Fachhochschule Nordwestschweiz
Hochschule für Technik

Institut für Mobile und Verteilte Systeme

FOKUS REPORT

2012

Editorial

Ziemlich genau dreissig Jahre dauerte es, bis aus den 1936 geäusserten theoretischen Überlegungen von Church, Turing und Post zur Berechenbarkeit Computer entstanden sind, welche eine von der Erde aus berechnete und gesteuerte, weiche Mondlandung der sowjetischen Luna 9 im Jahr 1966 ermöglichten. Bereits zehn Jahre später landete der Lander von Viking auf der Marsoberfläche und 1996 erkundete der kleine Rover von Pathfinder die Marsoberfläche und sandte erste hochauflösende Bilder zur Erde. Während sich die NASA auf den ersten bemannten Flug zum Mars vorbereitet, geistert in den Köpfen von einigen Astrophysikern und Fantasten die Idee durch den Kopf, das von der Sonne aus nächstgelegene Sternsystem Alpha Centauri mit laserbetriebenen Segelschiffen zu erkunden. Die Distanz bis dorthin beträgt ungefähr 4.4 Lichtjahre und würde mit einem heutigen Spaceshuttle gut hunderttausend Jahre dauern. Dieses Doppelsternsystem ist nicht nur wegen seiner äusserst geringen Nähe zur Erde von Interesse, sondern auch deshalb, weil kürzlich Genfer Astronomen einen Planeten entdeckten, der Alpha Centauri umrundet, und somit einmal mehr die Frage nach extraterrestrischem Leben aufkeimte. Bei einer geschätzten Milliarde von Planeten mit lebensfreundlicher Umgebung und Wasser im ganzen Universum, ist die Vorstellung von ausserirdischem Leben durchaus vertretbar.

Die gleichen Astrophysiker gehen davon aus, dass in zehn bis zwanzig Jahren der Nachweis für ausserirdisches Leben erbracht sein könnte. Ein solcher Nachweis mag dann wohl in gewissen Physikerkreisen akzeptiert werden. Um weitere Kreise davon zu überzeugen, wird es aber wesentlich mehr brauchen, als die Analyse des vom Planeten reflektierten Lichtspektrums, welches Hinweise auf die chemische Zusammensetzung der Atmosphäre und Planetenoberfläche und somit auf potentielle biologische Lebensformen liefern kann. Denn je nach Theorie, welche es gerade zu stützen gilt, verändert sich das Licht auf seiner langen Reise nicht oder wird eben durch sehr starke Gravitationsfelder abgelenkt und somit auch beeinflusst.

Falls extraterrestrisches Leben nachgewiesen wird, bleibt die Frage, ob es sich dabei um intelligente Lebewesen handelt oder nicht. Genügt es, wenn die Lebewesen den Turing-Test bestehen, d.h. einem menschlichen Juror in einem Frage-Antwort-Spiel ein menschenähnliches Denkvermögen vorgaukeln können, oder müssen sie in der Lage sein, über die eigene Herkunft und über das Universum nachzudenken, um als intelligent zu gelten? Nun, ziemlich offensichtlich ist, dass es nicht ausreicht, über eine riesige Informationsmenge zu verfügen. Ein Wikipedia im Speicher

Inhalt

Impressum

Herausgeberin:
Fachhochschule Nordwestschweiz FHNW
Institut für Mobile und Verteilte Systeme
Steinackerstrasse 5
CH-5210 Brugg-Windisch
www.fhnw.ch/technik/imvs
Tel +41 56 462 44 11

Kontakt: Marc Dietrich
info-imvs@fhnw.ch
Tel +41 56 462 46 73
Fax +41 56 462 44 15

Redaktion: Prof. Dr. Christoph Stamm
Layout: Andreas Hildebrandt
Erscheinungsweise: jährlich
Druck: jobfactory Basel
Auflage: 150

ISSN: 1662-2014

und eine Anzahl von Sensoren, um neue Daten zu erfassen, machen alleine noch nicht intelligent. Vielleicht bedarf es aber „nur“ noch zusätzlich der logischen Induktion. Diese hat die Funktion durch das Anführen von Einzelfällen das Allgemeine deutlich zu machen und erlaubt die Einzelfälle durch eine Verallgemeinerung zusammenzufassen. Falls diese Verallgemeinerung nicht hundertprozentig korrekt ist, aber dennoch eine sehr hohe Plausibilität aufweist, so schwindet der Unterschied zum menschlichen Denkprozess rapide.

Monate bevor Felix Baumgartner mit seinem atemberaubenden Sprung aus der Stratosphäre Millionen von Zuschauern zum Staunen brachte, haben Matthias Krebs, Martin Schindler und Marcus Hudritsch mit bescheidenen finanziellen Mitteln einen mit Helium gefüllten Wetterballon auf eine Höhe von 33'166 m ansteigen lassen und dabei eindrückliche Bilder gemacht. Bis vor wenigen Jahren war es alleine den Raumfahrtorganisationen vorbehalten, Bilder der Erde aus der Stratosphäre oder dem Weltall zu erstellen. Mit modernen Technologien wie Mobilfunk, GPS und digitalen Kameras können nun auch ambitionierte Hobby-Weltraumforscher Bilder der Erde aus der Stratosphäre erstellen.

Der Nutzung Cloud basierter Dienste zur Kostenreduktion und Verbesserung der Verfügbarkeit stehen viele IT-Verantwortliche skeptisch gegenüber, da sie berechtigte Sicherheitsbedenken haben. Hannes Lubich zeigt in seinem Artikel auf, wie mit technischen und organisatorischen Massnahmen eine hinreichend sichere Nutzung von Clouds möglich ist. Neben den verschiedenen Cloud-Diensten beschreibt er auch deren Auswirkungen auf die Informationssicherheit und das Risikomanagement. Zudem stellt er einen Katalog von dreizehn Sicherheitsanforderungen an Cloud-Computing-Umgebungen vor.

SmartPhones gehören auch bei Kindern und Jugendlichen zu den ständigen Begleitern. Meistens verwenden diese die Geräte für die Organisation und die Ausfüllung ihrer Freizeit, viel seltener jedoch für Aus- und Weiterbildungszwecke. Doch auch in diesem Segment können SmartPhones einen wertvollen Dienst leisten. Zdena Koukolíková und Carlo U. Nicola stellen ihre Anwendung *DashInEnglish* vor, welche das Texteingabewerkzeug Dasher auf Android implementiert und für die Eingabe von englischen Begriffen einsetzt. In ihrem Bericht gehen sie primär auf die notwendigen Anpassungen einer mobilen Anwendung ein, wenn die App sowohl auf einem SmartPhone als auch auf einem Tablet zum Einsatz kommen soll.

Die Autoren Christoph Denzler, Daniel Kröni und Maxim Moschko beschreiben in ihrem Beitrag die Schwierigkeiten, die beim Anbieten von nativen, mobilen Anwendungen auf mehreren Plattformen auftreten können und wie mit einer zentralen Testinfrastruktur die Äquivalenz des Verhaltens der verschiedenen Implementierungen der Komponenten (ohne GUI) überprüft werden kann. Da zum Startzeitpunkt ihres Projektes noch keine tauglichen Cross-Compiler für mobile Plattformen vorhanden waren, entschied man sich für einzelne, plattformspezifische Umsetzungen einer zuvor erstellten Referenzarchitektur. Im Gegenzug wird die Testentwicklung und -ausführung auf einem zentralen Server konzentriert, welcher über einfach gehaltene Schnittstellen Zugriff auf die mobilen Anwendungen erhält und dadurch die Ausführung identischen Testcodes auf allen unterstützten Plattformen erlaubt.

Die verschiedenen Klassen von mobilen Anwendungen (nativ, Web, hybrid) werden im Artikel von Andreas Hildebrandt und Chris Yereaztian einander gegenübergestellt und bezüglich der Anforderungen an eine fiktive Ticketing-Applikation verglichen. Dabei zeigt sich deutlich, dass die Grenze zwischen HTML5 Web-Anwendungen und nativen Apps immer undeutlicher wird, weil auch in Web-Anwendungen immer mehr das Bedürfnis aufkommt, auf gerätespezifische Sensoren zuzugreifen.

Domänenspezifische, formale Sprachen (DSL), wie zum Beispiel HTML und VHDL, werden für ganz bestimmte Problemfelder entworfen. Beim Entwurf derselben wird darauf geachtet, einen möglichst hohen Grad an Problemspezifität zu erreichen, so dass möglichst alle Problemstellungen innerhalb des Fachbereichs dargestellt werden können, aber nicht mehr. Markus Knecht und Jürg Luthiger beschreiben in ihrem Aufsatz die Entwicklung einer DSL-Engine auf Basis von Groovy, welche es ermöglicht, eine Fachsprache zu implementieren, deren Syntax und Semantik jederzeit an die sich ständig verändernden Bedürfnisse der Domänenspezialisten angepasst werden kann.

Bei internationalen Tennisturnieren werden vielfältigste Informationen zu den Spielen zusammengetragen und dem Publikum so schnell wie möglich zur Verfügung gestellt. Dazu zählen Aufschlaggeschwindigkeiten oder auch Simulationen des Ballflugs wie bei Hawk-Eye. Bei entsprechenden Badmintonturnieren ist die digitale Unterstützung der Zuschauer noch nicht im gleichen Mass fortgeschritten. Beispielsweise fehlt es nach wie vor an einer Messeinrichtung, welche die sehr hohen Anfangsgeschwindigkeiten von Schmetterbällen ermitteln kann. Christoph Stamm stellt einen Ansatz vor, wie mit Hilfe von Hochgeschwindigkeitskameras die Ballgeschwindigkeit in einem laufenden Spiel bestimmt werden kann. Dabei verweist er auf Tests, welche an den Badminton SwissOpen in Basel mit einem Vorläufer des aktuellen Systems gemacht worden sind.

Prof. Dr. Christoph Stamm
Forschungsleiter IMVS

Am Rande des Weltalls

Bis vor einigen Jahren war es alleine den Raumfahrtorganisationen wie der NASA vorbehalten, Bilder der Erde aus dem Weltall zu erstellen. Heutzutage stehen aber immer mehr moderne Technologien wie Mobilfunk, GPS und hochauflösende Kameras jedermann zur Verfügung und dies zu erschwinglichen Preisen. Deshalb sind nun auch ambitionierte Hobby-Weltraumforscher in der Lage, mit geringem finanziellem Aufwand erstaunliche Bilder der Erde aus der Stratosphäre zu produzieren und Messdaten zu sammeln.

Marcus Hudritsch, Matthias Krebs, Martin Schindler | matthias.krebs@fhnw.ch

In den letzten zwei Jahren sind bei YouTube immer wieder aufs Neue spektakuläre Videos aufgetaucht, in denen Aufnahmen der Erde aus rund 30 km Höhe zu sehen sind. Das Grundkonzept ist dabei immer das Gleiche: An einem mit leichtem Gas, für gewöhnlich Helium, gefüllten Wetterballon wird eine Kapsel mit Messinstrumenten befestigt. Der Wetterballon und im Schlepptau die Messinstrumente, meistens Kameras und GPS-Peilsender, manchmal aber auch kleine Computer mit Messsensoren und Funkantennen, steigen in luftige und weniger luftige Höhen, bis der Gasdruck im Ballon bezüglich dem Aussendruck so gross wird, dass der Ballon platzt und der schnelle Sinkflug beginnt. Dank eines kleinen Bremsfallschirms kann die Sinkgeschwindigkeit gebremst werden, so dass die Kapsel mit den Messinstrumenten nicht allzu hart auf dem Boden aufschlägt – sofern sie überhaupt auf dem Boden landet.

Beflügelt wurde diese Idee vor allem durch die Verfügbarkeit von Outdoor-Kameras wie den *GoPro*-Modellen, welche auch in rauen Umgebungen funktionieren und Videoaufnahmen in HD-Qualität ermöglichen. Dank GPS-Peilsendern und Mobilfunk kann die Box auch nach der Landung geortet und geborgen werden, vorausgesetzt die Messinstrumente haben die Landung überstanden.

Mit einem Wetterballon können Höhen von 30 bis 40 km erreicht werden. Damit befindet man sich mitten in der Stratosphäre. Strenggenommen darf man bei dieser Höhe noch nicht vom Weltall sprechen, denn dieses beginnt gemäss Definition erst bei ca. 100 km Höhe. Dennoch darf man getrost behaupten, so nahe am Weltall zu sein, wie es ohne den Einsatz von Raketen und damit grossem finanziellen Aufwand gerade noch möglich ist.

