Dozent: G.Döben-Henisch	PPmP
	VL4

VL4: Softwareprojekt - Modellierung/Design Teil 2

(Wegen Klausur verkürzte Vorlesung)

Inhalt

1. Einleitung

In der vorausgehenden Vorlesung wurde ein allgemeines Schema für Softwareprojekte eingeführt, wie es im Rahmen der *Model Driven Architecture (MDA)* der *OMG* angenommen wird. An einem einfachen Beispiel wurde erklärt, was es bedeuten kann, ein konkretes Anwendungsbeispiel als ein *computerunabhängiges ppmp2ps-Modell (ppmp2ps-CIM)* in Form eines *ppmp2ps-Anwendungsfalldiagramms (ppmp2ps-Use Case)* zu konstruieren. Dann wurde erklärt, wie man auf der Basis eines Anwendungsfalls ein *ppmp2ps-plattformunabhängiges Modell (ppmp2ps-PIM)* konstruieren und dieses dann in ein *ppmp2ps-plattformspezidisches Modell (ppmp2ps-PSM)* übersetzen kann. Dabei zeigte sich, dass im ersten Anlauf zwar die *statischen Eigenschaften* im Modell ppmp2ps-PIM erfasst und diese auch automatisch mit umbrello nach ppmp2ps-PSM übersetzt worden sind, es fehlten jedoch noch alle Angaben darüber, wie sich dieses Modell *dynamisch* verhält. Dies zeigte sich dann u.a. darin, dass im ppmp2ps-PSM alle Funktionen nur als Funktionsrümpfe, d.h. nur mit ihren *Signaturen* vorkommen. In dieser Vorlesung soll nun erklärt werden, wie man die dynamische Struktur des *computerunabhängiges ppmp2ps-Modells (ppmp2ps-CIM)* in das ppmp2ps-PIM Modell als ppmp2ps-PIM2-Modell integrieren kann.

2. Modellierung dynamischer Eigenschaften: ppmp2ps-PIM2

Eine *dynamische* Analyse setzt eine Zeitachse voraus, auf der Objekte existieren und zu bestimmten Zeitpunkten mit anderen Objekten *interagieren*. Diese Interaktionen werden mittels *Austausch von Nachrichten* realisiert oder durch *Aufrufen von öffentlichen Methoden*. Im Bild 1 ist ein einfaches Sequenzdiagramm dargestellt. Es gibt zwei Objekte auf ihren Zeitachsen: den normalen *Benutzer* und das Systemelement *ppmp2ps.* Von dem Systemelement ppmp2ps wissen wir, dass es zwei öffentlich zugängliche Methoden besitzt: *fileselect* und *convert_ascii_ps*. Dies bedeutet, der Benutzer kann mit dem Systemelement ppmp2ps dadurch in Interaktion treten, dass der Benutzer eine dieser öffentlich zugänglichen Methoden aufruft. Im Bild ist der Fall gezeigt, dass der Benutzer zuerst die Methode *fileselect(filename)* mit dem Namen einer ASCII-Datei aufruft, die geöffnet werden soll, und danach die Methode *convert_ascii_ps()* ohne Argument. Mit der

Methode *convert_ascii_ps()* wird das System aufgefordert, den zuvor ausgewählten ASCII-Text in einen Postscript Text zu übersetzen. Auf diese Weise kann man dem Diagramm entnehmen, welches Objekt zu welchem Zeitpunkt welche Aktion ausführt.

