Kapitel 1

Architektur

1.1 Vorgehensweise

Um die Architektur des Systems zu bestimmen, wird es in mehrere Teile (Subsysteme) zerlegt. Die Subsysteme bestehen aus einzelnen Modulen, die miteinander in Beziehung stehen können. Die Beziehungen zwischen den Subsystemen werden über Interfaces definiert.

Es gibt nun verschiedene Möglichkeiten, das System in Subsysteme zu zerlegen. Dazu bietet es sich an, von vordefinierten patterns (Mustern) auszugehen, die jeweils als Vorlagen dienen, das System zu definieren, zu unterteilen und zu organisieren.

Unsere Vorgehensweise besteht nun darin, zuerst die wichtigsten Use Cases zu gruppieren. Diese Gruppierungen liefern uns eine Partitionierung der Gesamtfunktionalität des Systems, sodass in der Entwicklungsphase einzelne Subsysteme möglichst unabhängig voneinander (Verbindungen zu anderen Subsystemen sind ja über die Interfaces wohldefiniert) in mehreren Iterationen entwickelt werden können.

Unsere Vorgehensweise bei der Zerteilung in Subsysteme und Module wird von folgenden Vereinbarungen bestimmt:

1. Verwendung des Object-Oriented-Pattern

2. Drei Schichten-Modell:

    - User-Interface
    - Business Layer
    - Database

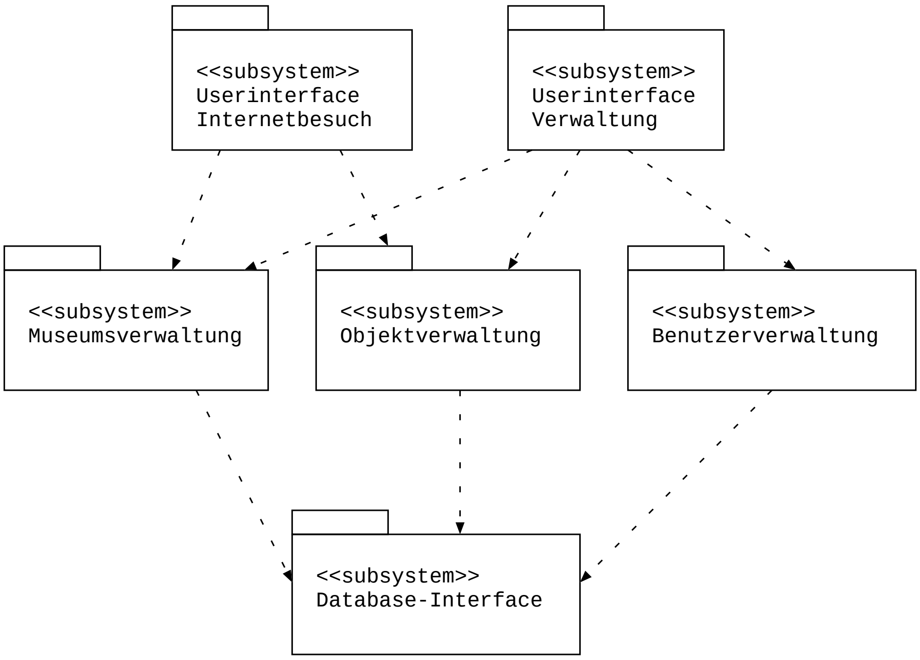
3. Verwendung von Java

4. XML wird zur Speicherung von Daten verwendet

1.2 Subsysteme

Ausgehend von den 3 Schichten User-Interface, Business-Layer und Database wird das System in Subsysteme aufgeteilt, die jeweils wieder aus Modulen bestehen.

Die Interfaces zwischen den Schichten werden weitgehend durch die Module (Klassen) des Business-Layer bestimmt, sodass der **Business-Layer eine zentrale Rolle in der Architektur** einnimmt.



1.2.1 Subsysteme des User-Interface

User-Interface Internetbesuch

User-Interface Verwaltung

Die Module der Subsysteme des User-Interface sind die Fenster, die im vorigen Kapitel dargestellt worden sind. Ihre Beziehungen untereinander sind bereits in den zugehörigen Diagrammen dargestellt worden.