Aufbruch ins Unbekannte

Inspiriert durch verschiedene Wetterballonprojekte und YouTube-Videos haben wir Mitte 2011 das Projekt *M3 Space* ins Leben gerufen [M3Sp]. Am 16. Oktober 2011 starteten wir unseren ersten Versuch in der Nähe von Basel. Die Landung war in etwa 50 km Entfernung in südöstlicher Richtung geplant. Mit an Bord waren eine *GoPro*-Kamera für HD-Videos, ein GPS-Tracker, welcher Positionsdaten via GPRS übermittelt und ein einfacher Funkpeilsender.

Aufgrund einer Fehlberechnung der Füllmenge stieg der Ballon viel langsamer als erwartet. Das Signal des GPS-Trackers brach in ca. 6000 m Höhe ab, da kein GSM-Empfang mehr möglich war. Erst vier Tage später stellte sich heraus, dass der Ballon über 1000 km in östlicher Richtung zurückgelegt hatte und in Rumänien gelandet war. Die Bitte einer Rücksendung der Kapsel blieb leider erfolglos, womit die Kapsel und damit auch die Daten verloren waren.

Nach den negativen Erfahrungen beim ersten Versuch wurde viel Zeit in die Verbesserung der Hard- und Software investiert, um einerseits neben Bildmaterial auch Messdaten zu sammeln und diese mit Hilfe einer Funkverbindung in Echtzeit zu übertragen.

Die Kapsel

Zur Unterbringung der Instrumente bedienen wir uns einer einfachen Box aus Styropor. Dieses Material ist leicht, isoliert sehr gut gegen die Kälte und bietet genügend Schutz vor einem Aufprall. Im Innern ist die Box zusätzlich mit Einlagen versehen, damit die Instrumente nicht verrutschen. Bei unserem zweiten Ballonstart haben wir folgende Messinstrumente in der Box untergebracht (Abb. 1):

- Mikrocontroller-Board als Steuereinheit
- GPS-Modul zur Lokalisierung
- Funkmodul zur Echtzeit-Datenübertragung
- Kameramodul zur Übertragung von Live-Bildern
- Temperatursensor innen und aussen
- *GoPro*-Kamera zur Aufzeichnung von HD-Videos
- GSM GPS-Tracker für die Lokalisierung nach der Landung

Als zentrale Steuereinheit verwenden wir ein *FEZ Panda II* Mikrocontroller-Board [FEZP]. Dieses ist mit einer ARM-CPU ausgestattet, besitzt 512 KBy-

te Flash-Speicher und 96 KByte RAM. Programmiert wird der Chip mit Hilfe des *.NET Micro Framework 4.0*, weshalb wie mit dem .NET Framework für Windows objektorientierte Programme entwickelt werden können, wenn auch mit einigen Einschränkungen. Dank vier RS232-Schnittstellen, I2C-Bus, A/D-Wandlern und vielen digitalen I/Os sind ausreichend Anschlussmöglichkeiten für Sensoren und Erweiterungsmodule vorhanden.

Während des Flugs werden die Telemetriedaten auf einer SD-Karte gespeichert. Damit die Daten auch in Echtzeit via Funk übertragen werden können, ist zusätzlich ein *XBee Pro 868* Funkmodul eingebaut [XBEE]. Es hat bei Sichtverbindung eine Reichweite von bis zu 40 km, was für einen Wetterballon in 30 km Höhe gerade ausreichend ist.

Die Videoaufnahmen werden mit einer *GoPro HD Hero* Kamera im 1080p-Format (Full-HD) produziert. Die Kamera ist horizontal ausgerichtet, damit der Horizont und damit die Erdkrümmung sichtbar werden. Als kleine Zugabe haben wir ein nach unten gerichtetes Kameramodul installiert, welches mit der Steuereinheit verbunden ist. Auf diese Weise werden alle fünf Minuten Live-Bilder via Funk übertragen.

Neben Bilddaten werden auch Messdaten aus der Atmosphäre gesammelt. Zwei analoge Temperatursensoren messen sowohl die Innen- als auch die Aussentemperatur und ein ebenfalls analoger Drucksensor ermittelt den Luftdruck.

Um die Position des Ballons vom Boden aus ermitteln zu können, sind zwei unabhängige GPS-Empfänger in der Kapsel installiert. Ein *uBlox* GPS-Modul ist an die Steuereinheit angeschlossen, damit die Positionsdaten via Funk an die Bodenstation übermittelt werden können. Für den Notfall ist zusätzlich ein GSM GPS-Tracker mit an Bord, damit die Position auch nach der Landung festgestellt werden kann, falls keine direkte Funkverbindung mehr möglich ist.

Ground Control

Für die Verfolgung des Ballonflugs haben wir ein Auto mit der notwendigen Elektronik ausgerüstet und damit zu einer mobilen Bodenstation umfunktioniert. Auf dem Dach des Autos haben wir einen Empfänger (*XBee Pro 868*) für die Telemetriedaten angebracht. Im Innern des Autos führen wir ein Notebook mit, welches via USB mit dem Empfänger auf dem Dach verbunden ist und dadurch die empfangenen Daten entgegennimmt. Mit Hilfe der eigens entwickelten C#-Software *GroundControl* werden die Daten grafisch angezeigt und aufgezeichnet. Damit der Flug auch im Internet live mitverfolgt werden kann, sendet *GroundControl* die Daten via UMTS-Modem an den Webserver von *M3 Space* [M3Sp].

In Abbildung 3 ist *GroundControl* ersichtlich. Das Log-Fenster protokolliert empfangene Datenpakete und Fehlermeldungen. Im Telemetrie-Fenster werden die aktuellen Messwerte als Text dargestellt. Das Map-Fenster zeigt eine Karte basierend auf Google Maps und stellt darin die Flugbahn des Ballons basierend auf den empfangenen GPS-Daten als Überlagerung dar. Dies wird mit Hilfe der Library *GMap.NET* realisiert [GMAP]. Im Live-Image-Fenster wird jeweils das zuletzt empfangene Live-Bild angezeigt. Das Graph-Fenster stellt schliesslich den zeitlichen Verlauf der Messwerte als Diagramme dar.

Im zweiten Versuch erfolgreich

Der zweite Startversuch mit der oben beschriebenen Hard- und Software fand am 13. August 2012 in der Nähe des Bielersees statt. Einige Tage zuvor wird die voraussichtliche Flugbahn des Ballons mit Hilfe eines *Online-Predictors* [CUSF] basierend auf meteorologischen Daten berechnet (Abb. 4). Die Winddaten sind sehr präzise, so dass mit den richtigen Parametern eine Vorhersage auf wenige Kilometer genau möglich ist. Der grüne Marker zeigt den Ort, an dem der Ballon voraus-

Abbildung 1: Die Instrumente der Kapsel

Abbildung 2: Startvorbereitungen der Kapsel

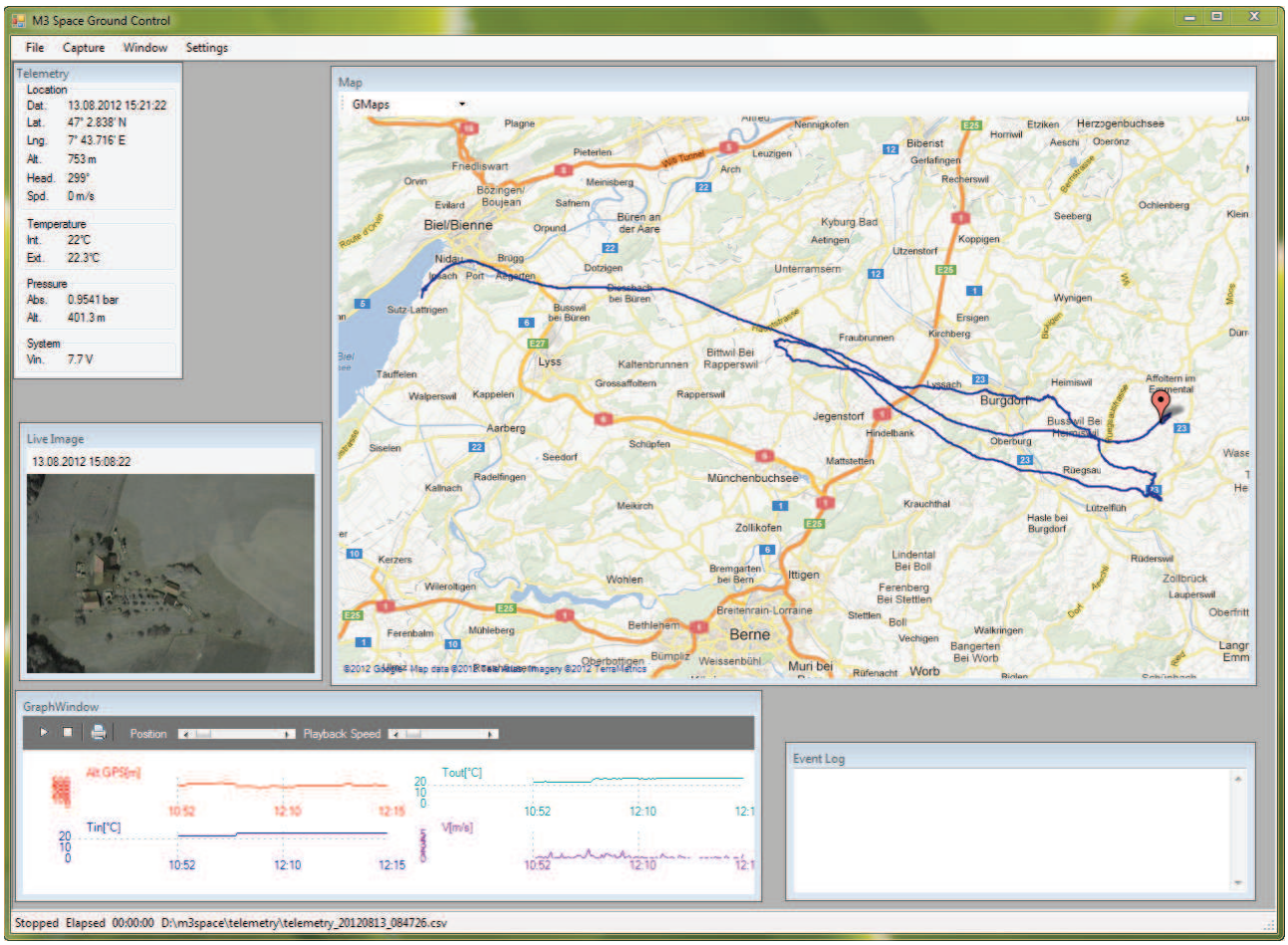


Abbildung 3: Die "GroundControl"-Software

sichtlich platzt, und der rote Marker zeigt den Landeort.

Bei den Startvorbereitungen (Abb. 2) ist die korrekte Füllung des Ballons mit Helium entscheidend. Um die gewünschte Steiggeschwindigkeit von 5 m/s mit einer Nutzlast von 600 g zu erreichen, wird ein vorausberechnetes Referenzgewicht von 1.3 kg am Ballon befestigt und so lange Helium eingefüllt, bis der Ballon das Referenzgewicht vom Boden zu heben vermag. Die während des Ballonaufstiegs übermittelte Steiggeschwindigkeit von 4 bis 5 m/s bestätigt unsere Berechnungen.

Nach über zwei Stunden Aufstieg erreicht der Ballon schliesslich die maximale Höhe von 33'166 m. Dann platzt er. Abbildung 5 und die Umschlagrückseite zeigen Aufnahmen der Erdoberfläche aus dieser Höhe. Die Erdkrümmung ist zwar noch nicht so ausgeprägt, aber dennoch deutlich sichtbar. In der Mitte ist der Genfersee zu erkennen. 40 Minuten später landet die Kapsel in einem kleinen Wäldchen im Emmental. Da sie sich in 30 m Höhe in einem Baum verfängt, kann sie erst zwei Tage später mit Hilfe kleiner Heliumballone, an denen eine Drachenschnur und Haken befestigt sind, geborgen werden.