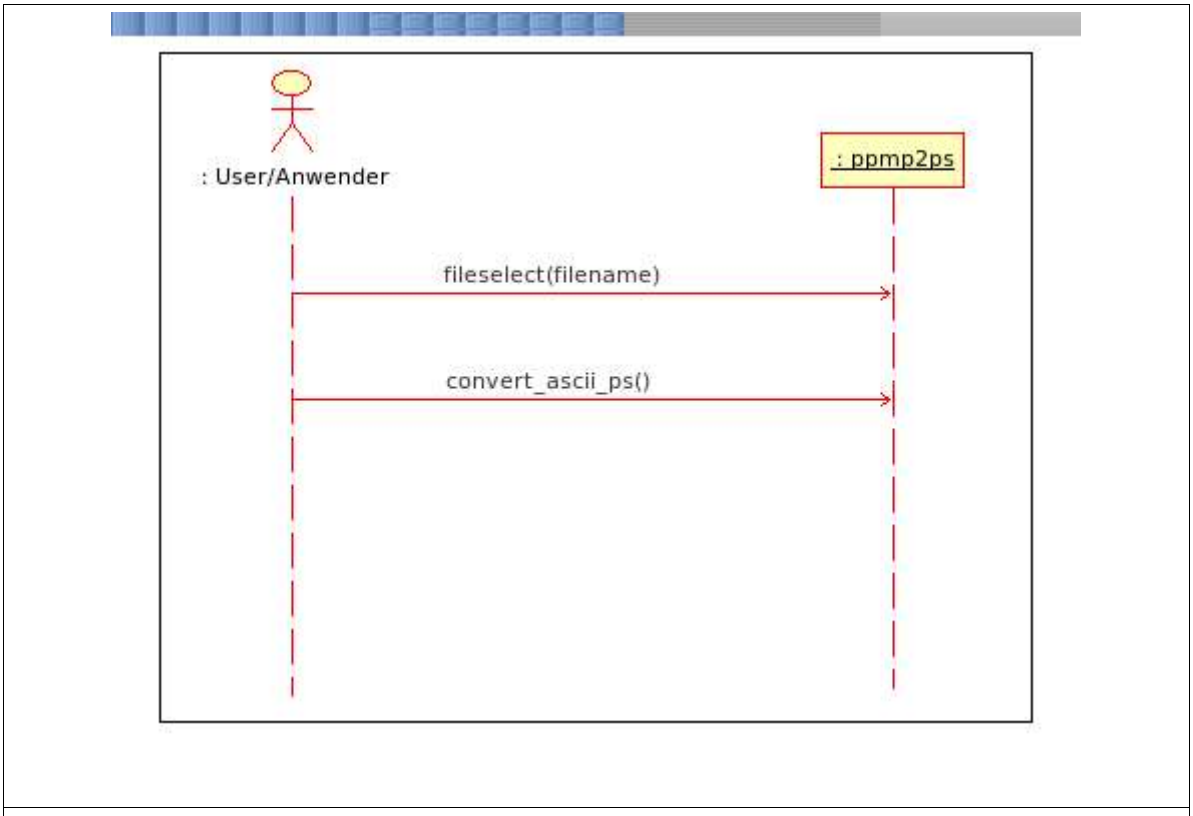

Bild 1: ppmp2ps-Sequenzdiagramm

Mit dieser zusätzlichen Analyse wird der zu analysierende Fall etwas "klarer". Es fragt sich allerdings, ob man noch weitere Analysen vornehmen kann?

Nimmt man das ppmp2ps-PIM-Modell so, wie es jetzt ist, sagt es nichts darüber aus, *wie* die Methoden *fileselect()* bzw. *convert_ascii_ps()* im einzelnen arbeiten, d.h. die *Aktivitäten*, die sich "hinter" den Methodennamen "verbergen", "liegen noch im Dunklen". Eine Möglichkeit, diese jetzt noch "unsichtbaren" Aktivitäten zu analysieren, wäre z.B. ein *Aktivitätsdiagramm (früher: Flussdiagramm)*.

Im Bild 2 wird eine Möglichkeit gezeigt, wie man die Aktivität "hinter" dem Methodennamen *fileselect(filename)* analysieren könnte.