Ihre Beziehungen zu den Subsystemen und Modulen des Business-Layers werden über Verknüpfungen der OK-, Cancel- etc. Buttons mit den Methoden der Module im Business-Layer hergestellt. Direkte Beziehungen zu den Modulen der Database sind nicht vorgesehen.

Implementiert werden die Subsysteme des User-Interface als **Java-Applets**.

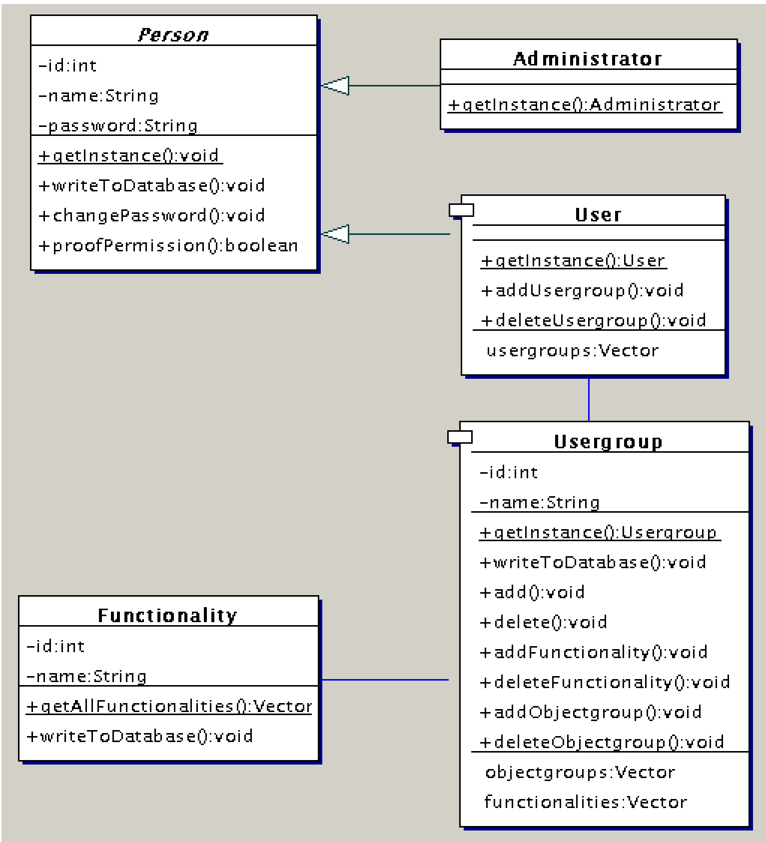
1.2.2 Subsysteme des Business-Layer

Die Module der Subsysteme, die zum Business Layer gehören, werden als Klassen spezifiziert. Die Interfaces der Klassen werden durch ihre Methoden, die Beziehungen untereinander durch Verweise auf andere Instanzen von den jeweiligen Klassen hergestellt.

Implementiert werden die Klassen des Business-Layers in **Java-Servlets**.