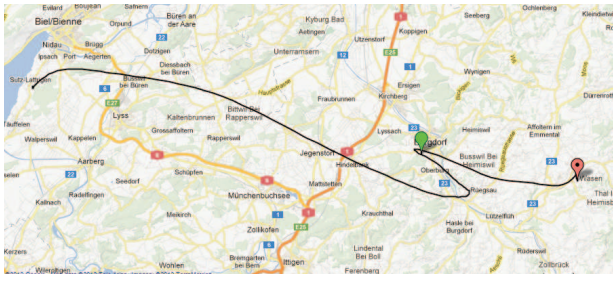


Abbildung 4: Die vorhergesagte Flugbahn des Ballons

Datenanalyse

Wie bereits erwähnt, werden während des Flugs verschiedene Messdaten gesammelt. In Abbildung 6 ist der zeitliche Verlauf der Flughöhe, der Innen- und Aussentemperatur ersichtlich. Weil sich der Ballon während des Steigflugs mit abnehmendem Luftdruck ausdehnt, bleibt der Auftrieb bis zum Platzen beinahe konstant. Nach dem Platzen erreicht die Kapsel eine maximale Geschwindigkeit von 230 km/h, bevor sie durch den Bremsfallschirm gebremst wird. Da der Luftdruck in

Abbildung 5: Die Erde aus 33 km Höhe

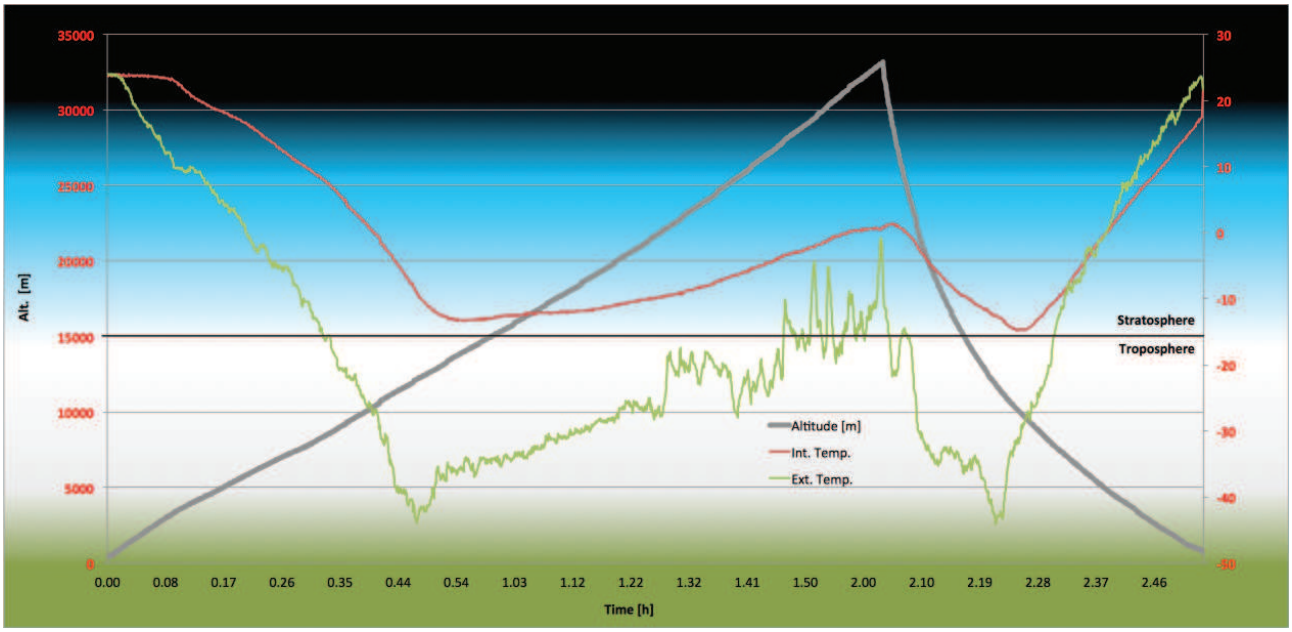


Abbildung 6: Die Messdaten im zeitlichen Verlauf

33 km Höhe weniger als 10 mbar beträgt, tritt eine Bremswirkung erst nach einiger Zeit ein.

Die Temperaturkurven zeigen einen interessanten Verlauf. Während der äussere Sensor jeweils sofort auf Änderungen anspricht, reagiert der innere Sensor stark verzögert. Das zeigt die hervorragende Isolationsfähigkeit der Styropor-Box. Ebenso interessant ist der Temperaturverlauf anhand der Höhe. Die minimale Umgebungstemperatur wird etwa auf 12'500 m Höhe erreicht, sie liegt dann bei ca. -45°C. In 33 km Höhe steigt sie dafür wieder auf Werte nahe dem Gefrierpunkt.

Die Luftdruckmessungen sind im Diagramm nicht angegeben, da die gemessenen Daten nicht brauchbar sind. Der Grund dafür liegt wohl darin, dass der verwendete Drucksensor für den Betrieb in grossen Höhen nicht geeignet ist.

Nächste Mission in Planung

Die Analyse der Daten und Bilder hat gezeigt, dass noch etliche Verbesserungen möglich sind. Das Wetter war am Flugtag zwar schön, aber etwas zu dunstig. Dadurch sind nur die Aufnahmen während der Startphase und diejenigen bei maximaler Flughöhe brauchbar. Zudem ermüden die starken Rotationen während des Aufstiegs den Betrachter des Videos. Die Konstruktion der Kapsel sollte deshalb verbessert werden, so dass sie im Wind nicht mehr so stark rotiert, wobei eine leichte Rotation nicht unerwünscht ist, weil dadurch eine Rundumsicht erreicht wird. Wie bereits erwähnt, hat der Drucksensor nicht wie gewünscht funktioniert und muss darum ersetzt werden.

Aus diesen Gründen ist bereits eine weitere Mission in Planung, in der zusätzliche Werte wie Luftdruck und Beschleunigungskräfte gemessen werden sollen. Das Kamera-Equipment soll zudem um eine weitere *GoPro*-Kamera ergänzt werden, welche für hochauflösende Fotos der Erdoberfläche sorgen wird.

Referenzen

[CUSF] CUSF Landing Predictor: http://habhub.org/predict/

[FEZP] GHI Electronics: http://www.ghielectronics.com/catalog/product/256

[GMAP] GMap.NET: http://greatmaps.codeplex.com/

[M3Sp] M3 Space Website: http://www.m3space.imvs.ch

[XBEE] Digi Corp.: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-868

Ist sicheres Cloud Computing möglich?

Bezüglich der Nutzung cloudbasierter Dienste zur Kostenreduktion und Verbesserung der Verfügbarkeit haben viele IT-Verantwortliche Sicherheitsbedenken. Dabei erlauben bestimmte technische und organisatorische Massnahmen eine hinreichend sichere Nutzung von Cloud-Diensten.

Hannes Lubich | hannes.lubich@fhnw.ch

Die Auslagerung der Erbringung von ICT-Diensten ist im Zuge der Globalisierung und des stetigen Kostensenkungs-, Optimierungs- und Innovationsdrucks weit fortgeschritten. Während bei bisherigen Outsourcing-Modellen jedoch eher die Übergabe oder die Migration bereits vorhandener Entwicklungs-, Betriebs- und Wartungsleistungen inklusive Personal und ICT-Infrastrukturen an externe Diensterbringer im Zentrum des Servicemodells standen, entsteht durch das Angebot von cloudbasierten Dienstleistungen ein neues Auslagerungsmodell. Dieses neue Modell basiert auf der Nutzung einer technologisch wesentlich stärker standardisierten, dafür jedoch weitgehend dynamischen, orts- und umgebungsunabhängigen Infrastruktur und Diensterbringung.

In diesem Auslagerungsszenario entstehen zwangsläufig neue Fragen nach der Definition, Erbringung und Überprüfbarkeit von Risikomodellen und entsprechenden Sicherheitsleistungen. Das wesentliche Ziel der Risikoanalyse und -bewirtschaftung ist es, das bislang erreichte Schutzniveau auch in einem Cloud-Dienstmodell, das die Grenzen der klassischen Informationssicherheit bezüglich der Kontrolle über die Sicherheitsanforderungen und deren Überwachung und Überprüfung deutlich überschreitet, beibehalten zu können.

Angebot und Nutzung cloudbasierter Dienste

Clouds sind definiert als meist parallele und geografisch breit verteilte Systeme, die aus einer Ansammlung miteinander vernetzter und oft virtualisierter Computersysteme bestehen. Diese Systeme werden dynamisch verwaltet und zugeteilt, erscheinen den Benutzern gegenüber jedoch als einheitlicher Service, dessen Leistungsfähigkeit dynamisch den Benutzeranforderungen angepasst werden kann. Das Nutzungsmodell basiert auf zuvor formell ausgehandelten Service Level Agreements gemäss der vorhandenen Serviceklassen und der zugehörigen, meist nutzungsabhängigen Tarifierung und Abrechnung.

Cloudbasierten Diensten liegt also die Annahme zugrunde, dass die ICT-Infrastruktur – Netzwerke und deren Komponenten, Server und deren Basisdienste (Speicher, Rechenleistung) –, aber ggf. auch standardisierte Dienstleistungen wie E-Mail, webbasierte Anwendungen inklusive der nötigen Datenbewirtschaftung usw. innerhalb einer für den Kunden nicht differenzierbaren „Wolke" gemäss einem definierten Service Level angeboten werden. Der Wolke hinzugefügt werden dann meist weitere Basisleistungen wie Benutzeridentifikation, -authentisierung und -autorisierung durch geeignete Zugriffsschutzmodelle, einfache Sicherheitsmechanismen wie Firewalls und „Intrusion Detection"-Systeme sowie die Betriebsüberwachung und Alarmierung bzw. Eskalation im Fall erkannter Störungen.

Das Geschäftsmodell eines solchen Cloud-Angebots basiert also einerseits auf einer sehr starken „economy of scale" mit möglichst vielen Nutzern, welche die Aufbau-, Betriebs-, Erweiterungs- und Erneuerungskosten der Cloud finanzieren, und andererseits auf der starken Standardisierung der Dienste in der Cloud, um die Komplexität der Cloud zu beschränken und damit entsprechende Risiken zu minimieren (im Gegensatz zur Übernahme von „Legacy"-Systemen in vielen klassischen Outsourcing-Ansätzen). Diese Standardisierung erstreckt sich dabei von den applikatorisch verfügbaren Umgebungen (meist webbasierte Services) über die Datenhaltung in standardisierten Datenbankumgebungen bis hin zu standardisierten (ggf. virtualisierten) Servern mit vorkonfigurierten Betriebssystemen. Eine Migration „in die Cloud" erfordert demzufolge vom Dienstnutzer das vorgängige „Aufräumen" der eigenen Infrastrukturen und Anwendungen, um „cloudkonform" zu werden.

Cloud-Dienste werden gemäss einer Hierarchie des Dienstangebots klassifiziert:

1. *Infrastructure as a Service (IaaS):* Die Infrastrukturanbieter stellen eine grosse Menge von ICT-Ressourcen zur Verfügung (Sekundärspeicher, Rechenleistung usw.). Durch Virtualisierung können diese Ressourcen dynamisch den Nutzern zugeordnet werden und bieten dadurch die Fähigkeit, zeitnah (und ggf.

kostengünstig, je nach Abrechnung und Tarifierungsmodell, z. B. Sockelbeitrag plus „Pay per Use“) die jeweiligen Benutzerbedürfnisse abzudecken. Gleichzeitig erlaubt dies dem Anbieter auch – analog zur häufigen Praxis von Fluggesellschaften – ein Überbuchen der verfügbaren Ressourcen bzw. die Befriedigung von Benutzerbedürfnissen, deren Summe die Kapazität der Ressourcen eigentlich übersteigt. Jedoch muss der Benutzer „seinen“ Software-Stack selbst erstellen, ausrollen, verwalten und betreiben. Ein typisches Beispiel ist der EC2 Cloud Service von Amazon [1], der die vorhandene Kapazität bzw. Überkapazität der Amazon-eigenen ICT-Infrastruktur als mietbaren Service im offenen Markt platziert.

2. *Platform as a Service (PaaS):* Anstelle einer virtualisierten Infrastruktur wird auch die Softwareplattform (Betriebssystem und zugehörige Middleware-Komponenten) als Service zur Verfügung gestellt. Die darunter liegende optimale Zuteilung der Hardware und sonstigen Betriebsmittel geschieht dabei „unsichtbar“ für den Benutzer. Ein typisches Beispiel für diese Dienstklasse ist die „Google App Engine“ [2], die es Anwendern erlaubt, eigene Webanwendungen in der Infrastruktur von Google auszuführen.
3. *Software as a Service (SaaS):* In diesem Szenario wird auch die Anwendungssoftware als verwalteter und abrechenbarer Dienst zur Verfügung gestellt. Die darunter liegende Plattform und technische Infrastruktur und deren Zuteilung bleiben dabei weiterhin vor dem Benutzer verborgen. Typische Beispiele für diese Serviceklasse sind die „Oracle Platform for SaaS“ [3] oder „salesforce.com“ [4] zur Auslagerung von CRM-orientierten Anwendungen. Einige dieser Plattformen erlauben durch die Bereitstellung entsprechender Entwicklungsumgebungen zusätzlich die Erstellung und Nutzung neuer Softwarekomponenten durch den Anwender in der Cloud.