Man beginnt mit einem Startpunkt und dem Namen der Methode, die man analysieren möchte. Dann gibt man *Teilaktivitäten* (in UML als Rechtecke mit gerundeten Ecken) innerhalb der Methode an, mittels deren man die Gesamtaktivität weiter zu analysieren trachtet. Sofern eine Aktivität mit einem möglichen *Fehler* verbunden sein kann, muss man diese möglichen *Fehler abfangen.* Dies kann man durch *Verzweigungen* erreichen (in UML als auf der Ecke stehende Quadrate realisiert). Alle Aktivitäten enden dann in

Endpunkten (Kreise mit einem Punkt).

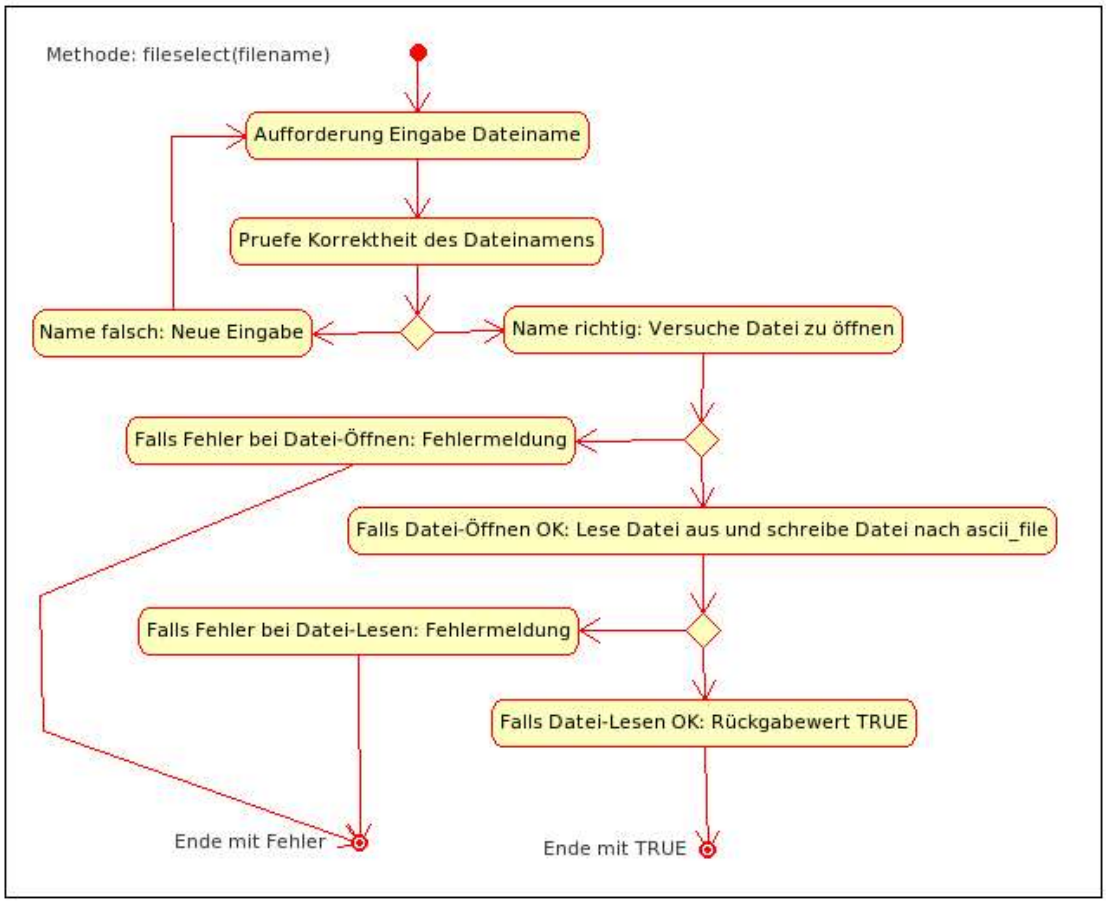

Bild 2: ppmp2ps-Aktivitätsdiagramm für die Methode fileselect(filename)