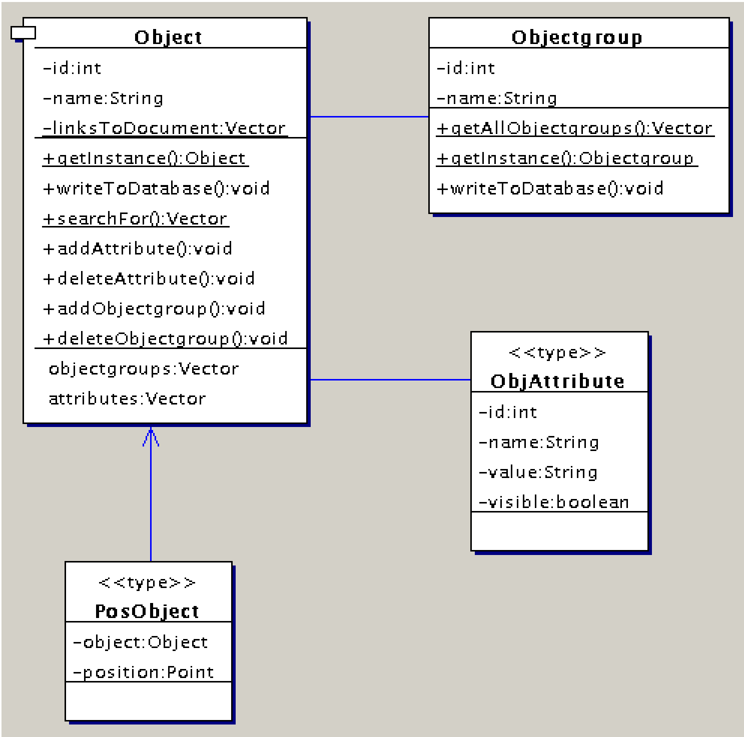
Die Kommunikation zu den User-Interfaces erfolgt über die definierten Methoden. Der Database-Schicht werden XML-Objekt geliefert, umgekehrt bezieht der Business-Layer seine Daten von der Datenbank in Form von XML-Objekten.

Benutzerverwaltung



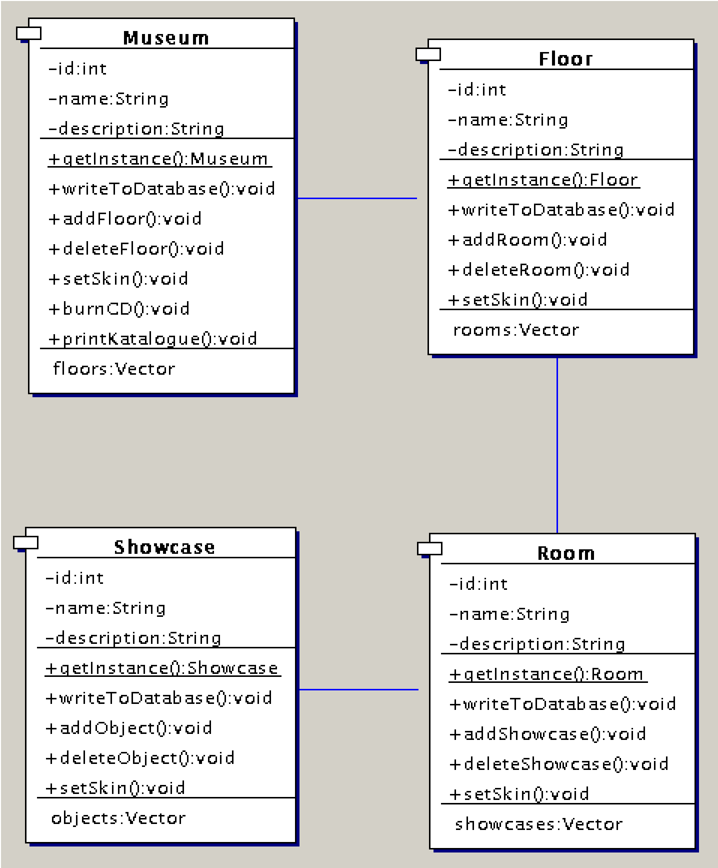
Verknüpfung zur Objektverwaltung: Usergroup-Objectgroup

Objektverwaltung



Verknüpfung zur Museumsverwaltung: PosObject-Showcase

Museumsverwaltung



1.2.3 Subsysteme der Database-Schicht

Das Database-Interface

Der Datenaustausch des Database-Interface mit den Subsystemen und Modulen des Business-Layers erfolgt mittels Austausch von XML-Objekten, die das Database-Interface in weiterer folge abspeichert.

1.3 Durchwanderung der Use Cases in der Architektur

1.3.1 BenutzergruppeErstellen

Um das Anwendungsfeld der einzelnen Operationen zu verdeutlichen wird teilweise ein Pseudocode verwendet.

