

# 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 (da Silva, 2014)

## 2 BESCHREIBUNG DER ANWENDUNG

Das Beispiel wurde aus dem Artikel (Fowler, 2006) 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 Specification 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. (Martin Gogolla, 2007)

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 Modellnamens. In diesem Fall ist das *IceCream*. Im Anschluss folgen Klassendefinitionen mit ihren jeweiligen Attributen und Methoden. Im Beispiel hat die

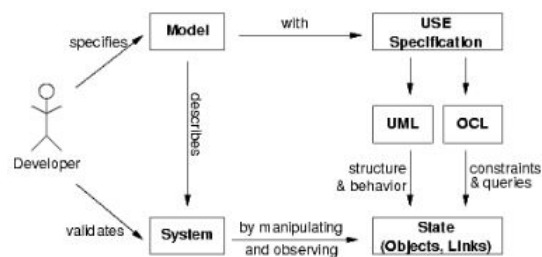


Abbildung 1: Workflow einer USE-Spezifikation (Database Systems Group, Bremen University, 2007)

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
    numberOfEntries : Integer
  operations
    entries() : Set(Entry) = self.records->asSet
    meanActualValue() : Real
    meanVarianceValue() : Real
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
    calculateVariance(): 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 (da Silva, 2014) 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. (Object Management Group, 2014, 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 umfangreich 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. Da diese innerhalb der Klassen spezifiziert werden, kann auf die Einleitung durch das Schlüsselwort *context* verzichtet werden.

Für die Klassen *Station* und *Entry* wurden die Invarianten *TargetValueCannotBeNegative*, *ActualValueCannotBeNegative* und *SelectedDateCannotBeInTheFuture* definiert. Diese repräsentieren Aussagen, welche für die Instanzen der jeweiligen Klasse zu jeder Zeit wahr sein müssen. (Object Management Group, 2014, S.188)

Listing 1: Invariante in der Klasse *Station*

```

inv TargetValueCannotBeNegative:
  target >= 0
  
```

#### Listing 2: Invariante in der Klasse Entry

```
inv ActualValueCannotBeNegative:
    actual >= 0
inv SelectedDateCannotBeInTheFuture:
    date.isBefore(date.today()) or date.
        isEqual(date.today())
```

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.

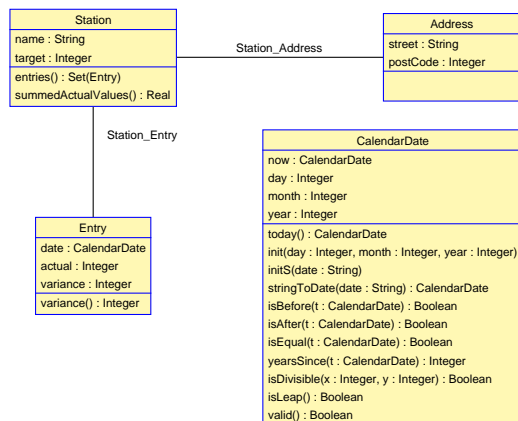


Abbildung 2: Klassendiagramm für das Beispiel

## 4 JUSE4ANDROID

Dieser Abschnitt

### 4.1 Vorstellung

Entstanden in Masterarbeit von ... db4o ist scheiße  
Api-Level 17

### 4.2 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(Business Logik)**. Dabei wurde konsequent das Model-View-View-Model Entwurfsmuster verwendet. Somit ist das Austauschen der Oberfläche theoretisch möglich. Das vorherrschende Architekturmuster ist das Naked-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 **Create Read Update Delete** gesteuert. Dabei werden neue Objekte sowohl in der Datenbank als auch der View verändert oder neu erstellt.

Master-Detail

#### 4.2.1 Abbildung von Assoziationen und OCL-Constraints

Wie bilden sich Assoziationen ab? Wie bilden sich Constraints ab?

#### 4.2.2 Generierter Code und zusätzliche Funktionalität

Der aus der USE-Definition generierte Code bedarf in den meisten Fällen einer manuellen Nachbearbeitung, um syntaktische Fehler der Generierung zu beheben. Diese äußern sich u.a. durch den Aufruf nicht deklarierter Variablen oder durch eine fehlerhafte Kommasetzung in Parameterlisten. Ebenfalls zeigten sich Probleme bei der Konfiguration des Build Paths in Form von nicht hinzugefügten Libraries.

da ist in einer Paketstruktur gegliedert **Bild** und hält die gängigen Code-Konventionen ein. **Einige Konventionen nennen und Oracle zitieren** 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, dass nur nach der Generierung manuelle Anpassungen oder zusätzliche Funktionalitäten vorgenommen werden

können. Diese werden dementsprechend bei einer erneuten Generierung wieder überschrieben. Ebenfalls ist keine 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.

To Do: Bremen Tool sagt, das Modell sei valide; JUSE4Anroid: keine Generierung aufgrund der Definition von abgeleiteten Attributen

#### 4.2.3 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.

i. Einfügen einer alternativen View für die Varianz-Abweichung

JUSE4Android bietet Annotationen an, welche im USE-File verankert sind und durch welche geringfügig auf die View Einfluss genommen werden kann. SIEHE MASTERARBEIT Seite

## 5 Zusammenfassung und Fazit

JUSE4Anroid zu starr; möglicherweise geht es in die Richtung der Generierung von Softwarekomponenten, im Gegensatz zur Generierung einer vollständigen Applikation

## REFERENCES

- da Silva, L. (2014). Model-Driven Generative Programming for BIS Mobile Applications. Master's thesis, ISCTE IUL University of Lisbon.
- Database Systems Group, Bremen University (2007). USE - A UML based Specification Environment. <http://www.db.informatik.uni-bremen.de/projects/USE/use-documentation.pdf>. [letzter Zugriff am 28.01.16].
- Fowler, M. (2006). GUI Architectures. <http://martinfowler.com/eaDev/uiArchs.html>. [letzter Zugriff am 28.01.16].
- Martin Gogolla, Fabian Buttner, M. R. (2007). USE: A UML-based specification environment for validating UML and OCL.
- Object Management Group (2014). Object Constraint Language. <http://www.omg.org/spec/OCL/2.4/>. [letzter Zugriff am 29.01.16].

## ANHANG

USE-Spezifikation, welche zur Validierung durch das USE-Tool genutzt wird Diese weicht ab von der tatsächlich zur Generierung verwendeten USE-Spezifikation Annotationen sind JUSE4Android spezifisch und nicht Bestandteil der OCL `\section*{APPENDIX}`