ENTWICKLUNG EINES DATEITRANSFER-KLIENTEN MITTELS METHODEN DER MODELLGETRIEBENEN SOFTWARE-ENTWICKLUNG

Mario Kaulmann[1], Naoufel Frioui[1] and Carole Noutchegueme[1]

[1] *Fachbereich Informatik und Medien, Technische Hochschule Brandenburg, Magdeburger Straße 50, Brandenburg an der Havel, Deutschland*

Keywords: MDSD, modellgetriebene Software-Entwicklung, md2, Dateitransfer, REST, Android

Abstract: Bei modelgetriebener Software-Entwicklung wird versucht Code zu generieren, ohne tiefgehende Programmierkenntnisse vorauszusetzen. In dieser Arbeit wurde mit MD^2 versucht eine Android-App zu erstellen, mit der man Dateien über einen REST-Service versenden kann. Außerdem soll man sich bei der App über den REST-Service anmelden und registrieren können. Dabei wurden Grenzen von MD^2 ermittelt und ein eigenes speziell auf das Projekt angepasstest Tool (Feature Provider) wurde erstellt, damit fehlende Eigenschaften der App automatisch generiert werden können. Zur besseren Umsetzung des Feature Providers wurde die App auf klassische Weise programmiert, um so Kentnisse darüber zu erlangen, wie der Feature Provider arbeiten muss.

1 EINLEITUNG

1.1 Motivation

Modellgetriebene Software-Entwicklung (MDSD) soll dazu dienen das Entwickeln von Anwendungen so zu vereinfachen, dass Menschen ohne Programmierkenntnisse, Anwendungen erstellen können, die für einen speziellen Anwendungsbereich eingesetzt werden können.
Das Entwickeln einer Smartphone-App ist eine komplexe Aufgabe. Wenn diese App mit einem Server kommunizieren soll, dann wird die Komplexität dieser Aufgabe noch erhöht.
Um trotzdem Ergebnisse erzielen zu können gibt es bereits Ansätze, mit deren Hilfe man durch Beschreibungen des Problems Code erzeugen kann, durch den eine lauffähige Anwendung entsteht.

1.2 Ziel

Das Ziel besteht darin eine Android-App mit Hilfe von Methoden der MDSD zu erstellen. Die App soll über eine REST-Schnittstelle Dateien auf einen Server laden können und Dateien von diesem Server runterladen können. Damit verschiedene Nutzer den Dateitransfer-Dienst nutzen können, soll auch eine Funktion bereitgestellt werden, die es ermöglicht, dass Nutzer sich registrieren, anmelden und abmelden können. Diese Funktionen werden von dem Server bereitgestellt und sind auch über eine REST-Schnittstelle nutzbar.
Bei der Umsetzung dieses Vorhabens wird herausgestellt, was mit den zum Zeitpunkt der Arbeit verfügbaren Mitteln möglich ist und welche Grenzen es noch gibt.

1.3 Aufgabenstellung

Zum Vergleichen des Ergebnisses, das mit Methoden der MDSD erstellt wird, wird vorher ein Prototyp erstellt, der den vollen Funktionsumfang bietet und auf klassische Weise programmiert wird.
Im ersten Schritt der Entwicklung werden Mittel zur modellgetriebenen Entwicklung eines Android-Klienten ausgesucht. Im zweiten Schritt wird versucht den Prototyp des Klienten zu entwickeln. Dabei wird der erzeugte Code anschließend in Android Studio geöffnet und kompiliert. Anschließend wird die App getestet und mit der klassisch programmierten App verglichen, um die aktuellen Grenzen aufzuzeigen und herauszufinden, ob der Prototyp die Anforderungen erfüllt.
Um den Funktionsumfang erfüllen zu können werden einige Teile bei der MDSD-App mit selbstgeschriebenen Code ergänzt.

1.4 Abgrenzung

In dieser Arbeit soll nur ein Android-Klient erstellt werden, der mit einem Server über eine REST-Schnittstelle kommuniziert. Es ist nicht Bestandteil dieser Arbeit eine Server-Anwendung zu erstellen, die die REST-Schnittstelle zur Verfügung stellt.
Das Produkt dieser Arbeit ist ein Prototyp, der durch verschiedene MDSD-Methoden erstellt wird. Der Anteil des selber geschriebenen Codes wird dabei versucht möglichst klein zu halten.