1. *Verwaltungs UserInterface:* Auswahl: Benutzergruppe erstellen
2. *Verwaltungs UserInterface:* Eingabe eines Bezeichners für eine Benutzergruppe
3. *Usergroup.getInstance(<Bezeichner>)*
   Es wird Überprüft ob es schon eine Benuztergruppe mit dieser Bezeichnung existiert. Wenn nicht wird null zurückgegeben, ansonsten wird eine Referenz auf dieses Objekt übergeben.
4. if (usergr ist ungleich null)
   (a) *Verwaltungs UserInterface:* Es wird mitgeteilt, dass diese Gruppe schon existiert, aber wenn eine Kopie von dieser Usergroup erstellt werden will, soll ein neuer Bezeichner eingegeben werden.
   (b) *newUsergr = Usergroup.getInstance(<neuer Bezeichner>)*
   Es wird wird überprüft ob eine Usergroup mit diesem Bezeichner existiert.
   (c) if(newUsergr ist nicht null)
      i. *Verwaltungs UserInterface:* Erneute Eingabe des Bezeichners.
      ii. springe zu Punkt 4b
   (d) if(newUsergr ist null)
      i. *newUsergr = new Usergroup(<neuer Bezeichner>)*
      Erzeugen eines neuen Objekt von Usergroup
      ii. *funct = usergr.functionalities()*
      for(i=1 to funct.length())
         - *newUsergr.addFunctionality(funct[i])*

      Alle Funktionalitäten, welche in usergr enthalten sind werden in newUsergr eingetragen.

        iii. *ogroups = usergr.opjectgroups()*
        for(i=1 to ogroups.length()

            - *newUsergr.addObjectgroup(ogroups[i])*

        Alle Objektgruppe, welche in usergr enthalten sind, werden in newUsergr eingetragen.

        iv. usergr = newUsergr Ab jetzt wird mit der neu erzeugten und mit den Daten der alten Usergroup geladenen Usergroup weitergearbeitet.

        v. end if

    (e) if(usergr ist null)

        i. *usergr = new Usergroup(<Bezeichner>)*
        Ein neues Objekt von Usergroup wird erzeugt und unter usergr abgelegt

        ii. end if

    (f) end if

5. *Objectgroup.getAllObjectgroups()*
   Es werden alle allgemein vorhandenen Objektgruppen ermittelt.

6. *Objectgroup.getAllFunctionalities()*
   Es werden alle allegemein vorhandenen Funktionalitäten ermittelt.

7. *usergr.objectgroups()*
   Es werden alle beim Usergroup Objekt eigetragenen Objektgruppen ermittelt.

8. *usergr.functionalities()*
   Es werden alle beim Usergroup Objekt eingetragenen Funktionalitäten ermittelt.

9. *Verwaltungs UserInterface:* Alle allgemein vorhandenen Objektgruppen und Funktionalitäten werden angezeigt und jene welche schon beim Objekt eingetragen wurden als aktiviert angezeigt.

10. *Verwaltungs UserInterface:* Es können Objektgruppen und Funktionalitäten aktiviert und deaktiviert werden.

11. *Verwaltungs UserInterface:* Auswahl von Objektgruppe speichern.

12. if(mindesten 1 Funktionalität und mindestens 1 Objektgruppe sind aktiviert)

   (a) *usergr.save()* Die Daten des Usergroup Objekts werden in der Datenbank abgespeichert.

13. else

    (a) Gehe zurück zu Punkt 9.

14. end if

1.3.2 ObjekteSuchen

1. Internetuser startet Applet für ObjekteSuchen im Browser durch Mausklick auf den entsprechenden Button

2. Internetuser gibt in die Maske Suchbegriff(e) und erweiterte/einfache Suche ein.

3. Applet übergibt der statischen Suchfunktion searchFor() am Server die Parameter Suchbegriffe(e) und erweiterte/einfache Suche

4. Suchfunktion durchsucht über das Datenbankinterface die Datenbanken nach entsprechenden Objekten

5. Datenbankinterface gibt Objekte mit Verweis auf die jeweilige Vitrine zurück

6. Suchfunktion testet gefundene Objekte auf Mehrfachvorkommen und auf Sichtbarkeit

7. Suchfunktion gibt Liste mit gefundenen und geprüften Objekten mit Referenzen auf die Datenbank an das Applet retour

8. Applet stellt gefundene Objekte in Form einer Liste im Browser dar

9. Internetuser wählt Objekt aus Liste aus

10. Vitrine mit Objekt wird in Datenbank dargestellt

1.3.3 MuseumBesuchen

1. Museum auswählen

   (a) Der Internetuser hat die Möglichkeit ein Museum bzw. aus mehreren Museen eines auszuwählen.

