Einarbeitung Nr. 3
UML

Im Praktikum wird die UML (Unified Modeling Language) mit der Vorgehensweise von OMT (Object Modeling Technique) verwendet. Wie dies geschieht, wird im folgenden kurz zusammengefaßt. Die Notation und die Vorgehensweise sind ausführlich in folgenden Büchern beschrieben:

- Bernd Oesterreich: Objektorientierte Softwareentwicklung mit der Unified Modeling Language. Oldenbourg-Verlag, 4., aktualisierte Auflage 1998
- James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen: Object-Oriented Modelling and Design. Prentice Hall, 1991

Diese Bücher sind in der Lehrbuchsammlung verfügbar. Aufgaben zum Erlernen (zur praktischen Verwendung) der Notation und Vorgehensweise können diesen entnommen werden. Ein Durcharbeiten der Bücher wird empfohlen. Mit den Einarbeitungsunterlagen erhaltet Ihr Kopien, die einen Überblick über die wichtigsten Beschreibungsmittel der UML geben. Darüber hinaus ist der Notation-Guide zu UML (zu Verwenden als Nachschlagewerk) von Rational in den Praktikumsräumen verfügbar.

Vorgehensweise im Praktikum

Die vorgestellte Vorgehensweise lehnt sich an die Vorgehensweisen in den oben beschriebenen Büchern an. Da der Schwerpunkt der Bücher auf der Entwicklung von Informationssystemen liegt, wurden außerdem Erfahrungen aus durchgeführten Praktika zur Definition des Vorgehens verwendet. Die Beschreibung setzt voraus, daß die UML-Notation bekannt ist.

Es werden im weiteren fünf Phasen der Software-Entwicklung unterschieden:

- Grobanalyse des Verhaltens des Systems
- Feinanalyse der statischen, dynamischen und funktionalen Aspekte des Systems
- Grobentwurf des Systems
- Feinentwurf und Komponenten-Anforderungen
- Implementierung

In den einzelnen Phasen werden verschiedene Aktivitäten ausgeführt.

1. Grobanalyse des Verhaltens des Systems

Während der Grobanalyse wird ein Anwendungsfalldiagramm (Use Case Diagram) erstellt. Dieses stellt typische Verwendungen des zu entwickelnden Systems dar. Hierfür müssen Akteure (engl. actors) lokalisiert werden, die bestimmte Aktionen des Systems auslösen. Ein Akteur kann der Benutzer sein, der z.B. Werte im System setzt. Akteure können aber auch Sensoren sein, die dazu führen, daß das System reagiert. Ein Beispiel ist der Regensensor, der Regen meldet und dadurch veranlaßt, daß das System die vollständig geöffneten Fenster kippt. Anwendungsfälle (Use Cases) werden durch eine Reihe von informellen Szenarien konkretisiert.

Die Szenarien müssen die informellen B-Anforderungen (funktionale + inverse) abdecken.

2. Feinanalyse der statischen, dynamischen und funktionalen Aspekte

Während der Feinanalyse werden eine Reihe von Diagrammen entwickelt: das Klassendiagramm mit erläuterndem Data-Dictionary, Sequenzdiagramme, Kollaborationsdiagramme und Zustandsdiagramme. Die Erstellung der Diagramme erfolgt iterativ, d.h. z.B. daß das Klassendiagramm um Informationen aus dem dynamischen Modell ergänzt wird. Außerdem muß die Konsistenz zwischen den Diagrammen gewährleistet werden.