1.5 Ergebnis

Am Ende der Arbeit gibt es eine App, die mit MD^2 generiert wurde und mit einem selbstgeschriebenen Feature Provider verbessert wurde. Die Vergleichs-App, die auf klassische Weise programmiert wurde, hat allerdings mehr Funktionen und bietet schönere Views. Durch das Erstellen des Feature Providers konnten verschiedene Möglichkeiten Quellcode für Android Apps automatisiert zu verändern und neu zu generieren ausprobiert werden.

2 VORGEHEN

2.1 App-Entwicklung

Parallel zu diesem Projekt befindet sich auch der Server, der verwendet werden soll in der Entwicklung. Zum Testen des Servers wurde eine Weboberfläche bereitgestellt.[1]
Um die Umsetzung eines Zugriffs auf einen REST-Service in Android zu testen, wurde ein Prototyp auf klassische Weise programmiert. Das bedeutet, dass der Code in Android Studio ausgehend von einem leeren Projekt entwickelt wird. Dabei werden einerseits Erkenntnisse generiert, wie die Verbindung mit dem REST-Server funktioniert und allgemein, wie die App in Android umgesetzt wird, sodass man sieht, was in welche Dateien geschrieben werden muss und wie diese strukturiert sind.
Außerdem wurde die App auch mit Hilfe von MD^2 umgesetzt. Dabei sieht man die Grenzen die bei MD^2 noch bestehen. Den Code der von MD^2 generiert wurde kann man dann analysieren und mit dem auf klassische Weise erzeugten Code vergleichen. Da bei der Erzeugung des MD^2-Codes ebenfalls Muster entstehen, die im Rahmen der in Android gültigen Strukturen bestehen, kann man diese nutzen, um ein weiteres Programm zu erzeugen, dass die fehlenden Elemente automatisch erzeugt.
Die Abbildung 1 zeigt das Vorgehen bei der Entwicklung der MD^2-App.

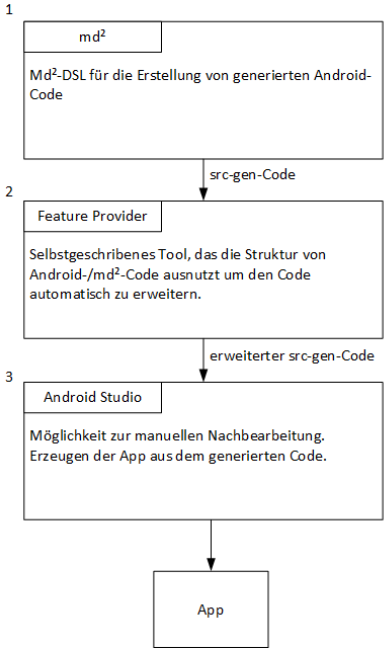

Figure 1: Es werden die einzelnen Arbeitsschritte dargestellt. Die Zahlen geben die Reihenfolge der Schritte an. An den Pfeilen steht die Ausgabe des Vorgängerschritts, die als Eingabe des nächsten Schritts dient.

Der Code, der von MD^2 erzeugt wird, wird in einen Ordner mit dem Namen "src-gen" gespeichert. Auf diesem Code arbeitet der Feature Provider weiter. Das Ergebnis des Feature Providers wird in dem gleichen Ordner gespeichert. Nachdem der Code erweitert wurde kann er mit Android Studio geöffnet werden. Bei Bedarf können manuelle Veränderungen an dem Code vorgenommen werden. In Android Studio kann die App dann kompiliert und getestet werden.

