MODELLGETRIEBENE ENTWICKLUNG EINER MOBILEN APPLIKATION MIT JUSE4ANDROID

Jano Espenhahn, Tobias Franz and Franziska Krebs
Fachhochschule Brandenburg, Fachbereich Informatik und Medien
{*espenhah, franzt, krebsf*}*@fh-brandenburg.de*

Keywords: MDA, UML, USE, OCL, Android

Abriss: ein deutsches Abstract: Entwicklung einer Applikation, USE-Spezifikation zur Definition eines Klassendiagramms; OCL-Constraints oben drauf;Im Anschluss Generierung der Anwendung mit Hilfe von JUSE4Android. Untersuchung der Anwendung anhand bestimmter Fragestellungen

Abstract: ein englisches Abstract

1 EINLEITUNG

Zitat Test (**?**)

2 BESCHREIBUNG DER ANWENDUNG

Das Beispiel wurde aus dem Artikel (**?**) entnommen. Es handelt sich um ein fiktives Programm der Regierung zur Kontrolle der Eispartikel in der Luft. Wenn die Konzentration zu niedrig ist, bedeutet das, dass die Bevölkerung zu wenig Eiscreme isst, was eine Menge an Risiken für die Umwelt und die öffentliche Ordnung darstellt. Um die Eispartikel in der Luft zu überwachen, hat der Staat Kontrollstationen im gesamten Land verteilt aufgestellt. Für jede Station gibt es einen festgelegten Zielwert der Eispartikel. Der aktuelle Wert weicht in der Regel vom Zielwert ab.
Die Anwendung ermöglicht es neue Stationen mit Zielwerten aufzunehmen und alte Stationen zu löschen. Außerdem gibt es die Möglichkeit eine Adresse zu einer Station anzugeben. Eine Adresse ist im Nachhinein auch wieder entfernbar. Die Erfassung von beliebig vielen Einträgen zu einer Station ist ebenfalls möglich. Auch Einträge lassen sich im Nachhinein wieder entfernen. Zudem wird für jeden Eintrag, nach Eingabe des aktuellen Wertes die Abweichung zum Zielwert angezeigt.

3 VORSTELLUNG USE

UML based Specifiation Environment (USE) wird zur Spezifikation von Informationssystemen verwendet und wurde an der Universität Bremen entwickelt. Neben dem Einsatz für Fallstudien, wird USE vor allem in der Lehre an Hochschule wie z. B. MIT, Cambridge, University of Edinburgh und University of Lisbon eingesetzt. USE basiert auf einer Teilmenge der Unified Modeling Language (UML) und der Object Constraint Language (OCL). Eine USE-Spezifikation besteht aus einer textuellen Beschreibung eines Modells, bei der Eigenschaften aus UML-Diagramm verwendet werden. Weitere Integritätsausdrücke für ein Modell können durch die OCL definiert werden. (**?**)

Die Abbildung 1 veranschaulicht den Workflow für eine USE-Spezifikation. Ein Entwickler spezifiziert ein plattformunabhängiges USE-Modell, welches ein System beschreibt und nutzt dafür UML- und OCL-Ausdrücke. Mithilfe von USE ist es ihm möglich, die bestimmten Anforderungen an sein System auf Erfüllung mit dem Modell zu validieren.

3.1 Spezifikation

Die textuelle Beschreibung eines Modells mit USE beginnt immer mit der Definition eines Modell-Namens. In diesem Fall ist das *IceCream*. Im Anschluss folgen Klassendefinitionen mit ihren jeweiligen Attributen und Methoden. Im Beispiel hat die

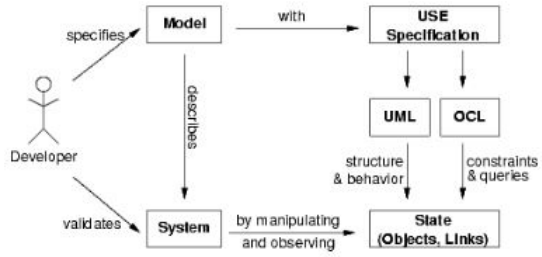

Abbildung 1: Workflow einer USE-Spezifikation (**?**)

