren. Methoden mit denen man auf die DB zugreifen kann. DAOImpl – Implementierung DAOImpl – DAOException fehlt. Schlecht weil Input wird nicht kontrolliert. EntityManager in try catch, sonst kann es kleschen. Zusätzlich in DaoException wrappen. AbstractEntity – hier wird die id gemanaged Service Layer DTO – Aufrufgeschw. Verringern, nicht jedes Objekt einzeln aufrufen, sondern mit einmal alle notwendigen Objekte. Mapper – von DTO in Entity und Entity ins DTO.

FrontController web.xml ViewHelper *ServiceImpl

24 Konfigurationsdateien (pom.xml), (persistence.xml) und noch a bissl mehr Scheiß

Resource plugin – klar für Ressourcen Wildfly – server Primeafce = jsf Framework Jacoco = test Coverage Slf4j = logger Jaxb – xml Cdi = context dependancy injection

25 Reihenfolge - Wildfly - Abfolge - einzelne Schritte

Reihenfolge:

1. Compile
2. Surefire (unitTests)
3. Packaging - war file erstellen
4. Wildfly - fressen und deployen
5. Failsafe IT-test
6. MVN site
7. Gui test

26 Frageart Prüfung

Welche Fehler können bei Exception-Handling vorkommen in unserem Projekt?? – wie funktioniert es grundsätzlich in unserem Code

DocumentDAO – DocumentService – DocumentController – so sollte Exception-Handling implementiert warden

DAO wirft Exception – im ServiceLayer wird dies gefangen und der Stack-Trace wird im weggeloggt und eine benutzerfreundliche Fehlermeldung wird ausgegeben (Destructive Wrapping).

Alle Patterns, die vorkommen – praktische Beispiele aus dem Code

Was sind JavaBeans? Wie funktioniert das Konzept? Wie wird es genau implementiert? NamedBean, TypedBean etc.