Eine ähnliche Taxonomie der Yankee Group [5] stellt diesen drei Cloud-Modellen noch ein weiteres Modell voran, in dem einfache Cloud-Services über das Internet bezogen werden (siehe Abb. 1).

Erste Cloud-Ansätze entstanden typischerweise im Umfeld von sehr grossen Technologienutzern und Serviceanbietern, die an einer Zusatzfinanzierung ihrer internen Über- oder Spitzenlastkapazität durch das Angebot einfacher cloudbasierter Dienste (typischerweise Ablage von Dateien, E-Mails oder ähnliche Dienste) interessiert waren. Durch den Sprung zu „Software as a Service“ und die Bereitstellung meist webbasierter Entwicklungs- und Betriebsumgebungen in der Cloud können nun jedoch Geschäftsmodelle für das Angebot komplexer Applikationen entwickelt und umgesetzt werden, die den kommerziellen Aufbau und Betrieb von Cloud-Diensten ohne Querfinanzierung erlauben. Der Marktforscher IDC schätzt, dass Cloud-Services ein stark wachsendes Marktpotenzial haben, da Firmen durch die Nutzung von Clouds ihre Infrastruktur- und Betriebskosten sowie die Kosten für die Sicherung, Pflege, Modernisierung, Leistungssteigerung usw. einsparen bzw. von einem Fixkostenmodell auf ein „Pay per Use“-Modell umstellen können. IDC erwartet weltweit eine Steigerung des Umsatzes von ca. 121.5 Mrd. US-Dollar im Jahr 20010 auf etwa 72 Mrd. US-Dollar im Jahr 2015 [6]. Zudem wird angenommen,

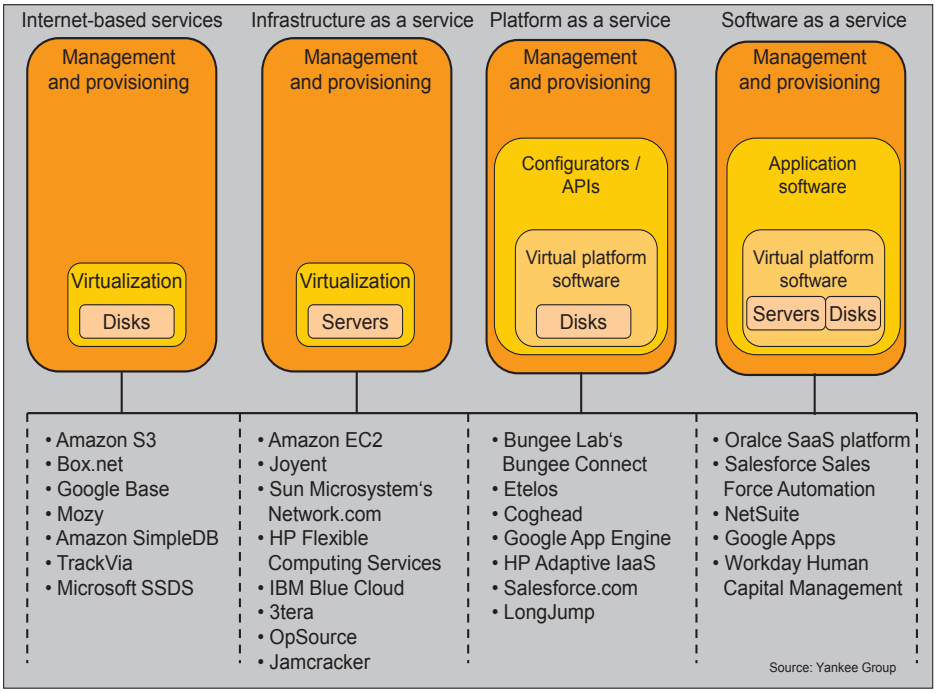


Abb. 1: Cloud-Taxonomie der Yankee Group

dass im Jahr 2012 von den Kosten für die Informatik eines Unternehmens etwa 25% auf die Nutzung von Cloud-Services entfallen werden [7].

Wesentliche Treiber für das Angebot und die Nutzung von Cloud-Diensten sind also die Reduktion von Komplexität, Fixkosten und Infrastrukturrisiken für den Kunden bzw. die Nutzung von Skaleneffekten homogenisierter Infrastrukturen und Dienste für den Anbieter. Mögliche bremsende Faktoren für den Transfer von ICT-Systemen und -Services in die Cloud sind:

- die Abhängigkeit von Fremdanbietern, deren Besitzverhältnisse und finanziellen Hintergründe oft nur ungenau überprüfbar sind oder sich stark verändern können;
- die mangelnde Kontrolle über den Ort der Diensterbringung (z. B. in Ländern mit nicht hinreichendem Datenschutz oder pauschalen Regelungen bezüglich des staatlichen Zugriffs auf Kundendaten);
- die Kontrolle der Einhaltung von Dienst- und Qualitätsgarantien;
- die Aufrechterhaltung der IT-Sicherheit und der nötigen Rechtssicherheit (z. B. bezüglich der Einhaltung des Datenschutzes oder des Geschäftsgeheimnisses);
- die Verfügbarkeit, Qualität und Bezahlbarkeit der nötigen Kapazität;
- die ggf. mangelnde Transparenz der Leistungserbringung und kundenspezifischen Dienstabrechnung.

Auswirkungen auf Informationssicherheit und Risikomanagement

Viele etablierte Elemente der Informationssicherheit und des ICT-Risikomanagements bei der Auslagerung von Diensten basieren auf der Annahme, dass die relevanten Betriebs- und Kontrollparameter bzw. deren Governance in der Kontrolle des Auftraggebers verbleiben, während die Umsetzung auf dem Einsatz von Fremdleistungen basieren kann.

War die Einhaltung der Vorgaben für Informationssicherheit und Datenschutz bereits in traditionellen Auslagerungsmodellen komplex und mit substanziellem Aufwand verbunden, so bietet die Kontrolle der Einhaltung entsprechender Vorgaben in cloudbasierten Modellen zusätzliche Schwierigkeiten. Diese Aspekte müssen zwingend in die Gesamtrisikobetrachtung bei der Nutzung von Cloud-Diensten prominent einfliessen.

Im Folgenden werden typische Problemfelder exemplarisch und ohne Anspruch auf fallspezifische Vollständigkeit diskutiert.

1. Es gibt keine Kontrolle über den Ort der Diensterbringung (inkl. Transitorte und Netze) und deren Sicherheitsdispositive (von physischem Schutz und Zugangskontrollen bis hin zu den jeweils geltenden betrieblichen IT-Sicherheitskonzepten, Zertifikaten etc.). Eine eigentliche „Due Diligence“ pro betrieblichem Standort und Land ist in einer breit verteilten, dynamischen Cloud also nicht mehr möglich.
2. Die Frage der dynamisch grenzüberschreitenden Funktionalität gegenüber einer immer noch stark nationalen oder regionalen Gesetzgebung hat direkte Konsequenzen auf die Selektion eines Cloud-Angebots. In der Schweiz sind z. B. im Gegensatz zu anderen Ländern nicht nur Personen-, sondern auch Unternehmensdaten geschützt. Auch E-Mails sind in diesem Sinne Personendaten, die dem Datenschutz unterstehen – eine Bekanntgabe ins Ausland kann nur dann erfolgen, wenn am Speicher- oder Aufbewahrungsort ein äquivalentes Datenschutzrecht gilt. Bei einer jederzeit ortsveränderlichen Diensterbringung ist diese Überprüfung aufwendig, wenn nicht unmöglich.
3. Die Kontrolle über das in einen Cloud-Service eingebrachte geistige Eigentum (z. B. Kundendaten, Herstellungs- oder Berechnungsverfahren etc.) erfordert in gemeinsam von mehreren Kunden genutzten Plattformen und Applikationen besondere Aufmerksamkeit. Insbesondere müssen die Betriebsprozesse in der Cloud gewährleisten, dass keine Verwechslung, keine Durchmischung und kein ungeplanter Abgleich von Daten (z. B. in nicht strikt mandantenfähig ausgelegten Applikationen) erfolgt.
4. Eine auf allgemeine Nutzung ausgelegte Cloud wird in aller Regel nur allgemein definierte und implementierte Sicherheitseinrichtungen aufweisen – die zugehörigen Service Agreements sind im gleichen Sinne stark standardisiert und decken die Bereiche Informationssicherheit, Risikomanagement, Betrieb im Krisenfall etc. nur in sehr generischer Form ab. Dementsprechend entstehen für spezifische zusätzliche Sicherheitsanforderungen hohe Zusatzkosten, die in der Regel nicht auf alle anderen Nutzer der Cloud umgelegt werden können, sofern der Cloud-Betreiber überhaupt zur Implementation zusätzlicher, kundenspezifischer Sicherheitsmerkmale bereit ist.
5. Die Überwachung des Zustands der Informationssicherheit und der operationellen Risiken in kundeneigenen Security-Information-Management-Umgebungen ist meist nicht Bestandteil der Dienstleistung. Entsprechende Reportingschnittstellen müssen daher spezifisch definiert und bewirtschaftet werden, sofern die der Cloud unterliegende Infrastruktur diese Daten mandantenfähig und kundenspezifisch trennen und aufbereiten bzw. liefern kann. Das bisher eher feinkörnige Sicherheits- und Risikomanagement, basierend auf kundenspezifischen „Key Performance Indicators“, steht in dieser Form als Führungs-

und Entscheidungsunterstützungswerkzeug nicht mehr zur Verfügung.

6. Die Abhängigkeitskette bezüglich der End-zu-End-Verfügbarkeit wird nicht nur länger, sondern für den Kunden durch die starke Dynamik der Diensterbringung (Virtualisierung, Ortsveränderlichkeit usw.) auch intransparenter. Dies muss insbesondere in der Szenarienplanung für die Betriebsweiterführung im Not- und Krisenfall berücksichtigt werden. Cloud-Services sind jedoch durch ihre Ausrichtung auf ein standardisiertes Dienstangebot und standardisierte Technologiekomponenten weniger komplex als die klassischen Outsourcing-Modelle inklusive des Betriebs bestehender „Legacy"-Umgebungen. Daher ist es denkbar, dass sich diese beiden Effekte bezüglich Gesamtrisiko und Aufwand beim Kunden gegenseitig neutralisieren.
7. Grosse kommerzielle Cloud-Services sind ein Primärziel für Angreifer, wobei die Palette der Motivationen von der Erpressung des Serviceanbieters durch „Denial of Service"-Angriffe bis hin zum „Mitlesen" und zum Datendiebstahl, ggf. schon in der Grauzone der Nachrichtendienste (Terrorismusbekämpfung, Geldwäscherei, Stärkung des eigenen Wirtschaftsstandorts etc.) oder der informationellen Kriegsführung reicht. Ein zusätzliches Problem entsteht durch den grossen Kreis der Betroffenen: In einer zwischen allen Kunden gemeinsam genutzten Infrastruktur und Applikationslandschaft leiden alle Kunden unter einem Angriff, auch wenn nur ein einzelner Kunde angegriffen werden sollte.

Katalog angemessener Sicherheitsmassnahmen

Im Spannungsfeld zwischen Kosten- und Effizienzdruck einerseits und den dargelegten Sicherheitsüberlegungen andererseits kann die Frage nicht lauten, ob man Cloud-Services verwendet oder nicht, sondern unter welchen Bedingungen und mit welchen flankierenden Massnahmen eine Nutzung von Cloud-Diensten möglich und ökonomisch sinnvoll ist.

Ein möglicher Startpunkt für diese Abklärung aus Sicht der IT kann die „Security Guidance for Critical Areas of Cloud Computing" der „Cloud Security Alliance" [8] sein, die seit November 2011 in der aktualisierten Fassung 3.0 vorliegt. Dieses Dokument spezifiziert die Sicherheitsanforderungen an Cloud-Computing-Umgebungen anhand von 13 Arbeitsbereichen, aufgeteilt nach aufsichtsbezogenen (siehe Tabelle 1) und eher betrieblich orientierten Themen (siehe Tabelle 2).

Für jeden dieser Arbeitsbereiche werden in der „Security Guidance for Critical Areas of Cloud Computing" generische Empfehlungen bezüglich Umsetzung in cloudbasierten Umgebungen gemacht, welche die Grundlage für die Spezifikation von Kundenanforderungen bezüglich Informationssicherheit bilden können.

Eine breit abgestützte Abbildung auf die bislang dominierenden Standards und „Best Practices" für Informationssicherheit (insbesondere ISO2700x und CoBIT) steht jedoch noch aus, ebenso fehlen konkrete Nutzungs- und Umsetzungserfahrungen aus dem Betrieb, sodass potenzielle Kunden gemäss ihrem jeweiligen Risikoprofil selbst zu beurteilen und zu entscheiden haben, ob sie bezüglich cloudbasierter Dienste als „Early Adaptor" oder doch eher als „Late Follower" agieren wollen.