Klasse *Station* das Attribut *name* und die Operation *entries* ohne Übergabeparameter. Die nachfolgenden Code-Ausschnitte verwenden lediglich UML.

```
model IceCream

class Station
   attributes
      name         : String
      target       : Integer
   operations
      entries()    : Set(Entry) = self.
          records->asSet
end
```

Klassen können untereinander in Abhängigkeit stehen. Für diese Abhängigkeiten sind Assoziationen vorgesehen. Um eine Assoziation auszudrücken, wird zuerst eine weitere Klasse *Address* eingeführt.

```
class Address
   attributes
      street   : String
      postCode : Integer
end
```

Für das dem Artikel zugrunde liegende Beispiel kann eine Station entweder eine oder keine Adresse haben.

```
association Station_Address between
   Station[ 1 ]
   Address[ 0..1 ] role place
end
```

Station_Address ist dabei der Name der Assoziation und das Attribut *place* nimmt in der Klasse *Station* die Rolle für die Adresse ein. Zum gesamten USE-Modell gehören weiterhin noch die Klasse *Entry* und die Assoziation *Station_Entry*.

```
class Entry
   attributes
      date     : CalendarDate
      actual   : Integer
      variance : Integer
   operations
      variance(): Integer = actual -
          station.target
end
```

```
association Station_Entry between
   Station[ 1 ] role station
   Entry[ * ] role records
end
```

Zur Vervollständigung des Modells gehört außerdem eine aus der Arbeit (**?**) entnommene Klasse *CalendarDate*.

3.2 Erweiterung durch OCL

Als Bestandteil der UML ist die OCL ebenfalls als Spezifikation zur Modellierung von Softwareartefakten zu verstehen. Die Entwicklung der OCL wurde angetrieben durch den Wunsch, zusätzliche Modelleigenschaften - welche nicht mithilfe grafischer Elemente ausgedrückt werden können - festlegen zu können. (**?**, S.5ff) Da diese Aspekte eindeutig und für alle Akteure verständlich sein sollen, wurde die OCL als eine formale und dennoch gut lesbare Sprache konzipiert. Das Vokabular der aktuell in Version 2.4. bereitgestellten Spezifikation ist sehr umfänglich und wird u.a. für die folgenden Zwecke genutzt:

- als Abfragesprache
- zur Definition von Invarianten
- zur Beschreibung von Vor- und Nachbedingungen von Operationen
- zur Definition von Ableitungsregeln für Attribute
- zur Definition von Restriktionen für Operationen

Im Folgenden werden die OCL-Konstrukte erläutert, welche zur textuellen Beschreibung von Bedingungen im IceCream Beispiel verwendet wurden. HINWEIS: keine context Schlüsselwort, da die Aussagen innerhalb der Klassen spezifiziert werden. Annotationen sind JUSE4Android spezifisch und nicht Bestandteil der OCL Für die Klassen Station und Entry wurden die Invarianten *TargetValueCannotBeNegative* und *ActualValueCannotBeNegative* definiert. Diese repräsentieren Aussagen, welche für die Instanzen der jeweiligen Klasse zu jeder Zeit wahr sein müssen. (**?**, S.188)

```
@TargetValueCannotBeNegative(rationale
    ="The defined target value of a
    station cannot be smaller than 0")
      inv TargetValueCannotBeNegative:
         target>=0

@ActualValueCannotBeNegative(rationale
    ="The actual value measured at a
    station cannot be smaller than 0")
      inv ActualValueCannotBeNegative:
         actual>=0
```

Die Annotation bietet die Möglichkeit, die Invariante zu bezeichnen und näher zu erläutern. Nach dem Schlüsselwort *inv* folgt erneut der Bezeichner.

3.3 USE-Tool

Um eine Spezifikation auf nicht-formale Anforderungen zu validieren, kann ein Modell mithilfe des USE-Tools animiert werden. Direkt nach dem Import eines Modells erhält man vom Tool ein Feedback über die Validität der UML- und OCL-Definitionen. Neben der Validierung bietet das Tool weitere Möglichkeiten, wie z. B. die Visualisierung eines Klassen-, Sequenz- oder Objektdiagramms. In der Abbildung 2 finden sich die im Kapitel 3.1 definierten Klassen und Assoziationen als Klassendiagramm wieder.