2.2 Feature Provider-Entwicklung

Bei der Entwicklung des Feature Providers werden die Erkenntnisse der klassischen Programmierung ausgenutzt. Dabei werden allgemein gültige Konzepte benutzt, sowie das Wechseln einer Activity mit Hilfe eines Intents. Für die Umsetzung der Kommunikation mit einem REST-Service gibt es verschiedene Bibliotheken. Der Feature Provider benutzt ausschließlich die Bibliotheken, die bei der klassisch programmierten App benutzt wurden.
Zuerst wurde die Benutzeroberfläche entwicklet, die es ermöglicht die entsprechenden Einstellungen auszuwählen. Besonders wichtig ist, die Auswahl des

[1] http://34.238.158.85/MDSD-2017_2018/doc/swagger-ui-master/dist/index.html

”src-gen”-Ordners, da der Absolute Pfad zu diesem Ordner auf jedem Gerät anders ist.
Danach wurde die Funktionalität des Erzeugens der Features implementiert.
Zur Entwicklung der Funktionalität des Feature Providers wurde in den von MD^2 erzeugten Code geschaut an welcher Stelle zusätzlicher Code eingefügt werden muss, oder wo der Code durch anderen ersetzt werden muss. In der klassischen App wurde der Code geprüft, wie die Funktionalität umgesetzt wurde. Bei der Erstellung des Feature Providers wurden dann einige Klassen direkt übernommen mit einigen Veränderungen. Nach der Erzeugung einer neuen Funktion des Feature Providers wurden die in Abbildung 1 dargestellten Arbeitsschritte durchgeführt um zu testen, ob die App dann noch funktioniert und um zu sehen, dass das gewollte Feature hinzugefügt wurde.

3 ENTWICKLUNG

3.1 Schnittstellen

Hier werden die Schnittstellen beschrieben, die mithilfe vom REST-Service vom Server zur Verfügung stehen. Diese basieren auf Standard-HTTP-Operationen.

1. Nutzeroperationen
   - Erstellung eines Nutzers (Post)
     Hier ist die Erstellung von einem Nutzer möglich. Nur registrierte Nutzer können die App benutzen. Dafür braucht man die Parameter:
     – username
     – password
     Antwort 200: Die Operation war erfolgreich. Die Antwort enthält die folgenden Werte:
     – id (ID des home-Ordners)
     – home
       ∗ subfolders (id, name)
       ∗ files (id, name)
     Antwort 400: Schlechte Anfrage, falls der Nutzer ungültige oder schon vorhandene Parameter einträgt (Passwort zu kurz oder Nutzername zu kurz)
   - Nutzer im System einloggen (Post)
     Hier ist das Einloggen von einem Nutzer im System möglich. Diese Funktionnalität ist nur möglich, wenn der Nutzer schon registriert ist. Dafür braucht man:
     – username
     – password
     Antwort 200: Die Operation war erfolgreich. Die Dateien, die zurück geschickt sind, sind:
     – token
     – id (ID des home-Ordners)
     – home
       ∗ subfolders (id, name)
       ∗ files (id, name)
     Antwort 400: Schlechte Anfrage, falls der Nutzer ungültige Parameter einträgt. (Nutzer nicht im System)
   - Delete (Meldet die aktuelle angemeldete Benutzersitzung ab) Hier ist die Abmeldung von der Sitzung möglich. Dafür braucht man den Token vom Nutzer.
     Antwort 200: Die Operation war erfolgreich. Antwort 400: Schlechte Anfrage, der angegebene Token ist nicht vergeben.
2. Dateioperationen
   - Hochladen von Dateien (Post) Hier ist das Hochladen von einer Datei möglich. Dafür braucht man:
     – token
     – folderId
     – file
     Antwort 200: Die Operation war erfolgreich. Die Antwort enthält folgende Parameter:
     – id (id des Ordners)
     – subfolders (id, name)
     – files (id, name)
     Antwort 400: Schlechte Anfrage, Im Fall die Größe der Datei größer als 30Mb ist.
     Antwort 403: Verboten (ungültiger Token)
     Antwort 404: Ordner nicht gefunden
   - Herunterladen von Dateien (Get) Hier ist das Herunterladen von einer Datei möglich. Dafür braucht man:
     – token
     – folderId (Id vom übergeordneten Ordner)
     – fileId (Id von der Datei)
     Antwort 200: Die Operation war erfolgreich. Die Antwort hat folgende Parameter:
     – id (id des Ordners)
     – subfolders (id, name)
     – files (id, name)
     Antwort 403: Verboten (ungültiger Token)
     Antwort 404: Datei / Ordner nicht gefunden
   - Bearbeitung vom Dateiname (Put) Hier ist die Bearbeitung von einer Datei möglich. Dafür braucht man:
     – token