Zusammenfassung

Die zuvor aufgeführte Liste von Sicherheits- und Risikomanagementaspekten kann zu dem Schluss führen, dass die Nutzung cloudbasierter Angebote generell nicht angezeigt ist. Dieser Haltung stehen jedoch die möglichen Einsparungen und Skaleneffekte gegenüber, die in einer Risiko-Gewinn- und Verlustrechnung zu berücksichtigen

Aufsichtsthema	Inhalt
Governance and Enterprise Risk Management	Fähigkeit eines Unternehmens, das Risiko durch die Nutzung von cloudbasierten Diensten zu beurteilen und zu steuern bzw. die Risiken und die nötigen Gegenmassnahmen in das unternehmensweite Risikomanagement einzubetten.
Legal Issues, Contracts and Electronic Discovery	Feststellung der rechtlichen und regulatorischen Rahmenbedingungen für die Nutzung cloudbasierter Dienste. Diesem Punkt kommt insbesondere bei der Nutzung grenz- und rechtsüberschreitender Angebote eine besondere Bedeutung zu (allgemeiner Datenschutz, Offenlegung von Daten gegenüber Behörden, Behandlung besonders schützenswerter Daten, Melde- und Berichtspflichten etc., aber auch Überprüfung und Management der Nutzungsbedingungen der Cloud).
Compliance and Audit Management	Regelmässige Überprüfung des ordnungsgemässen Betriebs der cloudbasierten Dienste und Aufbewahrung bzw. Rapportierung ausreichender Beweismittel aus der Überprüfung in Koordination mit den entsprechenden internen und externen Aufsichtsstellen; zudem Festlegung nötiger Korrekturmassnahmen und Überwachung der Umsetzung.
Information Management and Data Security	Management von Daten in der Cloud, Festlegung von Schutzbedarf, Rollen, Rechten und Verantwortlichkeit bezüglich dieser Daten (Data Ownership) und Umsetzung von Kontrollen zum Schutz der Daten vor nicht zulässiger Weitergabe, Nutzung, Verfälschung oder Löschung.
Interoperability and Portability	Aufrechterhaltung der Fähigkeit, Daten und Dienste von einem Dienstanbieter zu einem anderen zu verlagern (Interoperabilität) und Daten/Dienste bei Bedarf wieder in die eigene IT zu integrieren.

Tabelle 1: Die aufsichtsbezogenen Themen der „Security Guidance for Critical Areas of Cloud Computing"

Betriebliches Thema	Inhalt
Traditional Security, Business Continuity and Disaster Recovery	Beurteilung und Management des Einflusses der Nutzung cloudbasierter Dienste auf die vorhandenen Prozesse für Security Management und die Geschäftsweiterführung im Krisenfall. In diesem Bereich ist insbesondere zu beachten, dass die in der Cloud genutzten Dienste und Infrastrukturen nicht mehr dem eigenen Zugriff und der eigenen Kontrolle unterstehen. Die entsprechenden Prozesse des Dienstnehmers müssen also sehr genau spezifizierte Schnittstellen zu den entsprechenden Diensten und Prozessen des Dienstanbieters aufweisen und bewirtschaften.
Data Center Operations	Beurteilung der Architektur des Rechenzentrums des Dienstanbieters sowie der entsprechenden betrieblichen Prozesse. Dieser Aspekt ist insbesondere bei der Auswahl von Dienstanbietern, bei der Umgestaltung des Betriebs durch den Anbieter während der Vertragslaufzeit oder bei beobachteten betrieblichen Mängeln von Interesse.
Incident Response, Notification and Remediation	Angemessen zeitnahe und vollständige Erkennung von Störungen im Betrieb der cloudbasierten Dienste, stufengerechte Information der Dienstnutzer und rasche Störungsbehebung. Dieser Aspekt umfasst technische Schnittstellen (z. B. zwischen Helpdesks oder Trouble-Ticket-Systemen) sowie organisatorische Anpassungen (Informations- und Eskalationspfade, gemeinschaftliches Problemmanagement, nachgelagerte Ursachenanalyse, Forensik usw.).
Application Security	Sicherungsmassnahmen für Anwendungssoftware, die in einer Cloud entwickelt und/oder betrieben wird. Dieser Aspekt beinhaltet Fragen, ob eine Anwendung für die Cloud-Nutzung umgestaltet oder in die Cloud migriert werden kann und welche Plattform (SaaS, PaaS oder IaaS) dafür geeignet ist.
Encryption and Key Management	Identifikation und Gestaltung ausreichend starker Chiffrierung und ausreichend skalierbarer Schlüsselverwaltung in cloudbasierten Umgebungen mit verteilter betrieblicher Verantwortung. In diesem Bereich müssen auch Themen wie die Einbettung in oder bewusste Trennung von Verschlüsselungsinfrastrukturen zwischen internem Betrieb und der Cloud-Umgebung sowie die treuhänderische Hinterlegung von Schlüsselmaterial zwischen den Vertragspartnern diskutiert und bearbeitet werden.
Identity, Entitlement, and Access Management	Management von Identitäten, Rechten und Rollen, basierend auf Benutzerverwaltungs- und Verzeichnisdiensten, die entweder spezifisch für die Cloud-Umgebung aufgebaut und betrieben oder mittels geeigneter Schnittstellen an das interne Identitätsmanagement angeschlossen und durch entsprechende Bearbeitungsprozesse betrieben werden. In diesen Bereich fallen zudem Anlage und Auswertung von Protokollinformationen bezüglich Vergabe, Nutzung und Modifikation von Zugriffsrechten, um Missbräuche oder Fehlkonfigurationen rasch erkennen und beseitigen zu können.
Virtualization	Beurteilung und Kontrolle der Nutzung von hard- oder softwarebasierten Virtualisierungstechnologien in der Cloud-Umgebung bezüglich Aufrechterhaltung der Sicherheitsvorgaben (z. B. bei unkontrollierter Redundanz von Datenhaltungen oder paralleler Nutzung von Infrastrukturen durch unterschiedliche Kunden).
Security as a Service	Angebot von Sicherheitsüberprüfungen, Fallbehandlung und/oder Betriebsüberwachung sicherheitsrelevanter Einrichtungen durch spezialisierte Drittanbieter (ggf. mit der Delegation von Handlungsrechten im Schadenfall).

Tabelle 2: Die betrieblichen Themen der „Security Guidance for Critical Areas of Cloud Computing“

sind. Bei sorgfältiger Vorbereitung und Durchführung entsprechender Migrationsprojekte „in die Cloud“ und bei angemessener Berücksichtigung der „Security Guidance for Critical Areas of Cloud Computing“ ist ein ausreichend sicherer Betrieb basierend auf Cloud-Angeboten jedoch durchaus realistisch.

Neben den technischen Abklärungen und Transitionen müssen insbesondere die rechtlichen und regulatorischen Rahmenbedingungen und die korrekte Ausübung von Governance- und Compliance-Vorgaben bei cloudbasierten Services sehr sorgfältig und unter Einbeziehung aller betroffenen Instanzen vor einem Servicebezug abgeklärt werden. Zudem sind die Überprüfungen des gewählten Dienstanbieters im Betrieb zyklisch zu wiederholen und mit den vereinbarten Dienstgüteparametern und vertraglichen Vereinbarungen zu vergleichen, um eine „schleichende“ Dienstgefährdung zu vermeiden.

Referenzen

[1] Amazon Elastic Compute Cloud (Amazon EC2): http://aws.amazon.com/de/ec2/

[2] Google App Engine: https://developers.google.com/appengine/

[3] Oracle SaaS Platform: Building On-Demand Applications – An Oracle White Paper, Sep 2008: http://www.oracle.com/us/technologies/cloud/026989.pdf

[4] http://www.salesforce.com

[5] Yankee Group Cloud Comuting Survey, July 2011: http://waimingmok.files.wordpress.com/2009/01/yankeegroupcloudservices.jpg

[6] IDC Cloud Research: http://www.idc.com/prodserv/idc_cloud.jsp

[7] Cap Gemini Studie IT-Trends 2012: http://www.ch.capgemini.com/insights/publikationen/it-trends-2012/

[8] Cloud Security Alliance: Security Guidance for Critical Areas of Focus in Cloud Computing Version 3.0: https://cloudsecurityalliance.org/research/security-guidance/

Spielend Fremdwörter büffeln mit DashInEnglish

Moderne Smartphones und Tablets haben auf die heutige Jugend eine sehr grosse Anziehungskraft, die für spielerisches Lernen genutzt werden kann. Anstatt englische Vokabeln mithilfe von Karteikärtchen oder Wörterbüchern zu büffeln, lässt sich das Pflichtvokabular auf einem mobilen Gerät spielerisch erwerben. Die Android-App DashInEnglish basiert auf dem unkonventionellen Dasher-Schreibsystem, welches zur Texteingabe auf einem berührungsempfindlichen Bildschirm dient. Dabei fährt die Benutzerin mit einem Finger mit einer sanften Bewegung von einem Buchstaben zum nächsten und erhält dabei laufend Rückmeldung über die Korrektheit der angesteuerten Buchstaben.

Zdena Koukolíková, Carlo U. Nicola | carlo.nicola@fhnw.ch

Dasher wurde ursprünglich in der Inference-Gruppe von David MacKay an der Universität Cambridge entwickelt [WBM00] und von uns für die Android-Plattform in einer veränderten Form in Java von der Pike auf neu geschrieben [GKN12a/b]. Das Programm stellt ein Texteingabe-Interface zur Verfügung, das mit natürlichen, kontinuierlichen Zeigegesten geführt wird. Das Schreiben fühlt sich an, als ob man mit dem Finger einen virtuellen Wagen auf dem Bildschirm steuern würde; die Benutzerin fährt stets in die Richtung des Buchstabens, den sie ihrem Text als nächsten hinzufügen möchte (siehe Abb. 2). Dieses spielerische Schreiben dürfte Kinder und Jugendliche anregen, das wenig geliebte, aber durchaus notwendige Fremdwörterbüffeln mit mehr Spass in Angriff zu nehmen.

Im vorliegenden Beitrag stellen wir unsere Android-App *DashInEnglish* vor, welche die Grundzüge einer spielerischen Lernhilfe für Englischvokabeln illustriert. Diese App ist sowohl für SmartPhones als auch Tablets konzipiert. Um beide Endgerättypen gleichermassen gut zu unterstützen, sind ein paar Vorkehrungen bei der Implementierung zu treffen, die wir in diesem Artikel genauer vorstellen werden.

Aufbau des GUIs

Auf einem Nexus 7 Tablet sieht die grafische Benutzungsoberfläche von *DashInEnglish* wie in Abbildung 1 (Anfangsposition) oder in Abbildung 2 aus. Im linken Teil der Anwendung lässt sich aus einer Liste der verschiedenen Lektionen die gewünschte auswählen, die dann rechts auf dem Bildschirm erscheint. Im oberen Textfeld wird jeweils das zu übersetzende deutsche Wort angezeigt und im darunterliegenden Textfeld werden die bis dahin geschriebenen Buchstaben der englischen Übersetzung dargestellt.

In Abbildung 2 ist die erste Lektion ausgewählt worden und die Benutzerin wird aufgefordert, eine englische Übersetzung („letterbox“) für das Wort „Briefkasten“ mit Hilfe des auf der rechten Seite angezeigten Dashers einzugeben. Zum aktuellen Zeitpunkt hat sie bereits den ersten richtigen ‚l‘ Buchstaben ausgewählt und der *Prediction Partial Matching* (PPM) Algorithmus die drei wahrscheinlichsten Nachfolgebuchstaben ‚a‘, ‚e‘ und ‚o‘ vorausberechnet. Quadrate unterschiedlicher Farben stellen verschiedene Buchstaben dar. Nun muss die Benutzerin mit ihrem Finger das Quadrat des nächsten Buchstabens in Richtung des Zentralkreuzes (siehe Abb. 1) lenken. Sobald das Zentralkreuz in einem Buchstabenquadrat liegt,

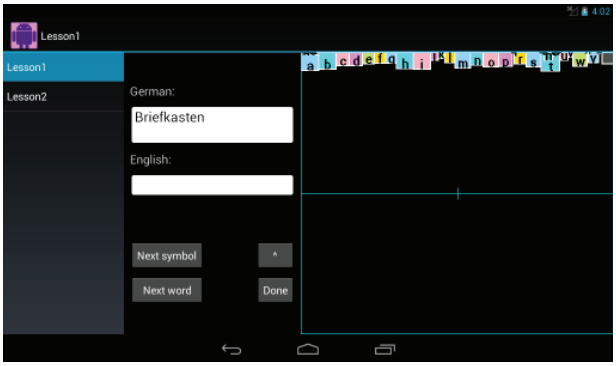


Abbildung 1: Dasher-Anfangsposition für ein neues Wort

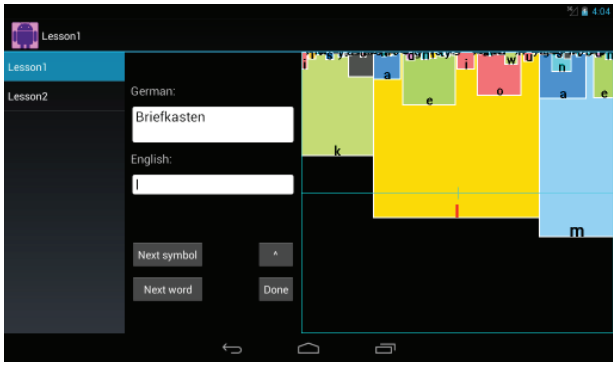


Abbildung 2: Eingabefläche von DashInEnglish auf einem Android Nexus 7 Tablet in horizontaler Lage

wird der entsprechende Buchstabe automatisch ins untere Textfeld übernommen.