Alternativ zu diesem Vorgehen könnte man sich auch vorstellen, eine *Klasse fileselect* zu generieren in der alle die genannten Teilaktivitäten Methoden dieser Klasse wären. In diesem Fall könnte man die Abfolge der Methoden wieder durch ein Sequenzdiagramm darstellen. Die Entscheidung, ob man etwas nur als Methode sieht, die mit einem Aktivitätsdiagramm analysiert wird, oder ob man aus der Methode wieder eine Klasse macht, deren Methoden die Teilaktivitäten der vorherigen Methode sind, ist eine Ermessensfrage. Wir werden im weiteren Verlauf diesen Fall wenigstens einmal durchspielen.

In Bild 3 eine mögliche Analyse der Aktivitäten in Verbindung mit der Methode *convert_ascii_ps()*. Speziell im Fall dieser Methode wird es sich nahelegen, diese Methode eventuell als Klasse weiter zu analysieren.

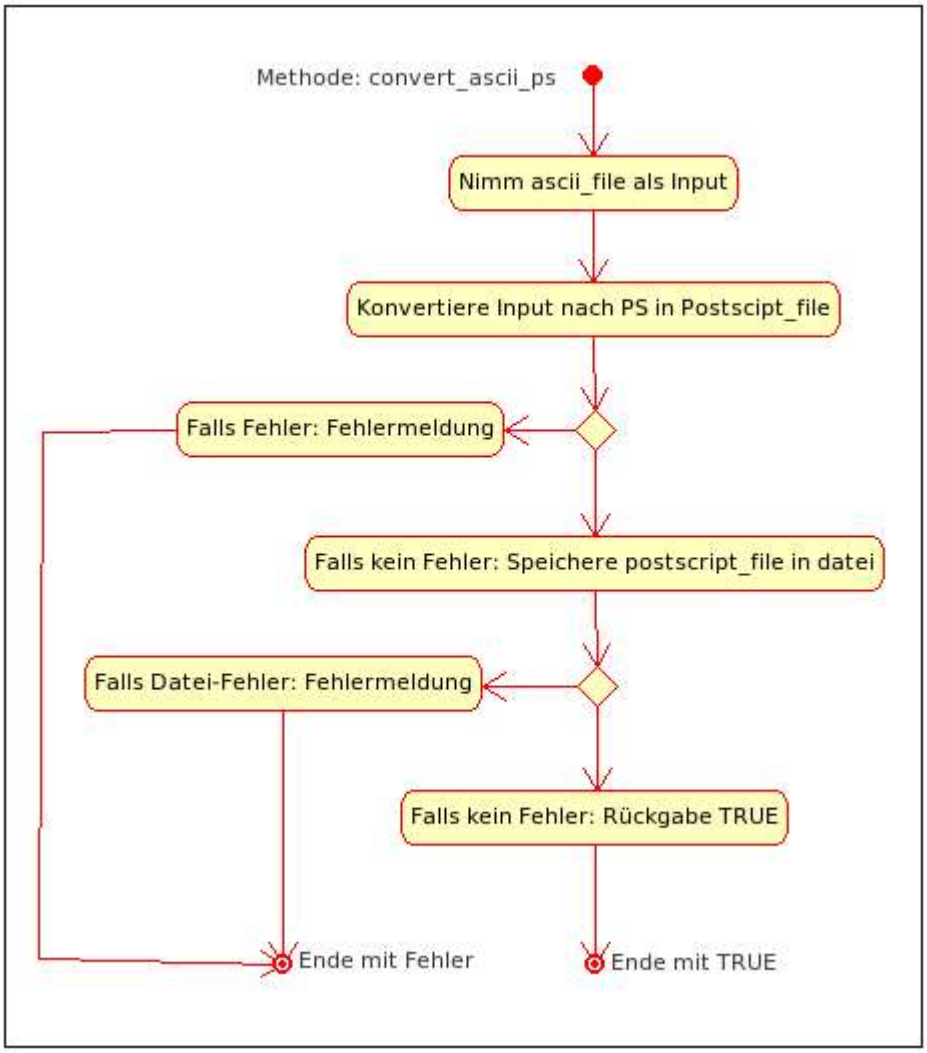

Bild 3: ppmp2ps-Aktivitätsdiagramm für die Methode convert_ascii_ps()

