Flugbuchungssystem

Aufgabe 1: Use Case Diagramm

Um die Anforderungen des fiktiven Kunden Airwings zu erfassen, sollen Use Case Diagramme verwendet werden. Im informellen Gespräch ergeben sich folgende groben Anforderungen an das System:

> Das System ist verantwortlich für das Erfassen der Buchungen. Die Buchungen sollen durch die Sachbearbeiter in den Reisebüros erfolgen. Desweiteren wird eine Software für die Platzverwaltung an Bord des Flugzeuges benötigt, mit der das Flugpersonal die bei der Buchung erfaßten Daten auswerten kann (z.B. die Wahl des Menüs). Auch das Gepäckleitystem soll modernisiert werden und von den bie der Buchung erfaßten Daten profitieren. Hierzu sollen die Gepäckstücke am Check-In einen Barcode erhalten, der durch Sensoren des Gepäckleitsystems abgefragt werden kann.

Um Einzelheiten zu erfragen sollen wir uns direkt an die Sachbearbeiter wenden. Hier erhalten wir folgende Information:

> Möchte ein Sachbearbeiter im Flugsystem eine Buchung erstellen, muß er beim Reisenden die Kundendaten ermitteln. Falls der Kunde neu ist, ist zusätzlich eine Kundenregistrierung notwendig.

Nach dem Gespräch können wir davon ausgehen, daß es im Verlauf des Requirement-Capture weitere Anwendungsfälle geben wird, die Kundendaten ermitteln müssen.

Aufgabe 2: Package Diagramme

Neben der in Aufgabe 1 näher betrachteten Buchung soll das zu entwickelnde System auch noch die Kundenverwaltung und das Gepäckleitsystem enthalten. Um das Projekt übersichtlich zu gestalten, soll je ein Paket für diese Teilaufgaben zur Modularisierung verwendet werden.

Gehen Sie dazu wie folgt vor:

1. Selektieren Sie im Model Pane des Explorers *default package*
2. Wählen Sie über die rechte Maustaste *open* oder *Open in New Tab* um ein neues Diagrammfenster zu öffnen.
3. Generieren Sie zwei Pakete im Diagrammfeld; bezeichnen Sie das erste mit `airwings`, das weite mit `buchung`.
4. Ziehen Sie das Paket `buchung` auf das Paket `airwings`.
5. Sie gelangen in das Paket `airwings`, indem Sie doppelt auf den Text in der Namenzeile klicken.
6. Erzeugen Sie innerhalb dieses Pakets zwei weitere Pakete `gepaeckleitsyste` und `kundenverwaltung`.
7. Verschieben Sie mit Hilfe des Projektexplorers das UseCase-Diagramm für die Flugbuchung durch Cut&Paste in das Paket `buchung`.

Aufgabe 3: Activity Diagramm

Um den genauen Ablauf bei einer Flugbuchung modellieren zu können, erfragen wir weitere Informationen beim Sachbearbeiter in der Buchungszentrale. Folgende Punkte sind zu beachten:

- Es kann zwischen verschiedenen Fluggesellschaften gewählt werden.
- Nach der Wahl einer Fluggesellschaft ermittelt der Sachbearbeiter die verfügbaren Flüge.
- Aus dem Angebot sucht der Kunde einen Flug aus mit dem er fliegen möchte.
- Entspricht kein Flug dem Kundenwunsch, so schlägt der Sachbearbeiter eine alternative Fluggesellschaft vor.
- Möchte der Kunde keine alternative Fluggesellschaft, so endet der Geschäftsprozess hier.
- Falls ein Flug ausgewählt wurde muss noch eine Überprüfung der verfügbaren Sitzplätze erfolgen.
- Der Flug kann nur gebucht werden, wenn genügend Sitzplätze vorhanden sind. Der Kunde erhält andernfalls die Möglichkeit einen anderen Flug zu wählen, sich in eine Warteliste eintragen zu lassen oder den Geschäftsfall zu beenden.
- Um die Buchung fertigzustellen benötigt der Sachbearbeiter die Personalien sowie die Optionen für den Flug (Essen, Fensterplatz u.ä.). Die Erfassung dieser Daten ist zeitlich unabhängig von der Erfassung der Flugdaten.

Aufgabe 4: Component Diagram

Im Activity Diagram können die Themenbereiche *Buchungsclient*, *Fluggesellschaft* und *Reisebüro* identifiziert werden. Dieses sind die Komponenten, deren Schnittstellen im folgenden definiert werden sollen. Im Activity Diagramm ist darüberhinaus die Notwendigkeit für Nachrichtenfluss zwischen den Komponenten ersichtlich:

- Der Buchungsclient erfragt Fluginformationen von der Datenbank der Fluggesellschaft.
- Die Datenbank des Reisebüros stellt Informationen bzgl der Verfügbarkeit von Sitzplätzen und die Wartelisten zur Verfügung.
- Der Buchungclient enthält Masken über die der Sachbearbeiter über den Status der Wartelisten informiert wird.