- **Erstellung eines Klassendiagramms:** Mit dem Klassendiagramm wird die statische Struktur eines Systems beschrieben. Neben der Beschreibung der Identität (Attribute und Methoden) der Klassen werden die Beziehungen zwischen den Klassen im System festgehalten. Zur Unterstützung des Prozesses wird ein Data-Dictionary angelegt, in welchem die Klassen mit ihren Attributen und Methoden beschrieben werden. Für die Erstellung des Klassendiagramms bietet sich folgendes Vorgehen an:
  - *Identifiziere Objekte und Klassen:* Verwende hierfür die Gebäudearchitektur und bilde für jede Größe eine Klasse (z.B. Klasse Raum, Fenster oder Haus). Bilde anschließend die Struktur des Gebäudes durch Assoziationen und Aggregationen nach (ein Raum hat ein Fenster, ein Haus hat mehrere Räume). Bilde weiterhin für jeden existierenden Sensor und Aktuator eine Klasse (z.B. Temperatursensor oder Kippmotor). Integriere die Klassen für Sensoren/Aktuatoren über Assoziationen und Aggregationen mit denen der Gebäudearchitektur (ein Tempertursensor existiert in jedem Raum, ein Regensensor hängt am Haus). Bilde anschließend Funktionsklassen (Controllerklassen), die verschiedene Sensorwerte miteinander verbinden (Anwesenheitscontroller, Temperaturregler). Werte, die der Benutzer eingibt (z.B. Zeitspannen für Personenanwesenheit) können ebenfalls als Sensorwerte verstanden werden. Integriere die Controllerklassen in die existierende Struktur über Assoziationen und Aggregationen.
  - *Identifiziere erste Attribute und Methoden* der Klassen. Überlege Dir hierfür, welche Aufgaben die einzelnen Klassen haben sollen und welche Werte in ihnen gespeichert sind.
  - *Lege ein Data-Dictionary an*, in welchem jede Klasse vorläufig im Gesamtkontext mit ihren Attributen und Methoden beschrieben wird.
  - *Strukturiere und vereinfache* die Klassen durch die Verwendung von Vererbungsbeziehungen. Insbesondere bei den Sensoren und Aktuatoren wird sich eine Vererbungsstruktur anbieten.
  - *Gruppiere Klassen zu Teilsystemen*, die eng gekoppelt sind und Funktionen realisieren, die zueinander in Beziehung stehen. Im zu entwickelnden System wurden bereits vier solcher Teilsysteme lokalisiert. Dafür wird in UML das Beschreibungsmittel “Packages” verwendet.
  - *Iteriere das Vorgehen* und erweitere das bestehende Modell.
- **Entwicklung eines dynamisches Modells:** Mit diesem Modell werden die Kontrollaspekte eines Systems beschrieben. Das dynamische Modell enthält die Sequenzdiagramme, die Kollaborationsdiagramme und die Zustandsdiagramme.
  - *Sequenzdiagramme:* Die Sequenzdiagramme formalisieren die informellen Szenarien, d.h. für jedes informelle Szenario muß ein Sequenzdiagramm entwickelt werden. Dafür werden die informellen Szenarien abgebildet auf die Interaktion zwischen Instanzen von Klassen. Nachrichten, die eine Instanz einer Klasse im Sequenzdiagramm geschickt bekommt, müssen als Methoden der Klasse definiert und ins Klassendiagramm aufgenommen werden.
  - *Kollaborationsdiagramme:* Die Kollaborationsdiagramme enthalten die gleiche Information, wie die Sequenzdiagramme. Sie stellen die Information nur etwas anders dar. Die Kollaborationsdiagramme können aus den Sequenzdiagrammen automatisch generiert werden.

- *Zustandsdiagramme:* Für jede Klasse mit dynamischen Verhalten wird ein Zustandsdiagramm erstellt. Eine Klasse hat ein dynamisches Verhalten, wenn sie sich durch verschiedene Zustände auszeichnet und das Verhalten der Klasse abhängig von ihren verschiedenen Zuständen ist. Ein Beispiel ist die Klasse Fenster, die die Zustände: geschlossen, geöffnet oder gekippt hat. Im Zustandsdiagramm werden alle möglichen Zustände und Zustandsübergänge beschrieben. Nachrichten, die eine Instanz einer Klasse im Sequenzdiagramm geschickt bekommt, müssen im Zustandsdiagramm als Nachrichten auftauchen, auf die die Klasse wartet (z.B. als Bedingung am Zustandsübergang). Nachrichten, die eine Instanz einer Klasse verschickt, müssen im Zustandsdiagramm ebenfalls verschickt werden. Dies gilt auch umgekehrt, d.h. alle Nachrichten, die im Zustandsdiagramm empfangen oder verschickt werden, müssen in mindestens einem Sequenzdiagramm auftauchen.
  Eine Ausnahme dieser Regel sind Durchreichmethoden (z.B. Haus reicht die Regenmeldung an die einzelnen Räume weiter) und Methoden zum Setzen und Lesen von Attributen. Diese müssen im Zustandsdiagramm nicht auftauchen, allerdings im Sequenzdiagramm.
  Alle für die Klasse globalen Variablen, die in einem Zustandsdiagramm verwendet werden, müssen als Attribute der Klasse im Klassendiagramm definiert werden.