- folderId (Id vom übergeordneten Ordner)
- fileId (Id von der Datei)
- name (Der Name von der Datei)

Antwort 200: Die Operation war erfolgreich. Die Antwort hat folgende Parameter:

- id (id des Ordners)
- subfolders (id, name)
- files (id, name)

Antwort 400: Schlechte Anfrage
Antwort 403: Verboten
Antwort 404: Nicht gefunden

- Löschen von einer Datei (Delete) Hier ist das Löschen von einer Datei möglich. Dafür braucht man:
  - token
  - folderId (Id vom übergeordneten Ordner)
  - fileID (Id von der Datei)

  Antwort 200: Die Operation war erfolgreich.
  Antwort 403: Verboten (ungültige Token)
  Antwort 404: Datei / Ordner nicht gefunden

3. Verzeichnisoperationen
   - Herunterladen von einem Ordner (Get) Hier ist das herunterladen von einem Ordner möglich. Dafür braucht man:
     - token
     - folderId (Id vom Ordner)

     Antwort 200: Die Operation war erfolgreich. Die Antwort hat folgende Parameter:
     - id (id des Ordners)
     - subfolders (id, name)
     - files (id, name)

     Antwort 403: Verboten
     Antwort 404: Nicht gefunden
   - Erstellen eines Ordners (Post)
     Hier ist das Erstellen von einem Ordner möglich. Dafür braucht man:
     - token
     - folderId (id des übergeordneten Ordners)
     - folder (Name des neuen Ordners)

     Antwort 200: Die Operation war erfolgreich. Die Antwort hat folgende Parameter:
     - id (id des Ordners)
     - subfolders (id, name)
     - files (id, name)

     Antwort 400: Schlechte Anfrage, Im Fall die Größe der Datei größer als 30Mb ist.
     Antwort 403: Verboten
     Antwort 404: Nicht gefunden
   - Bearbeitung von einem Ordner (Put)
     Hier ist die Bearbeitung von einem Ordner möglich. Dafür braucht man:
     - token
     - folderId (Id vom übergeordneten Ordner)
     - folder (ein neuer Name)

     Antwort 200: Die Operation war erfolgreich. Die Antwort hat folgende Parameter:
     - id (id des Ordners)
     - subfolders (id, name)
     - files (id, name)

     Antwort 400: Schlechte Anfrage
     Antwort 403: Verboten
     Antwort 404: Nicht gefunden
   - Delete
     Hier ist das Löschen von einem Ordner möglich. Dafür braucht man:
     - token
     - folderId (Id vom übergeordneten Ordner)

     Antwort 200: Die Operation war erfolgreich.
     Antwort 403: Verboten
     Antwort 404: Nicht gefunden

3.2 klassische App

Die klassische App wurde programmiert, um zu testen, wie man eine Android-App mit einem REST-Service verbindet. Dafür wurden ein paar Bibliotheken ausprobiert. Die App setzt alle Funktionen um, die vorher festgelegt wurden. Diese Funktionen sind:

- Datei-Upload
- Datei-Download
- Ordner erstellen
- Nutzer registrieren
- Nutzer anmelden
- Foto machen und hochladen

Die klassische App benutzt die folgenden externen Bibliotheken:

- loopj[2]
  stellt einen asynchronen callback-basierten HTTP-Client basierend auf Bibliotheken von Apache zur Verfügung.
- okio[3]
  erleichtert den Zugriff, Speicherung und die Verarbeitung von Dateien.
- EasyPermission[4]
  ist eine Wrapper-Bibliothek zur Vereinfachung der grundlegenden Systemberechtigungslogik für Android.

[2] https://github.com/loopj/android-async-http
[3] http://square.github.io/okio/1.x/okio/
[4] https://firebaseopensource.com/projects/googlesamples/easypermissions/

- okhttp[5]
  wird benutzt um schnelle Input- und Output-Operationen durchzuführen und größenveränderbare Puffer zu benutzen.
- Retrofit[6]
  wird genutzt, um die Dateitransfers zu ermöglichen. Es können auch synchrone und asynchrone Anfragen über HTTP gesendet werden.