Die vier Eingabeknöpfe unterhalb des Textfelds haben die folgenden Bedeutungen:

- „Next symbol“: bewegt Dasher zum nächsten richtigen Buchstaben;
- „Next word“: wählt zufällig ein neues Wort aus dem Vokabular der aktuellen Lektion;
- „^“ stellt von Klein- auf Grossbuchstaben um und umgekehrt;
- „Done“: verlässt die App.

Texteingabe

Wie zuvor erwähnt, erfolgt die Texteingabe des englischen Textes im rechten Teil der App. Mit einer Fingerberührung auf die obere Hälfte der Oberfläche beginnt Dasher sich in Richtung des Fingers vorwärts zu bewegen, wobei sich die angepeilten Buchstabenquadrate laufend vergrössern. Sobald ein Quadrat das Zentralkreuz bedeckt, wird der entsprechende Buchstabe in das Englischtextfeld übernommen. Wenn der Buchstabe korrekt ist, wird er innerhalb des Dashers farblich hervorgehoben. Um das positive Feedback zu erhöhen, ertönt gleichzeitig ein akustisches Feedback. Wenn der Buchstabe dagegen nicht richtig ist, kann zwar weitergeschrieben werden, die Buchstaben bleiben aber schwarz.

Falls man einen Tipp braucht oder sich total „verlaufen“ hat, kann man jederzeit den Knopf „Next symbol“ drücken. Dadurch erscheint der auf den schon geschriebenen Text nächstfolgende, korrekte Buchstabe sowohl in der Texteingabe als auch im Dasher. Von da an kann man wieder mittels Fingerbewegungen weiterschreiben oder wenn nötig den nächsten Buchstaben verlangen.

Wenn man einen falschen Buchstaben eingegeben hat, ist dies nicht weiter schlimm, da man sehr einfach einen anderen Pfad einschlagen kann. In der in Abbildung 2 dargestellten Situation könnte man z. B. den Dasher einfach in Richtung des Symbols ‚k‘ lenken und damit das bereits geschriebene ‚l‘ überschreiben. Dies geschieht sobald das ‚l‘-Quadrat das Zentralkreuz verlässt und das ‚k‘-Quadrat das Kreuz bedeckt. Wenn mehrere Buchstaben falsch sind, berührt man am besten die untere Hälfte der Dasher-Oberfläche und fährt den Dasher somit zurück. Sobald ein Quadrat das Zentralkreuz verlassen hat, wird der entsprechende Buchstabe gelöscht.

Das Navigieren im Dasher-Raum wird durch zusätzliche Gesten erleichtert. So lassen sich die Quadrate entweder nach links oder nach rechts bewegen, indem man rechts bzw. links auf der Mittellinie des Dashers drückt. Mit zwei Fingern ausgeführte Swipe-Bewegungen lassen den Dasher entweder ein- oder auszoomen. Nach einer sehr kurzen Einarbeitungszeit beherrscht man diese verschiedenen Optionen und das Navigieren auf dem Bildschirm wunderbar.

Die Eingabe eines Wortes wird als beendet betrachtet, wenn das drauffolgende Leerzeichen-Quadrat (ohne Symbol) das Zentralkreuz bedeckt. Sobald dies eintritt, ertönt das eingegebene Wort auf Englisch. Hierzu verwendet *DashInEnglish* die vom Android-System bereitgestellte, synthetische Sprache. Stattdessen liessen sich aber auch Aufnahmen einer menschlichen Stimme verwenden. Durch Betätigen des Knopfes „Next word“ kann das nächste Wort in Angriff genommen werden. Die Auswahl der Wörter geschieht zufällig, damit die Wörter unabhängig voneinander gelernt werden. Es wird jedoch garantiert, dass innerhalb einer Lektion alle gespeicherten Wörter behandelt werden.

Textvoraussage

Um das Schreiben zu vereinfachen und somit zu beschleunigen, benutzt Dasher ein prädiktives Sprachmodell der jeweiligen Sprache, das laufend den nächstfolgenden Buchstaben im gegebenen Textkontext vorschlägt. Man beachte, dass die Grösse der verschiedenen Quadrate, die den potenziellen nächstfolgenden Buchstaben darstellen, proportional zu der Wahrscheinlichkeit ist, mit welcher die entsprechenden Zeichen in der jeweiligen Sprache vorkommen.

Ward, Blackwell und MacKay [WBM00] haben die arithmetische Kodierung- und die statistische Modellierungsmethode, die ursprünglich zur Textkomprimierung entwickelt wurden, sehr elegant an die Berechnung der Wahrscheinlichkeit des nächsten im Text vorkommenden Buchstabens angepasst [AK]. Das hierfür benutzte Sprachmodell basiert auf dem *Prediction Partial Matching* (PPM) Algorithmus [CW84]. In diesem werden die nächsten Buchstaben anhand des vorangehenden Kontextes, d. h. der n vorherigen Buchstaben, vorgeschlagen. So ist in unserem Beispiel in Abbildung 2 die Wahrscheinlichkeit sehr gross, dass das nächste Zeichen entweder ein ’a’, ‘e’ oder ein ‘o’ sein wird, wenn man auf Englisch bereits das ‚l‘ geschrieben hat. Wir verwenden die vier letzten Buchstaben für die Vorhersage. Für Details zum angewendeten Sprachmodell verweisen wir auf [GKN12a/b].

Es ist offensichtlich, dass bei diesem Eingabesystem die ersten Buchstaben des gesuchten Wortes oft schwerer zu „erraten“ sind, als die letzten. Dies ist darauf zurückzuführen, dass die Wahrscheinlichkeitsberechnung für die ersten Buchstaben auf einem sehr kurzen Kontext beruht. Somit muss die Benutzerin anfangs gut überlegen, in welche Richtung sie Dasher navigieren soll. Schliesst das Wort hingegen mit einer in der jeweiligen Sprache gängigen Endung, so schlägt Dasher mit grosser Wahrscheinlichkeit die letzten Buchstaben von alleine korrekt vor. Aus Erfahrung kann man aber annehmen, dass zum Erlernen neuer Fremdwörter meistens die

ersten Buchstaben des jeweiligen Wortes wichtig sind; die letzten fallen einem dann oft wie von selbst ein.

Anpassung an die Bildschirmgrösse

Die Herausforderung dieses Projekts besteht darin, die als Android-Texteingabe-Service von uns bereits implementierte Dasher-Anwendung so zu adaptieren, dass *DashInEnglish* perfekt sowohl an Smartphones, als auch an die immer beliebteren Tablets angepasst ist.

Wir haben die Entwicklung mit Hilfe der neuesten Android Version 4.1 (API 16) durchgeführt [APIa]. Unsere App kann auf eine sehr einfache Art verändert werden, damit sie auch auf älteren Android-Versionen funktionsfähig ist [APIb].

Da die Bildschirme der SmartPhones und der Tablets unterschiedliche Grössen und Formfaktoren aufweisen, muss die Darstellung der Applikation an das jeweilige Gerät angepasst werden. Diese Flexibilität wird durch die in der Version 3.x eingeführten *Fragments* ermöglicht. Ein *Fragment* repräsentiert normalerweise einen wiederverwendbaren Teil eines *Activity User Interfaces*.

Wie aus den Abbildungen 1 und 2 ersichtlich ist, haben wir das Tablet-UI in zwei Fragmente aufgeteilt: das linke Fragment *(LessonsListFragment)* zeigt die Liste der verschiedenen Lektionen, das rechte *(DasherFragment)* zeigt das zu übersetzende deutsche Wort der entsprechenden Lektion und die Oberfläche der graphischen Text-Eingabe. Beide Klassen sind von *Fragment* abgeleitet. Je nach Bildschirmgrösse sind die zwei Fragmente entweder gleichzeitig oder aber nacheinander zu sehen. So wird auf einem Smartphone nur die Lektionsliste als Teil der Hauptaktivität *(DashInEnglishActivity)* gezeigt, das zweite Fragment erscheint aber erst nach Auswahl einer Lektion als Teil der neu aufgerufenen *DasherFragmentActivity* auf dem Bildschirm.

Da das Android-System erlaubt, nicht nur eigene Layouts für verschiedene Orientierungen, sondern auch für verschiedene Gerätgrössen zu entwerfen, wird der Entscheid, nur ein oder beide Fragmente gleichzeitig auf dem Bildschirm darzustellen, mit Hilfe der XML-Dateien *fragments_layout.xml* gefällt, die sich in den Layout-Resources Ordnern „res/layout-sw320dp“ (Listing 1) und „res/layout-sw600dp“ (Listing 2) befinden. Für kleine Bildschirme wird im ersten Layout nur das Liste-Fragment definiert, während für grosse Bildschirme beide Fragmente festgelegt werden.

In beiden Fällen wird automatisch das *LessonsListFragment* erstellt, wenn die Hauptaktivität *DashInEnglishActivity* ihr Layout in der Methode *onCreate()* mit Hilfe von *setContentView(R.layout.fragments_layout)* sozusagen aufbläht.

Nur im zweiten Fall aber wird das *FrameLayout* Dasher erstellt, das den Platz auf dem Bildschirm definiert und reserviert, worin das *DasherFragment* dargestellt wird. Wenn eine Lektion aus der Liste ausgewählt wird, wird im nachfolgenden Code, ermittelt, ob im aktuellen Layout

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <fragment
    class="ch.fhnw.android.dasher.english.LessonsListFragment"
    android:id="@+id/lessons"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
  />
</FrameLayout>
```

Listing 1: Layout für Smartphones

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <fragment
    class="ch.fhnw.android.dasher.english.LessonsListFragment"
    android:id="@+id/lessons"
    android:layout_weight="1"
    android:layout_width="0dip"
    android:layout_height="match_parent"
  />

  <FrameLayout android:id="@+id/dasher"
    android:layout_weight="4"
    android:layout_width="0dip"
    android:layout_height="match_parent"
  />
</LinearLayout>
```

Listing 2: Layout für Tablets

der Rahmen existiert, worin das *DasherFragment* dargestellt werden kann.

```
View dFrame = findViewById(R.id.dasher);
mDualPane = dFrame != null &&
dFrame.getVisibility() == View.VISIBLE;
```

Wenn kein *dFrame* existiert, wird wie gewohnt eine neue Aktivität *(DasherFragmentActivity)* aufgerufen, die die entsprechende Lektion samt der Dasher-Texteingabefläche darstellt. Auf einem grossen Bildschirm dagegen wird die Lektion rechts in Form eines Fragments abgebildet. In beiden Fällen verinnerlicht das *DasherFragment* die ganze Lerntätigkeit. Gleichzeitig wird der Name der gegenwärtigen Lektion in der oberen Leiste *(ActionBar)* angezeigt. Im Fall eines Tablets wird beim Anwählen einer neuen Lektion das aktuelle Fragment auf einem Stack (dem sogenannten *backstack*) zwischengelagert, damit es wiederhergestellt werden kann, wenn die Benutzerin auf die Back-Taste drückt.