1. **Konstruktion eines funktionalen Modells**: Mit diesem Modell werden die Aspekte des Systems beschrieben, in denen Werte durch Funktionen, Abbildungen oder funktionale Abhängigkeiten verändert werden. Die Konstruktion eines funktionalen Modells ist immer dann wichtig, wenn das System umfangreiche Berechnungen durchführen muß, z.B. im Falle der Berechnung der Haltetemperatur. Die umfangreiche Berechnung taucht im Zustandsdiagramm meist als Do-Aktivität auf. Zur Verfeinerung wird ein funktionales Modell erstellt. Dafür wird ein Activity-Diagramm verwendet.

2. **Überarbeitung der drei Modelle:** Verifiziere, iteriere und überarbeite die drei Modelle. Das fertige Analysedokument besteht aus einem konsistenten statischen, dynamischen und funktionalen Modell.

1. Grobentwurf des Systems

Während des Grobentwurfs werden übergeordnete Entwurfsentscheidungen getroffen, die Auswirkungen auf die gesamte Struktur des Systems haben (z.B. Integration einer Portabilitätsschicht). Das Systementwurfsdokument beschreibt die grundlegende Systemarchitektur, sowie grundlegende Strategieentscheidungen. Raumbaugh et al. schlagen vor, zur Erfüllung dieser Phase im einzelnen wie folgt vorzugehen:

- Ordne zusammengehörige Aufgaben und bilde Teilsysteme.
- Identifiziere inhärente Parallelität in der Problemstellung.
- Ordne Teilsysteme zu Prozessoren zu.
- Wähle die Basisstrategie für die Implementierung des Speicherns von Daten.
- Wähle einen Ansatz zur Implementierung der inhärenten Parallelität der Zustandsdiagramme. Im Praktikum geschieht dies über einen Scheduler-Mechanismus.
- Betrachte Grenzbedingungen.
- Etabliere „trade-off“ Prioritäten.

Die Entwurfsentscheidungen sind im Praktikum überwiegend vorgegeben.

2. Feinentwurf und Komponenten-Anforderungen

Im Feinentwurf wird das Analysemodell entsprechend konkreter Implementierungsdetails (der Entwurfsentscheidungen) überarbeitet. Das Feinentwurfsdokument besteht aus dem detaillierten Klassendiagramm, dem detaillierten dynamischen Modell und dem detaillierten funktionalen Modell. Im Praktikum werden das Systementwurfsdokument und das Feinentwurfsdokument zu einem Entwurfsdokument zusammengefaßt. Um diese Phase zu erfüllen, sollten folgende Schritte durchgeführt werden:

- Überlege, wie Klassen angelegt und gelöscht werden. Definiere Konstruktoren und Destruktoren.
- Entwerfe Algorithmen, um Methoden zu implementieren:
  - Wähle Algorithmen, die die Implementierungskosten von Methoden minimieren.
  - Wähle zu diesen Algorithmen angemessene Datenstrukturen aus.
  - Definiere, wenn notwendig, neue Klassen und Methoden.
- Optimiere die Zugangspfade zu den Daten:
  - Füge redundante Assoziationen hinzu, um die Zugangskosten zu minimieren und den Komfort zu erhöhen.
  - Organisiere die Berechnungen hinsichtlich größerer Effizienz um.
  - Speichere abgeleitete Werte, um eine Neuberechnung komplizierter Ausdrücke zu vermeiden.
- Implementiere die Kontrollstrukturen unter Verwendung des Ansatzes, der während des Grobentwurfs ausgewählt wurde.
- Überarbeite die Klassenstruktur, um das Konzept der Vererbung umfassender auszunutzen:
  - Ordne die Klassen und die Methoden neu an, so daß das Konzept der Vererbung umfassender ausgenutzt wird.
  - Abstrahiere gemeinsames Verhalten aus einer Gruppe von Klassen heraus.
  - Verwende Delegation, um an den Stellen Verhalten aufzuteilen, an denen Vererbung semantisch unzulässig ist.
- Entwerfe Implementierungen für Assoziationen:
  - Analysiere die Gegenseitigkeit von Assoziationsbeziehungen.
  - Implementiere jede Assoziation als ein spezifisches Objekt oder durch das Hinzufügen von Objektwert-Attributen zu einer oder beiden Klassen der Assoziation.
  - Lege die exakte Repräsentation der Objektattribute fest.
  - Fasse Klassen und Assoziationen in Module zusammen.

3. Implementierung

In diesem Schritt wird der Entwurf in eine ausführbare Form überführt. Im Praktikum existieren Richtlinien für die Erstellung von Kode aus dem Feinentwurf.