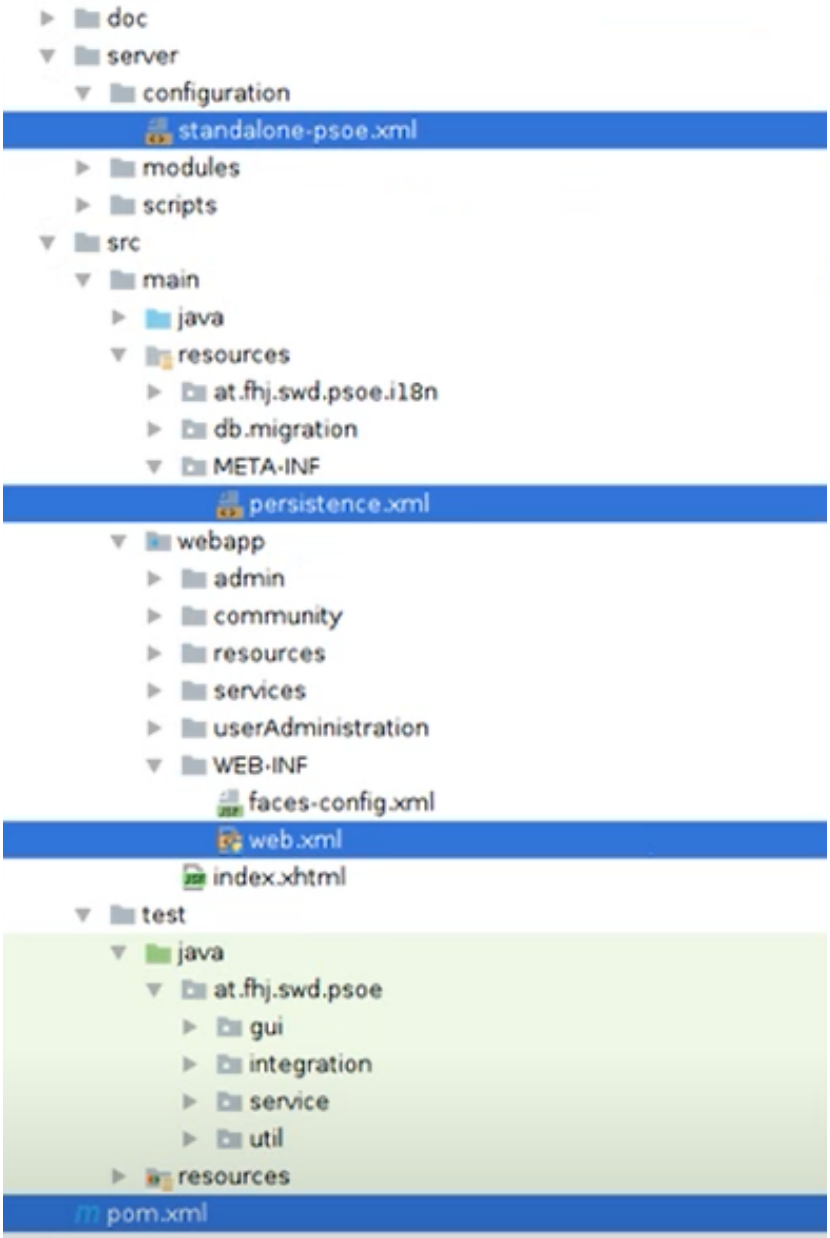
DTO

27 Die CONFIG-Files

27.1 web.xml

- konfiguriert den Java Webserver (Wildfly - JBOSS)
- befindet sich im Ordner **src/main/webapp/WEB-INF/web.xml**

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ↪ xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  ↪ xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  ↪ http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
3 ...
4 <servlet>
5 <servlet-name>Faces Servlet</servlet-name>
6 <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
```


doc
server
configuration
standalone-psoe.xml
modules
scripts
src
main
java
resources
at.fhj.swd.psoe.i18n
db.migration
META-INF
persistence.xml
webapp
admin
community
resources
services
userAdministration
WEB-INF
faces-config.xml
web.xml
index.xhtml
test
java
at.fhj.swd.psoe
gui
integration
service
util
resources
pom.xml

```
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>

<!-- Security roles -->
<security-role>
<description>administrators</description>
<role-name>ADMIN</role-name>
</security-role>
<security-role>
<description>portal administrators</description>
<role-name>PORTALADMIN</role-name>
</security-role>
<security-role>
<description>standard user</description>
<role-name>USER</role-name>
</security-role>

<!-- Security constraints -->
<security-constraint>
<web-resource-collection>
<web-resource-name>admin area</web-resource-name>
<url-pattern>/admin/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>community area</web-resource-name>
<url-pattern>/community/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>USER</role-name>
<role-name>PORTALADMIN</role-name>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>user administration area</web-resource-name>
<url-pattern>/userAdministration/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>USER</role-name>
<role-name>PORTALADMIN</role-name>
<role-name>ADMIN</role-name>
```

```
</auth-constraint>
</security-constraint>


<security-constraint>
<web-resource-collection>
<web-resource-name>user functionalities</web-resource-name>
<url-pattern>/user.xhtml</url-pattern>
<url-pattern>/userlist.xhtml</url-pattern>
<url-pattern>/notImplemented.xhtml</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>USER</role-name>
<role-name>PORTALADMIN</role-name>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>other functionalities</web-resource-name>
<url-pattern>/notImplemented.xhtml</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>USER</role-name>
<role-name>PORTALADMIN</role-name>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>

<login-config>
<auth-method>FORM</auth-method>
<realm-name>pse</realm-name>
<form-login-config>
<form-login-page>/login.xhtml</form-login-page>
<form-error-page>/login.xhtml</form-error-page>
<!--                    <form-error-page>/loginerror.xhtml</form-error-page> -->
</form-login-config>
</login-config>
</web-app>
```