Erstellen Sie ein Komponentendiagramm, daß diese Sachverhalte darstellt. Dazu soll ein neues Diagramm des Typs Komponentendiagramm erstellt werden und dem Paket Buchung zugeordnet werden. Tragen Sie dann im Diagramm die Komponenten, ihre Interfaces (Verbindung mit *supports*) und die Abhängigkeiten der Komponenten von den Interfaces ein (Verbindung mit *dependency*).

Aufgabe 5: Deployment Diagram

Ein Deployment Diagram gibt die Zuordnung von Komponenten zu physikalischen Objekten an. In unserem Beispiel gibt es vier solche Laufzeitobjekte, nämlich die Rechner in den Reisebüros, den Rechner am Check-In, die Boardrechner (die die Verteilung der Menüs und andere flugspezifische Information verwaltet) und das Rechenzentrum, über das die gesamte Information ausgetauscht wird und wo die zentrale Datenbank installiert ist. Diese Rechner sind über verschiedene Medien verbunden:

- Die Reisebüros sind mit dem Rechenzentrum und untereinander per ISDN verbunden.
- Der Check-In ist mit dem Rechenzentrum über TCP/IP verbunden.
- Das Boardsystem ist mit dem Rechenzentrum über Funk angebunden

Erzeugen Sie ein Deployment Diagram das den obigen Sachverhalt widerspiegelt mit dem Namen *deployment-gesamt* im Paket *airwings*.

Aufgabe 6: State Diagram

Für den Bereich Buchung gibt es Regeln, die das Fortschreiten des Buchungsvorgangs beschreiben. Diese sollen in einem Zustandsdiagramm festgehalten werden; dabei gehen wir davon aus, daß hier der Zustand eines eindeutig zugeordneten Objekts (das im folgenden auch Buchung genannt wird) beschrieben wird. Das Objekt und damit das Zustandsdiagramm ist dem Paket Buchung zuzuornden. Die Regeln sind die folgenden

- Zuerst muß der Buchung ein Kunde zugeordnet werden.
- Wird ein passender Flug gefunden, so erhält die Buchung den Status *Flugnummer zugeordnet*.
- Ist kein passender Flug verfügbar, so erfolgt die Beendigung des Geschäftsfalls.
- Beim Betreten des Zustands emphFlugnummer zugeordnet wird gepüft ob ein Sitzplatz verfügbar ist. Bei der folgenden Sitzplatzreservierung wird nun entweder ein Sitzplatz zugeteilt (dh reserviert) oder der Kunde wird auf eine Warteliste gesetzt oder ein neuer Flug gewählt.

  Anmerkung: Identifizieren Sie hier die Übergangsbedingungen!
- Wird ein Eintrag auf die Warteliste vorgenommen, so soll automatisch regelmäßig geprüft wrden ob ein Sitzplatz storniert wurde und damit für unseren Kunden reserviert werden kann.

Identifizieren Sie geeignete Zustände für das Objekt Buchung und definieren sie Transitionen zwischen den Zuständen unter Verwendung der Events und Guards (in den Properties der Zustände editieren). Für interne Transitionen (entry, exit, do) kann die Properties-Karte über das Kontextmenü (rechte Maustaste auf dem Zustand) ausgewählt werden. Bei internen Transitionen sind die Eventnamen (dh entry, exit, do) ausschließlich im Diagramm editierbar.

Anmerkung: Das Zustandsdiagramm kann einer Klasse direkt zugeordnet werden. Dies kann durch Aktivierung des Kontextmenüs in einem freien Bereich des Diagramms geschehen; Auswahl von *target class*. Danach kann einer *action expression* direkt eine Methode der Klasse zugeordnet werden und von diesem Punkt an werden Action und Methode synchrongehalten (zB. bei Namensänderungen).

Aufgabe 7: Sequence Diagram 1

Der Vorgang der Buchung wird im Folgenden detaillierter beschrieben:

- Zunächst ruft der Sachbearbeiter die Funktion `bucheReise():void` auf, die vom Objekt `buchungsdialog` ausgeführt wird; danach ruft er die Funktion `buche():void` auf, die im selben Objekt verfügbar ist.
- Die Funktion `bucheReise` läuft wie folgt ab:
  - Sie erzeugt als erstes eine Instanz der Klasse `Buchung` namens `eine Buchung`
  - danach ruft sie ein Funktion des eigenen Objektes namens `erfasseKundendaten():void` auf
  - Anschliessend werden diese Daten über die Funktion `setKunde(Kunde):void` dem Objekt `eineBuchung` mitgeteilt.
  - Erst nach Beendigung dieser Funktion können die verfügbaren Flüge bei dem Objekt `einFlugKatalog` der Klasse `FlugDB` erfragt werden; hierzu wird die Funktion `getFluege():void` verwendet.
  - Die erhaltenen Daten werden dann mit der internen Funktion `zeigeFluege():void` im Objekt `Dialog` angezeigt werden.
- Die Funktion `buche` des Objekts `buchungsDialog` erledigt die weiteren Aufgaben der Buchung nachdem der Flug feststeht:
  - zunächst werden mit der Funktion `getFreiePlaetze():int` von einem weiteren Objekt namens `reisebuerokontingent` der Klasse `ReisebueroDB` die Anzahl der freien Plätze ermittelt und in die Variable `freiePlaetze` gespeichert
  - Ist diese Anzahl > 0, so wird erst die Funktion `setFlugNummer(int):void` des Objekts `eineBuchung` aufgerufen,
    dann die Reservierung mit `setReservierung(Buchung):boolean` bei `reisebuerokontingent` durchgeführt; der Rückgabewert enthält Information über den Erfolg der Reservierung.
  - Falls jedoch keine freien Plätze mehr verfügbar sind (d.h. wenn `freiePlaetze` $= 0$), dann wird zunächst von `buchungsDialog` die eigenen Funktion `getAlternative():void aufgerufen und dabei die Variable alternative gesetzt`. Ist `alternative` == `wait`, so wird ebenfalls die Flugnummer in der Buchung gesetzt und im Anschluss die Warteliste abgefragt mit der Funktion `getWaiting(Buchung):void` des Objekts `reisebuerokontingent`. Ist die Alternative `cancel`, so wird das Objekt `eineBuchung` gelöscht und anschließend der Dialog mit der eigenen Funktion `schliesseDialog():void` geschlossen.

Setzen Sie diese informelle Beschreibung in ein Sequenzdiagramm um. Erzeugen Sie dazu ein Sequenzdiagramm im Paket Buchung. Verwenden Sie für das Diagramm den

Akteur `SB` für den Sachbearbeitern und legen Sie zu Beginn 4 Objekte über die Option **New Class** des Kontextmenüs an (damit sind die Klassen auch angelegt unter dem Paket Buchung):

- `buchungsDialog` von der Klasse `Dialog`
- `eineBuchung` von der Klasse `Buchung`
- `reiseBueroKontingent` der Klasse `ReisebueroDB`
- `einFlugKatalog` der Klasse `FlugDB`

Erzeugen Sie die Kommunikationen als Messages mit Hilfe der Schablonen und der Explorer-Panel. Um die Message-Namen zu setzen empfiehlt es sich über das Kontextmenü die Option **New Operation** zu verwenden, die auch gleich die Methoden in den jeweiligen Klassen anlegt. Achten Sie dabei darauf, wie die einzelnen Aktionen nummeriert werden.

Nachdem das Sequenzdiagramm erstellt ist, können Sie den Code der Funktionen erzeugen, indem Sie die Maus über der Methode positionieren (für unser Beispiel sind hier `bucheReise` und `buche` sinnvoll) und aus dem Kontextmenü die Option **generate Implementation** aufrufen.

Beachten Sie die Anteile des Codes, der farbig markiert ist; hier fehlen noch Definitionen! Ergänzen Sie diese soweit möglich und achten Sie dabei auf die Hilfen von Together.

Aufgabe 8: Sequence Diagram 2

Aufgabe 7 demonstriert die Möglichkeit mit Together aus einem Sequenzdiagramm Code zu erzeugen. Der umgekehrte Weg ist auch möglich und soll im Folgenden geübt werden.

Erzeugen Sie zunächst im Paket `gepaeckleitsystem` ein Klassendiagramm und fügen Sie die folgenden 3 Klassen ein:

- Klasse `Weiche` mit privatem Attribut `stellwert: int` (private Attribute werden mit einem `-` gekennzeichnet), und zwei public Methoden (diese werden mit `+` markiert) `setZiel(gepaeck:Gepaeck):void` und `setStellwert (stellwert:int):void` .
- Klasse `Gepaeck` mit der public Methode `getAdresse():Adresse`
- Klasse `Adresse`, die mit `Gepaeck` verknüpft ist und das private Attribut `verladestelle: int`, sowie die public Methoden `getVerladestelle():int` und `setVerladestelle (verladestelle:int):void` besitzt.

Implementieren Sie in den jeweiligen Klassen die Methoden (zB. im Code-Fenster):

```
public void setZiel (Gepaeck gepaeck) {
  Adresse adr = gepaeck.getAdresse();
  setStellwert (adr.getVerladestelle());
}

public void setStellwert (int stellwert) {
  this.stellwert = stellwert;
}

public Adresse getAdresse() {
  return lnkAdresse;
}
```

Öffnen Sie im Diagrammfeld das Kontextmenü über der Methode `setZiel()` und wählen Sie die Option **generate Sequence Diagramm**. Vergleichen Sie das Ergebnis mit dem erwarteten Output.