Damit die App auf dem Tablet-Bildschirm immer korrekt dargestellt wird, auch dann wenn das Gerät während der Arbeit gedreht wird, muss der aktuelle Zustand in allen drei Hauptkomponenten in deren *onSaveInstanceState()* Methode gesichert werden. Dies ist notwendig, weil alle drei Komponenten beim Drehen zerstört werden. Der gegenwärtige Titel, d.h. der Name der dargestellten Lektion, wird im Bundle *outState* der Hauptaktivität *DashInEnglishActivity* gespeichert, und in *onCreate()* wieder in die *ActionBar* eingesetzt. Die beiden Fragmente müssen jeweils die Daten sichern, anhand derer ihr Zustand rekonstruiert werden kann. Im *LessonsListFragment* wird der Index der zuletzt ausgewählten Lektion in der Liste im Bundle abgelegt und in der Methode *onActivityCreated()* wieder als ausgewählt markiert, und die Hauptaktivität davon benachrichtigt. Das *DasherFragment* benötigt nicht nur die Eingaben aus den beiden Textfeldern, d.h. das aktuelle deutsche Wort und die Buchstaben, die bis dahin ins englische Textfeld geschrieben wurden, sondern auch zusätzliche Informationen,wie Name der Lektion, ihr Index in der Lektionsliste, Anzahl und Liste der bis dahin bearbeiteten deutschen Wörter und das korrekte englische Wort. Mit den gespeicherten Angaben wird das jeweilige Fragment in den Methoden *onCreate()* und *onActivityCreated()* von neuem erzeugt und rekonstruiert.

Auswählen einer Lektion

Der Lebenszyklus des *DasherFragments* muss auch darum sorgfältig behandelt werden, da wir die entsprechenden Fragmente beim Wechseln von einer Lektion zur anderen (in Form einer *FragmentTransaction*) auf den *Backstack* schieben, und diese mit Hilfe des Back-Tastendrucks wieder zum Leben erwecken wollen. Die so gestapelten Fragmente werden nicht völlig zerstört, sondern nur gestoppt: die zugehörige *View* wird aber dabei zerstört, da die Fragmente ja nicht mehr sichtbar sind. Deshalb müssen wir die Angaben, die zur Rekonstruktion der jeweiligen *View* notwendig sind, in der *onStop()* Methode des *DasherFragments* speichern. Es handelt sich hier lediglich um das zuletzt bearbeitete deutsche Wort und den bis dahin geschriebenen englischen Text, die als Instanzvariablen *mLastGermanWord* und *mLastEnglishWord* gesichert werden. Die Werte dieser Variablen werden dann in der Methode *onActivityCreated()* in die entsprechenden, neu erstellten Textfelder eingesetzt. Anhand dieser Angaben wird genau wie im oberen Fall der letzte Zustand der graphischen Texteingabefläche und der Dasher mit Hilfe der Methode *setDasherViewToInputText()* rekonstruiert, indem Dasher sozusagen bis zu dem zuletzt geschriebenem Buchstaben vorgerückt wird. Die restlichen Angaben sind in dem jeweiligen Fragment in Form weiterer Instanzvariablen gesichert.

Damit die beiden Fragmente möglichst unabhängig voneinander sind, wird die Kommunikation via die Hauptaktivität bewerkstelligt. Da das *LessonsListFragment* beim Auswählen einer Lektion die Hauptaktivität benachrichtigen muss, damit sie im *DasherFragment* dargestellt wird, definieren wir im *LessonsListFragment* das Callback-Interface OnLessonSelectedListener mit der einen Methode *onLessonSelected(String lesson, int index)*, welche dann in der *DashInEnglishActivity* implementiert wird.

In der Methode *onAttach()*, die aufgerufen wird, sobald das Fragment mit seiner Host-Aktivität verbunden worden ist, erhält *LessonsListFragment* eine Referenz auf eine Instanz der Interface-Implementierung (Listing 3).

```
@Override
public void onAttach(Activity activity) {
  super.onAttach(activity);
  try {
    mCallback = (OnLessonSelectedListener)activity;
  } catch (ClassCastException e) {
    throw new ClassCastException(activity.toString()
        + " must implement OnLessonSelectedListener");
  }
}
```

Listing 3: Registrierung der Hauptaktivität als Selection-Listener im LessonsListFragment

```
@Override
public void onListItemClick(ListView l, View v, int pos, long id) {
  String lesson = (String)l.getItemAtPosition(pos);
  // Hauptaktivität benachrichtigen: ein Element der Liste ausgewählt.
  mCallback.onLessonSelected(lesson, pos);
  mCurCheckPosition = pos;
}
```

Listing 4: Benachrichtigung der Hauptaktivität beim Auswählen einer Lektion

Beim Berühren eines Elements in der Lektionsliste wird somit die Aktivität mittels der Methode *onLessonSelected()* benachrichtigt, wie in Listing 4 gezeigt.

Die Hauptaktivität reagiert auf dieses Ereignis (siehe Listing 5), indem sie zuerst bestimmt, ob die Applikation auf einem Gerät mit grossem oder kleinem Bildschirm läuft, d.h. ob im Layout Platz für zwei oder nur ein Fragment reserviert ist. Im ersten Fall wird das *DasherFragment* auf dem Bildschirm dargestellt, sofern es nicht schon abgebildet ist, und der entsprechende Titel erscheint in der *ActionBar*-Leiste. Gleichzeitig wird die ganze Transaktion, d.h. hier das Ersetzen der Instanz des gegenwärtigen Fragments mit einer Instanz des neuen Fragments, auf dem *backstack* gespeichert, damit die vorherigen Fragmente später via Back-Taste erreichbar sind. Im zweiten Fall wird mittels eines *Intent* eine neue *DasherFragmentActivity* aufgerufen, die das entsprechende *DasherFragment* abbildet. Im *Intent* schicken wir die nötige Information, damit die richtige Lektion gezeigt wird.

Wenn die Benutzerin auf dem Nexus 7 Tablet die Back-Taste drückt, müssen die beiden Fragmente synchronisiert gezeigt werden, d.h. der Name der in der Liste ausgewählten Lektion muss mit dem Titel und mit der im *DasherFragment* tatsächlich abgebildeten Lektion übereinstimmen. Zu diesem Zweck haben wir uns des Interfaces *OnBackStackChangedListener* bedient, das in der Android Klasse *FragmentManager* zur Verfügung steht, und haben die abstrakte *onBackStackChanged()* Methode gemäss Listing 6 implementiert. Diese Methode wird aufgerufen, sobald der Inhalt des backstacks sich ändert und ist darum ideal, um den Titel in der *ActionBar*-Leiste zu aktualisieren und, wie in unserem Fall, die zwei Fragmente in Einklang zu bringen.

```
@Override
public void onLessonSelected(String lesson, int index) {
// Existiert ein Rahmen in der UI wohin das Dasher Fragment
// eingesetzt werden kann?
  View dView = findViewById(R.id.dasher);
  mDualPane = (dView != null && dView.getVisibility() == View.VISIBLE);
  if (mDualPane) {
    // Prüfen welches Fragment momentan gezeigt wird; ersetzen, wenn nötig.
    DasherFragment dF = (DasherFragment)
          getFragmentManager().findFragmentById(R.id.dasher);
    if (dF == null || !dF.getLessonName().equalsIgnoreCase(lesson)) {
      // Neues Fragment erstellen, das die Lektion zeigt.
      dF = DasherFragment.newInstance(lesson, dir.getAbsolutePath(), index);
      // Eine Transaktion ausführen: jegliches beliebige Fragment mit dem
      // neuen ersetzen.
      FragmentTransaction ft = getFragmentManager().beginTransaction();
      ft.replace(R.id.dasher, dF);
      // Das alte auf backstack schieben.
      ft.addToBackStack(null);
      ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
      ft.commit();
      // Titel setzen.
      setTitle(lesson);
      currTitle.setLength(0);
      currTitle.append(lesson);
    }
  } else {
    // Ansonsten neue Aktivität aufrufen, die die ausgewählte Lektion
    // in ihrem Dasher Fragment darstellt.
    Intent dI = new Intent(this, DasherFragmentActivity.class);
    dI.putExtra("lessonName", lesson);
    dI.putExtra("dir", dir.getAbsolutePath());
    dI.putExtra("lessonIndex", index);
    startActivity(dI);
  }
}
```

Listing 5: Reaktion der Hauptaktivität beim Auswählen einer Lektion

```
  getFragmentManager().addOnBackStackChangedListener(
     new FragmentManager.OnBackStackChangedListener() {
    @Override
    public void onBackStackChanged() {
      // Prüfen, welches Fragment momentan gezeigt wird.
      DasherFragment dF = (DasherFragment)
        getFragmentManager().findFragmentById(R.id.dasher);
      if (dF != null && dF.isVisible()) {
        setTitle(dF.getLessonName());
        currTitle.setLength(0);
        currTitle.append(dF.getLessonName());
        // Das List-Fragment benachrichtigen.
        LessonsListFragment lF = (LessonsListFragment)
          getFragmentManager().findFragmentById(R.id.lessons);

        if (lF.mCurCheckPosition != dF.getLessonIndex()) {
          // Das Element als ausgewählt markieren.
          lF.getListView().setItemChecked(dF.getLessonIndex(), true);
          lF.mCurCheckPosition = dF.getLessonIndex();
        }
      } else {
        // Alle Fragmente sind geschlossen.
        setTitle(appTitle);
        currTitle.setLength(0);
        currTitle.append(appTitle);
        // Das List-Fragment benachrichtigen.
        LessonsListFragment lF = (LessonsListFragment)
          getFragmentManager().findFragmentById(R.id.lessons);
        // Kein Element ausgewählt.
        lF.getListView().setItemChecked(-1, true);
        lF.mCurCheckPosition = -1;
      }
    }
  });
}
```

Listing 6: Synchronisation der beiden Fragmente und Aktualsierung des Titels

Zusammenfassung

Basierend auf einem sehr intuitiven und einfachen Texteingabe-Interface haben wir als Demonstration dessen Anwendungsmöglichkeiten eine Applikation entwickelt, die als Hilfsmittel zum Erlernen von Fremdsprachewörtern dienen und für jede beliebige Sprache angepasst werden könnte. Wir sind überzeugt, dass der Einsatz neuer Technologien die Motivation von Kindern und Jugendlichen erheblich steigern könnte, unbeliebte, aber doch sehr notwendige Aufgaben spielend zu bewältigen.

Referenzen

[AK] Arithmetische Kodierung und PPM: http://marknelson.us/1991/02/01/arithmetic-coding-statistical-modeling-data-compression/.

[APIa] Fragments: http://developer.android.com/guide/components/fragments.html

[APIb] Building a Dynamic UI with Fragments: http://developer.android.com/training/basics/fragments/index.html

[CW84] Cleary, J.G., Witten, I.H. Data Compression Using Adaptive Coding and Partial String Matching, IEEE Trans. on Comm., Vol. COM-32, No. 4, 1984.

[GKN12a] Gasser, M., Koukolíková, Z., Nicola, C.U. Texteingabe als Spiel, Android User, 01, 106-110, 2012.

[GKN12b] Gasser, M., Koukolíková, Z., Nicola, C.U. Texteingabe als Spiel – Teil 2, Android User, 03, 108-113, 2012.

[WBM00] Ward, D.J., Blackwell, A.F., MacKay, D.J.C. Dasher—a data entry interface using continuous gestures and language models, UIST '00 Proc. of the 13th annual ACM symposium on User interface software and technology, 2000.

Equivalence Testing Mobile Apps

Mobile apps are often developed and then evolved on more than one mobile operating system. For the publisher of such apps, the problem is how to ensure equivalence of the product on the various platforms, in the sense of acting equivalently with respect to a test set. In this paper we present an approach that tackles this problem from two directions: architecture and testing. First we will explain the role and pitfalls of reference architectures. Then we will present our equivalence testing framework. It is based on mocking components of the target implementation. Instead of implementing mock components for all platforms, our approach is to implement these mocks only once and run them on a central server. On the tested target device, stub components are injected that forward to their server-based counterparts. Thus the exactly same test code is applied to all platform specific implementations.

Christoph Denzler, Daniel Kröni, Maxim Moschko | christoph.denzler@fhnw.ch

Mobile phone technology is evolving rapidly. The market is highly competitive and market share figures are changing year by year. New mobile operating systems are still emerging, and established ones have to be supported in different versions. This fragmentation forces application developers to program for different platforms and devices in order to reach a large customer base.

The term 'platform' refers to a mobile operating system (like iOS, Android, Symbian, etc.) and their application programming interfaces (API). Adapting an application to different platforms (porting) confronts the programmer with a variety of abilities (camera resolution, transmission rates, multi-threading etc.), interfaces (wireless communications abilities, sensors and user interfaces), programming languages (Objective-C, Java, C, C#, JavaScript) and concepts (multi- versus single-tasking, reference counting versus garbage collection or user interaction).

We faced fragmentation when we were asked to support software development for a startup company. The application dealt with calling a security service in threatening situations, and it required access to a camera, a GPS, the phone and the network. As the application will be hopefully rarely used, it should provide a familiar user interface, i.e. the user interface should not distract the user from sending an alarm.