4 WERKZEUGE ZUR ENTWICKLUNG

4.1 MD^2

MD^2-DSL ist eine domänenspezifische Sprache (DSL), die entwickelt wurde um datengetriebene Business-Apps in textueller Form beschreiben zu können (Heitkötter et al., 2013).
MD^2 ist ein akademischer Prototyp, der auch Anwendung in praktischen Projekten findet und dabei auch weiterentwickelt wird (Majchrzak et al., 2015).
In diesem Projekt wurde die aktuelle Version von MD^2 verwendet, die auf dem git-Repository[7] zur Verfügung stand (Stand: 23.11.2017). Diese steht als Plugin für Eclipse zur Verfügung.
MD^2 ist nach dem MVC Prinzip aufgebaut und hat dazu Workflow-Elemente, die spezielle Controller-Elemente sind (Kuchen and Ernsting, 2015). Durch das erste Workflow-Element wird immer beim Build des Projekts eine "StartActivity" erzeugt, deren View Buttons enthält zum Starten der Programmierten App. Der Umfang, der in dem Handbuch für Modellierer versprochen wurde konnte nicht komplett genutzt werden. Außerdem gab es Standardfunktionen, wie das Schließen der App, die nicht umgesetzt waren.
Nach jedem Build des Projektes wurde der alte Code komplett ersetzt durch den neu generierten Code. Dadurch wurden auch manuelle Erweiterungen des Codes zerstört. Aus diesem Grund entstand die Idee, ein Tool zu entwickeln, das automatisch die Erweiterungen des Quellcodes wieder hinzufügt, die die Funktionalität der App erweitert haben.

4.2 Android Studio

In diesem Projekt wird die freie integrierte Entwicklungsumgebung Android Studio zum Testen der MD^2-App verwendet und um den klassisch programmierten Client zu implementieren. Dadurch können Tests auf Smartphones und Emulatoren durchgeführt werden. Dabei wird auch festgestellt, ob der erzeugte Code von Eclipse und den zusätzlich installierten Plugins unabhängig ist.

4.3 Feature Provider

Der Feature Provider ist ein Tool, das nur für die in diesem Projekt entwickelten App erzeugt wurde. Es ist ein Prototyp, der bei seiner Entwicklung dazu beiträgt, die Probleme bei MDSD zu verstehen. Ein paar Features können in Verbindung mit einer MD^2-App allerdings allgemein genutzt werden.
Nachfolgend werden ein paar Beispiele gegeben, wie der Feature Provider arbeitet.
Das Grund-Feature, dass der Feature Provider immer hinzufügt ist, dass eine App geschlossen wird, wenn man die erste View sieht und auf den Back-Button drückt. Der Code, der von MD^2 in der "StartActivity" für die Aktion "onBackPressed" erzuegt wird sieht wie folgt aus.

```
public void onBackPressed() {
// remain on start screen
}
```

Hier ist zu erkennen, dass diese Methode keine Funktion enthält. Aus diesem Grund hat es keine Wirkung, wenn man auf den "Back Button" drückt.
Der Feature Provider öffnet die Java-Datei dieser Activity und sucht nach dem Methodenkopf der "onBackPressed"-Methode. Der Kommentar wird dann durch den Code zum Schließen der App ersetzt. Dabei entsteht der folgende Code aus (Basri, 2015):

```
public void onBackPressed() {
Intent intent = new Intent(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_HOME);
intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
System.exit(0);
}
```

Für die Erzeugung einiger Features müssen mehrere Dateien verändert werden. Zum Beispiel bei der Erzeugung der Funktionen zur Kommunikation mit dem REST-Service. Dafür wird eine Bibliothek[8] verwendet, die nicht zum Standard von Android gehört. Diese muss dann in die "build.gradle"-Datei im App-Verzeichnis im Abschnitt "dependencies" eingetragen werden. Hier scheint die Reihenfolge der Eintragungen nicht wichtig zu sein, also kann die Eintragung auch am Anfang der Datei erfolgen. Der Feature Provider sucht in der Datei nach der Zeichenkette

[5] http://square.github.io/okhttp/
[6] http://square.github.io/retrofit/
[7] https://github.com/wwu-pi/md2-framework
[8] http://loopj.com/android-async-http/