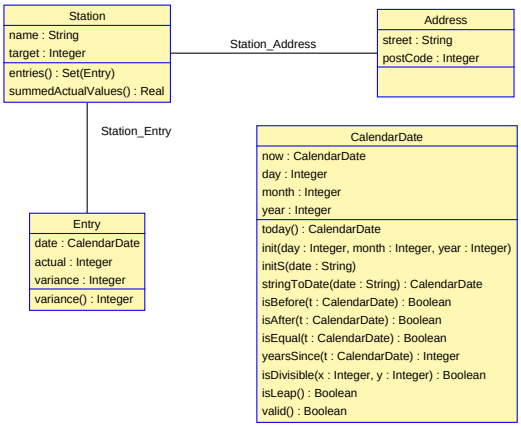

Abbildung 2: Klassendiagramm für das Beispiel

4 JUSE4ANDROID

5 Erstellte Android Applikation

5.1 Untersuchung der erstellten Applikation

Das Tool Juse4Android erstellt aus dem Use-File eine Android-Applikation, welche sich durch folgende Schichten definiert: **Persistenz-Schicht, Presentations-Schicht, Model-Schicht(Buisiness Logik)**. Dabei wurde konsequent das Model-View-View-Model Entwurfsmuster verwendet. Somit ist das Austauschen der Oberfläche theoretisch möglich. Das vorherrschende Architekturmuster ist das Nacked-Object-Pattern. Dieses definiert 3 Prinzipien. 1.) Die gesamte Geschäftslogik wird in Domänen Objekten gekapselt, 2.) Die Benutzeroberfläche ist eine direkte Repräsentation dieser Objekte und 3.) die Benutzeroberfläche kann oder wird direkt aus der Definition dieser Objekte erstellt. Diese 3 Prinzipien, werden von Juse4Android vollständig umgesetzt. Zudem werden die Domänen Objekte direkt in eine Objekt Orientierte Datenbank gespeichert.
Der Programmfluss wird über die 4 grundlegenden Datenbankoperationen(**C**reate **R**ead **U**pdate **D**elete) gesteuert. Dabei werden neue Objekte sowohl in der Datenbank, als auch der View verändert oder neu erstellt.

5.2 Generierter Code und zusätzliche Funktionalität

Der aus der USE-Definition generierte Code ist in einer aussagekräftigen Paketstruktur gegliedert und hält die gängigen Code-Konventionen ein. Negativ muss erwähnt werden, dass dieser nur geringfügig kommentiert wurde und teilweise verschiedene Sprachen für die Kommentare verwendet wurden. Alles in allem ist der Code für einen Fortgeschrittenen Java/Android Entwickler nachvollziehbar und verständlich. Negativ ist zu betrachten, das nur nach der Generierung, manuelle Anpassungen oder zusätzliche Funktionalitäten vorgenommen werden können. Diese werden dementsprechend bei einer erneuten Generierung wieder überschrieben. Ebenfalls ist kein Möglichkeit vorgesehen, geschützte Bereiche zu definieren. Auch das auslagern von Manuellem Code in separate Dateien kann nicht vorgenommen werden, da der gesamte Code, mit Ausnahme von wenigen vordefinierten Standardklassen dynamisch durch die Applikation generiert wird. Somit ist eine Trennung von generiertem Code und Manuellem Code nicht möglich.

5.3 Veränderungen der Benutzeroberfläche

Durch die Verwendung des MVVM Musters, ist die Veränderung oder der Austausch der Benutzeroberfläche theoretisch möglich. Dabei ist aber zu Unterscheiden zwischen dem Austausch der gesamten oder einem Teil der Oberfläche und dem Anpassen und Verändern des Designs.
Die Anpassung des Designs ist schon durch die Gestaltungsmechanismen des Betriebssystems Android ohne Probleme möglich. Ebenfalls können verwendete Grafiken einfach Ausgetauscht werden. Dies ist sogar vor der Generierung möglich.

Annotationen

ANHANG

USE-Spezifikation, welche zur Generierung genutzt wird

*section**{*APPENDIX*}