3. Übersetzung von ppmp2ps-PIM2 nach ppmp2ps-PSM2

Die vorausgehenden zusätzlichen UML-Diagramme haben aufgezeigt, wie man die Dynamik des ppmp2ps-Anwendungsfalls ppmp2ps-CIM durch Erweiterung des ppmp2ps-PIM zu ppmp2psPIM2 weiter analysieren kann.

Das opensource-Werkzeug *umbrello* ist noch nicht in der Lage, Diagramme über die Klassendiagramme hinaus automatisch in Sourcekode zu übersetzen. Dies bedeutet, man muss diese zusätzlichen UML-Diagramme (Sequenz und Aktivität) "per Hand" in en Sourcekode einbauen.

```
/*************************************************************************
 name: ppmp2ps.h

author: gerd d-h
first time: april-16, 2005
last change: april-16, 2005
intention: Read an ascii-textfile.txt and convert it into postscript textfile.ps

This file was generated on Sa Apr 16 2005 at 23:41:31
/home/gerd/public_html/fh/II-PPmP/VL/VL3/ppmp2ps.h
*************************************************************************/
#ifndef PPMP2PS_H
#define PPMP2PS_H
#include <string>

/**
 * Class ppmp2ps
 *
 */
class ppmp2ps {
/**
 * Public stuff
 */
public:

  /**
   * Operations
   */
  /**
   *
   * @param file_name Bekommt vom User einen Dateinamen fuer eine ASCII-Datei mit Endung .txt und liest dann diese Datei in einen internen Puffer
   */
  void  fileselect (string file_name=dummy.txt);

  /**
   * Konvertiert einen zuvor ausgewählten ASCII-Text in eine Postscript Datei
   */
  void  convert_ascii_ps ();
```

```
/**
 * Private stuff
 */
private:
  /**
   * Fields
   */
  /**
   * Enthält den Dateinamen der aktuell zu konvertierenden ASCII-Datei
   */
   string file_name;
  /**
   * Ein Textpuffer mit der eingelesenen ASCII-Datei
   */
   string ascii_file;
  /**
   * textpuffer fuer die neu konvertierte Postcript Datei
   */
   string postcript_file;
  /**
   *
   */
};
#endif //PPMP2PS_H
```

Dann die Angabe einer C++-Implementierungsdatei, die zunächst nur einen Rumpf bildet der gerade die Signaturen der UML-Operatoren enthält.

```
/*************************************************************************
name: ppmp2ps.cpp
author: gerd d-h
first time: april-16, 2005
last change: april-16, 2005
```

```
intention: Read an ascii-textfile.txt and convert it into postscript textfile.ps

This file was generated on Sa Apr 16 2005 at 23:41:31
The original location of this file is
/home/gerd/public_html/fh/II-PPmP/VL/VL3/ppmp2ps.cpp
**************************************************************************/
#include "ppmp2ps.h"
#include <string>

/**
 * Methods
 */
/**
 * Wählt einen  ASCII-Text aus und lädt ihn
 */
void ppmp2ps::fileselect (string file_name) {

}
/**
 * Konvertiert einen zuvor ausgewählten ASCII-Text in eine Postscript Datei
 */
void ppmp2ps::convert_ascii_ps () {

/**
* Ende der Implementierungsdatei
*/
```

Als nächstes wird die Implementierung durch Erläuterung der C++-Plattform weiter ausgearbeitet.