”dependencies” und fügt danach die neue Bibliothek hinzu.
Die Erzeugung der Klassen zur Kommunikation mit dem REST-Service erfolgt mit Hilfe von Klassen, die aus der klassischen App kopiert werden. Diese wurden im Vorfeld so verändert, dass sie in das Projekt nur eingefügt werden müssen und anschließend nur noch verwendet werden müssen. Dafür hat der Feature Provider einen Ordner (Quelldateien), in der vollständige nutzbare Klassen abgelegt sind.
Außerdem ist es notwendig die Android-Manifest-Datei zu verändern, da man für die REST-Services die Berechtigung benötigt ins Internet zu gehen.
Wenn eine Veränderung einer Klasse dazu führt, dass eine neue Klasse verwendet wird ist es notwendig, dass entsprechende Import-Statements am Anfang der Klasse ergänzt werden. Da die Reheihenfolge hier beliebig ist, werden die neuen Importe direkt nach dem Eintrag ”package” gemacht.
Die Erweiterung einer View benötigt mindestens Veränderungen einer Activity, einer XML-Datei zu der View und der ”ids.xml”-Datei. Die richtige Platzierung ist in der XML-Datei zur View nötig, da diese die Position auf der View beeinflusst. In diesem Projekt wurde implementiert, dass zwei ListView-Elemente in der ”DateiDownloadActivity” hinzugefügt werden. Die Platzierung in der View kann dabei frei gewählt werden. Dafür werden die IDs für die View-Elemente dem Benutzer des Feature Providers gezeigt und dieser kann wählen, nach welchem Element er die neue ListView einfügen will. Außerdem werden die IDs der neu hinzugefügten Elemente in der ”ids.xml”-Datei ergänzt. In der Activity wird dann die Lebenszyklusmethode ”onCreate” so verändert, dass die ListViews den gewünschten Inhalt anzeigen.
Zum Einfügen einer Aktion werden die Klassen im Package ”md2.testprojekt.md2.controller.action” verändert. Hier liegen die Klassen, die für die erzeugten Aktionen benutzt werden. Z.B. wenn man einen Wechsel der Views erzeugt hat, der mit Hilfe eines Buttons realisiert wird.

5 ZUSAMMENFASSUNG UND AUSBLICK

Das Werkzeug MD^2 hat sich in der verwendeten Version als nicht so mächtig erwiesen, wie es gedacht war, weshalb nur ein geringer Anteil der App mit MD^2 umgesetzt werden konnte. Die Entwicklung eines Feature Providers, der auf dem von MD^2 erzeugten Code weiterarbeitet bot eine Möglichkeit das Erzeugen von Quellcode selber umzusetzen, auch wenn der Feature Provider ein sehr eingeschränktes Werkzeug ist, das nur bei diesem Projekt eingesetzt werden kann. Die Umsetzung aller gewünschten Funktionen konnte nicht für die App realisiert werden, die mit MD^2 erzeugt wurde. Allerdings gibt es eine klassisch programmierte App, die den vollen Funktionsumfang bietet, die als Grundlage für Überlegungen zur Erweiterung des Feature Providers dienen kann. Zukünftig könnten noch weitere Funktionen umgesetzt werden, der Feature Provider könnte allgemeiner gemacht werden, sodass er für beliebige Anwendungen anwendbar ist.
Methoden der MDSD können gut eingesetzt werden, um Android Apps zu generieren. Dafür haben Android Apps entsprechende Strukturen, die sich in der Aufteilung der Code-Elemente wiederfinden. Als Beispiel dafür seinen die Lebenszyklusmethoden genannt und die Aufteilung der View und Activity in verschiedenen Dateien.

QUELLEN

Basri, H. (2015). https://gist.github.com/CreatorB/99cdb013a4888453b8a0 (Zugriff: 04.01.2018).

Heitkötter, H., Majchrzak, T. A., and Kuchen, H. (2013). Md2-dsl eine domnenspezifische sprache zur beschreibung und generierung mobiler anwendungen. *Proceedings der 6. Arbeitstagung Programmiersprachen (ATPS), Software Engineering*.

Kuchen, H. and Ernsting, J. (2015). Md2 handbook. Technical report, University of Münster.

Majchrzak, T. A., Ernsting, J., and Kuchen, H. (2015). Model-driven cross-platform apps: Towards business practicability. *Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015)*.