2. Stockwerk auswählen

    (a) “Museum” hat eine Liste, in dem alle möglichen Stockwerke aufgelistet sind. Der Internetuser wählt ein Stockwerk aus und kommt so dorthin.

3. Raum auswählen

    (a) Hier wird wieder aus einer Liste aller verfügbaren Räume aus “Floor” der gewünschte ausgewählt.

4. Vitrine auswählen

    (a) Die Liste mit allen Vitrinen aus “Room” steht wieder zur Verfügung. Eine Vitrine wird ausgewählt.

5. Objekt auswählen

    (a) Aus der Liste der in einer Vitrine dargestellten Objekte aus “Showcase” wird das darzustellende Objekt ausgewählt.
    (b) Die Position und die Informationen zu einem Objekt aus “Object”werden geholt.

1.3.4 VitrineErstellen

Mit der Durchwanderung des Use Case VitrineErstellen im Klassendiagramm wird "uberpr"uft, ob die zur Verf"ugung gestellten Funktionalit"aten des Klassendiagramms ausreichend sind, damit das Use Case vollst"andig implementiert ist. Mit Hilfe des schon obengenannten Flow of Events f"ur den Use Case VitrineErstellen, wird der Use Case durchwandert und dabei wird betrachtet, ob jeder Punkt im Flow of Events implementiert ist und was die jeweiligen Klasssen und Methoden dazu beitragen.

1. Der Vorgang beginnt mit der Selektion „Vitrine erstellen“.

    Im Klassendiagramm:
    Mit der Selektion „Vitrine erstellen“ wird die Methode *getInstance():Showcase* in der Klasse *Showcase* aufgerufen. Mi dieser Methode(=Konstruktor) wird eine Instanz von Showcase (=Vitrine) gebildet und die Attribute *id* (=Prim"arschl"ussel der Vitrine), *name* (Titel der Vitrine) und description (=Beschreibung der Vitrine) initalisiert.

2. Festlegung wie die Vitrine dargestellt werden soll.

   Im Klassendiagramm:
   Mit der Methode *setSkin(): void* in der Klasse *Showcase* wird das Aussehen der Vitrine festgelegt.

3. Beschreibung der Vitrine

   Im Klassendiagramm:
   Wie schon oben erw"ahnt, wird mit der Methode *getInstance():Showcase* das Attribut description initalisiert.

4. if (Vitrine Raum zuweisen)

   (a) Vitrine Raum zu weisen.

   Im Klassendiagramm:
   Um einer Vitrine einen Raum zu zuweisen, wird die Mehtode *addShowcase():void* von der Klasse *Room* aufgerufen. Das L"oschen einer Zuweisung einer Vitrine zu einem Raum erfolgt mit *deleteShowcase():void* aus der Klasse *Room*. Im Vector *showcases* in der Klasse *Room* wird gespeichert, welche Vitrinen einem Raum zugewiesen sind.

5. if (Objekt in Vitrine positionieren)

   (a) while (weitere Objekte positionieren)

      i. Objektpositionierung

   (b) end

   Im Klassendiagramm:
   Mit den Methoden *addObjekt():void* bzw. *deleteObject():void* von der Klasse *Showcase* kann ein Objekt in einer Vitrine positioniert bzw. gel"oscht werden. Welche Objekte einer Vitrine zugeordnet sind, wird im Vector *objekts* gespeichert. Die Position des Objekts in der Vitrine wird im Typ *PosObjekt* festgehalten.

6. if (Speichern)

   (a) Der ganze Vorgang „Vitrine erstellen“ wird in der Datenbank abgespeichert.

   (b) Ende des Vorgangs „Vitrine erstellen“.

7. if (Abrechen)

8. Ende des Vorgangs „Vitrine erstellen“ ohne den Vorgang in der Datenbank zu speichern.

Im Klassendiagramm:
Immer wenn ein Vorgang abgeschlossen ist, wie z.B. Objekt positionieren, wird das Ergebnis in der Datenbank abgespeichert. D.h. *Speichern* bedeutet einfach die Sitzung ist abgeschlossen. Mit *Abbrechen* wird mittels einer Hilfsmethode in *Showcase* alles, was in dieser Sitzung „Vitrine erstellen“ in der Datenbank gespeichert wurde, wieder gel"oscht.

Kapitel 2

Projektplan für die Construction Phase

2.1 Kategorisierung der Use Cases

Wichtigkeit und Risiko werden jeweils in Stufen von 1-10 (10 = höchste Wichtigkeit bzw. größtes Risiko), Entwicklungszeit in geschätzten Mannwochen angegeben.

2.1.1 Objektverwaltung

2.1.1.1 Artefakte eingeben

- Wichtigkeit: 10
- Risiko: 10
- Entwicklungszeit: 15

2.1.1.2 Artefakte löschen

- Wichtigkeit: 10
- Risiko: 5
- Entwicklungszeit: 1

2.1.1.3 Objektgruppen erstellen

- Wichtigkeit: 10

- Risiko: 6
- Entwicklungszeit: 5

2.1.2 Benutzerverwaltung

2.1.2.1 Benutzergruppe erstellen

- Wichtigkeit: 10
- Risiko: 10
- Entwicklungszeit: 16

2.1.2.2 Rechte festlegen

- Wichtigkeit: 10
- Risiko: 7
- Entwicklungszeit: 5

2.1.2.3 Identifizierung eines neuen Mitarbeiters

- Wichtigkeit: 10
- Risiko: 6
- Entwicklungszeit: 1

2.1.3 Museumsverwaltung

2.1.3.1 Museum erstellen

- Wichtigkeit: 10
- Risiko: 6
- Entwicklungszeit: 5

2.1.3.2 Vitrine erstellen

- Wichtigkeit: 10
- Risiko: 8
- Entwicklungszeit: 12

2.1.4 Sonstige

2.1.4.1 Katalog erstellen

- Wichtigkeit: 5
- Risiko: 1
- Entwicklungszeit: 5

2.1.4.2 CD-ROM erzeugen

- Wichtigkeit: 5
- Risiko: 1
- Entwicklungszeit: 5

2.1.4.3 Suchen von Objekten

- Wichtigkeit: 8
- Risiko: 3
- Entwicklungszeit: 5

2.1.4.4 Datensicherung

- Wichtigkeit: 10
- Risiko: 1
- Entwicklungszeit: 2

2.2 Iterationen

Aufgrund der oben geschätzten Entwicklungszweiten, Wichtigkeiten und der Größe des Entwicklungsteams wird eine Iterationslänge von 4 Wochen festgelegt.

Insgesamt wird es 4 Iterationen geben.

Genannt sind jeweils die Use Cases und die zu schaffenden Voraussetzungen

Iteration 1

Artefakte eingeben und löschen, Objektgruppe erstellen:
Teil des User-Interface Verwaltung
Objektverwaltung
Datenbank-Interface

Iteration 2

Museum erstellen, Vitrine erstellen, Museumsbesuch:
Museumsverwaltung
Teil des User-Interface Verwaltung
Datenbank-Interface
User-Interface Internetbesuch

Iteration 3

Benutzergruppe erstellen, Identifizierung eines neuen Mitarbeiters, Rechte festlegen:
Benutzerverwaltung
Teil des User-Interface Verwaltung
Datenbank-Interface

Iteration 4

Suchen von Objekten, Datensicherung, CD-ROM erzeugen, Katalog erstellen:
Alle Subsysteme des Business-Layer
Alle User-Interfaces
Datenbank-Interface