These requirements (and some others such as memory footprint) ruled out the two currently established possibilities to build cross platform applications (see [2]): Native cross-platform frameworks, such as PhoneGap [7], and mobile web application frameworks, such as Sencha Touch [8]. In our case the problem with mobile web applications was the limited access to device features (camera, phone). A native cross-platform framework was not an option because it introduces a dependency to the framework provider: only those platforms can be reached that are supported by it.

As listed in [1] there are many ways to cope with fragmentation. However manually implementing an application for different platforms remains a strong option. It is often the most pragmatic way to satisfy all of the above mentioned requirements and to deal with the different partners who implement the different ports and who are not able or willing to give up their development processes. We decided to leave as is the development of the application on the different platforms but surround it with our approach consisting of architecture and tests.

In this paper we will present a process and the tools to pragmatically support multi-platform development by testing equivalence of the specific implementations. Our approach embraces the differences between platforms but does not hide their individualities.

The paper is structured as follows: Section *Overview* provides an general view of our equivalence testing approach. In section *Abstract Architecture* we show how a common architecture can be defined and we hint at some pitfalls to avoid. Section *Equivalance Testing Internals* conceptually explains how our testing infrastructure works. Finally, section *Case Study*, using a simple example, illustrates our approach and shows how tests are written to verify the equivalence of the different implementations. Comments on related work, conclusions and future directions complete this paper.

Overview

The decision which mobile platforms will finally be supported depends on the availability of features vital to the application, e.g. the availability of certain sensors such as a GPS receiver or a camera. Regardless of the platforms chosen they should not influence the applications requirements. These are specified only once and describe what the application should do. Implementati-

on details should not be part of a requirements specification, be it a functional or non-functional requirement. For example an application that manages notes needs the possibility to persist them – regardless of the platform on which it is finally implemented. Typically such requirements do not change between platforms.

Our approach evolves from a platform-independent architecture to which all platform specific implementations must adhere. This abstract architecture defines a structural contract between components. It is then enhanced by a set of integration tests which check the behavior of the system.

Once device specific implementations have been developed and tested separately, there is no easy way to ensure that the different test sets will be consistent. Thus, it is not possible to check whether or not an implementation fulfills the common requirements. Moreover, changing requirements demand implementation adaptions and test code adaptions for each platform. This is time consuming, demotivating and above all, error prone.

We introduce equivalence tests which run against the abstract architecture without consideration of any concrete platform. They are written only once and are executed on the server. Remote method invocation is used to invoke functionality on the mobile device. Our approach does not deploy real mock components on the mobile device. Instead lightweight proxy objects are injected during test setup. They are responsible for the forwarding of any interaction to their server-based counterparts which are implementations of the mocking objects. Our method uses a variety of approaches described in [1]. At first, manually implement the application on different platforms. Then, write test and mock components which will be abstracted from the target platforms by a remote method forwarding scheme. The necessary proxy objects are generated for each platform.

Abstract Architectures (AA)

In order to test different platform-specific implementations against a common test set, it is essential to establish a common basis. This basis is provided in form of an abstract architecture.

An abstract architecture defines – as any software architecture should – the basic components of the software and their interactions. Software architecture specifies an application's modularization and defines interfaces between these components. According to the Quasar reference architecture [3] software components can be classified as either technical components (T-components), which depend on the target platform, or business components (A-components), which encapsulate business functionality. An AA specifies the behavior of the business components exclusively, thereby avoiding any assertions about technical implementations.

An AA is not a reference architecture in the sense of 'one size fits all'. It is much more a tailor-made architecture which is defined only once for each multi-platform project. It is a collaborative effort among specialists of each targeted platform.

When architecting an application which is amenable to equivalence testing, the following should be considered:

- Platform independence: the AA specifies the different components and their responsibilities in an abstract way that does not contain any assumptions about or dependencies to concrete target platforms.
- Practicability: the AA must be implementable on any target platform. This implies a profound knowledge of the features and limitations of all target platforms.
- Equivalence testing framework requirements: the AA must provide a mechanism to access and replace the involved business components at runtime to enable testing.

These criteria will be discussed below in more detail.

Platform independence

Platform independence inherently prohibits any use of platform specific parts. As a consequence the AA needs to be self-contained so that it defines the complete API of all the (A- and T-) components in the system. T-components are inherently not abstract (they refer to a concrete platform). They need to be modeled in the AA by providing a platform agnostic interface. If, for instance, a graphical user interface (GUI) component is responsible for user interaction, then its interface must be part of the AA, although the graphical, haptic or acoustic representation is not part of the AA. It explicitly adapts to the target platform's user experience.

These constraints arise out of necessity: the ultimate goal of equivalence testing is to implement a single set of equivalence tests so as to be able to test all implementations. Thus tests are only allowed to access functionality which will be available on all platforms. This common functionality is determined by the AA.

Our testing infrastructure offers predefined abstract base types like integer, float and string and includes a default mapping to the obvious counterparts of the most popular mobile platforms. Every other data type must be defined in the AA. An ordered collection, such as a list, for example, needs to be modeled explicitly in the AA. Furthermore every single platform provides a corresponding type in its libraries. This is again essential to platform agnostic testing. For example, a test cannot and must not know that on one plat-

form a method is called 'length' and on another 'size'. To abstract from such naming clashes, it is common to provide a data wrapper type which implements a neutral interface and wraps the corresponding type of the target library.

Practicability

The second important requirement of a valid AA is practicability on each target platform. It must be possible to implement the specified interfaces on every selected platform without any deviations. If this is not possible, a workaround has to be found to elude the platform's limitation. Let's assume an application should be implemented for two different platforms from which one supports function overloading and the other does not. Under these circumstances the AA must not contain function overloading and would instead use distinct function names to bypass this constraint. Not all platform differences are smoothed away that easily. There are other platform specific constraints which impose much stronger restrictions on an AA.

The selection of the target platforms determines what is allowed or disallowed in an AA. To describe this more precisely we interpret the platform and its programming language as a set of capabilities. Some examples for such capabilities are inheritance, function overloading or multi-threading. The set of available capabilities which can be utilized to specify the AA is the intersection of all platform capability sets. On the other hand, all the capabilities that are not supported by all the platforms should be avoided in the AA except if there is a feasible workaround. For example, let us compare JavaScript and C# in terms of inheritance. At a first glance JavaScript does not support inheritance. However it is possible to emulate inheritance capabilities in JavaScript. If the effort to emulate inheritance on JavaScript is considered reasonable, this feature can be used in the AA. As a consequence the set of available capabilities depends heavily on target platforms. If too many capabilities are missing on a platform, its porting should then be scrutinized.

Equivalence-testing framework requirements

To be amenable to be used in our equivalence testing framework, each AA needs to offer a component registry which grants access to individual components at runtime. There are two important rules that any implementation of an AA must follow:

1. the system's components may only interact with the ones that have been retrieved from the component registry and
2. these components must communicate only through the interfaces specified by the AA.

This enables the testing framework to remotely invoke methods on components and/or replace them by mocks. A more detailed description of this is in the next section.

Scope of the abstract architecture

One reason to develop a native application is to ensure the best user experience on its target platform. GUI frameworks and user interaction concepts are usually very closely coupled to their platform. The usage of only a common (sub)set of user interface widgets that are available on all supported platforms leads to an 'unnatural' user experience or irritating GUIs. Therefore, the AA usually does not specify any user interface aspects.

Similar considerations should be applied to components that are tied closely to a platform or proprietary hardware features. It does not make sense to try to equivalence-test non-equivalent features.

The Process

The AA is the foundation for development and testing. Therefore, the interfaces have to be designed carefully. Three aspects should be considered in this context.

- The AA should not be too restrictive. A developer must be able to implement the requirement according to the given interfaces and programming conventions on his platform. For example, experience showed that developers are reluctant to adopt the AA when it imposed capitalization rules on method names (C# versus Java).
- Maintainability of code is crucial. The later a bug is recognized, the more expensive its removal will be. This general rule of software development multiplies by the number of supported platforms. Eventual changes cannot be completely avoided but an elaborated AA can minimize them.
- The components testability must be ensured. The granularity of the interfaces determines the possibilities of equivalence testing. In other words, the more details a component offers through its interfaces, the more comprehensively it can be tested.

Difficulties can also arise if a further platform is added to a project in progress. The features and restrictions of this new platform were probably not considered during the architecture phase. As a result, the existing AA might not be applicable to the new platform. Therefore, all target platforms should be ideally fixed at the beginning of the project.

A solid experience with the target platforms is essential for a good and stable architecture. Ideally each platform is represented by a skilled developer within the architecture team. During the design of the AA his task is to ensure that the common architecture complies with his platform.

If necessary, prototyping critical parts of the AA can help to discover whether they can be implemented on a specific platform. The main goal is to identify any problems as soon as possible.

Establishing an AA is a challenging task. It is about finding the right level of abstraction: being specific enough in order to build testable implementations and at the same time abstract enough to give the specific implementations freedom to make use of the peculiarities of their hosting platform.

One advantage worth mentioning about AAs is that all the development teams that work on the different platform ports have a common basis that they have to agree to. Furthermore it simplifies the exchange of ideas and experiences as it provides a common understanding and vocabulary for those involved.

Although an AA can be specified in any suitable platform-independent notation, we value UML because of the benefits of its standard representation and the diverse tools to generate code skeletons for the target platforms. Thus possible misunderstandings and misinterpretations of the AA can be avoided.

Equivalence Testing Internals

In this chapter we describe the functional principle of the equivalence testing tool along with the intentions which directed its development. The main objective is to verify that two or more implementations of an application act equivalently. This equivalence is verified with respect to a test set. Tests are written once against the AA and then applied to all specific implementations.

There are two major approaches which allow the application of one common test set to many different platforms. On the one hand, test cases could be defined in an abstract language as described in [4]. These tests then need to be translated into platform specific executable code. This system makes use of user defined transformation rules to perform this translation. On the other hand, tests can be written in a concrete programming language. The second approach requires a means for inter-platform communication to interact with test targets on different platforms.

We followed the second approach as it appears to be more promising with regard to its applicability to other projects and application domains. Our approach requires the onetime implementation of one inter- platform communication system (IPCS) per platform. This IPCS can then be reused for any equivalence testing project. In fact we provide these IPCS for Java (Android), .Net (Windows Mobile) and M-shell (an exotic Symbian based system). In comparison to the first approach, our system allows the reuse of third party libraries such as testing and mocking frameworks without additional work.

The resulting equivalence testing environment consists of a test server and test clients which connect over the network. Mobile devices equipped with the pre-installed application (the test target) plus an additional test runner software act as the test clients. The test server is responsible for handling the clients, the test execution and the test result reporting. Individual tests access transparently their mobile target through the IPCS. Entry point to the mobile target is the already mentioned component registry. This IPCS mechanism supports value and reference semantics for parameter and return values. Therefore, it is possible to pass objects between both communication partners. In the case of reference types, proxies are created on the remote site which forwards the communication to the real instance.

One big advantage of this testing infrastructure is the possibility of utilizing mocks. Mocking is an established approach to replace dependencies during component or integration testing. Mocks are configured as part of the test and the IPCS now allows these mocks to be injected into the mobile target by installing forwarding proxies into their component registry. This works, fully transparent from the perspective of both the mobile client as well as the test code. Without remote mock objects we would have to provide and maintain mocks for every platform.

Fig. 1 shows a simple example of a test execution. The goal of the test is to verify that component C1 on the test client behaves according to its specification. To test C1 in isolation, its dependency to C2 should be replaced by a mock. Let's see how this works. First the test client establishes a connection to the server. The server then locally starts a test. Inside the test, a mock object M2 is instantiated and its expected behavior is configured. To be able to invoke methods on C1 the test needs to get a handle to this component. It retrieves this handle C1' through the mandatory component registry which is exposed by the IPCS. C1' acts as a proxy for the real component C1. Now the test injects the mock M2 into the same component registry. This leads to a proxy M2' on the test client which delegates method calls to its server hosted counterpart. Finally a method on

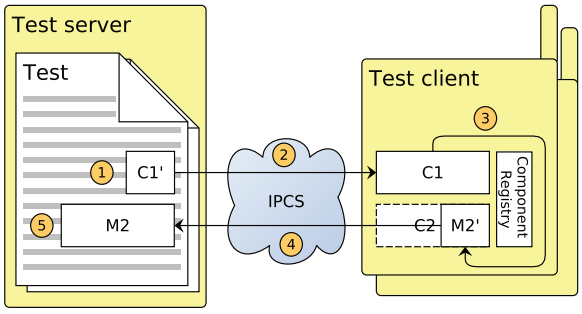


Figure 1: Overview of our testing infrastructure

C1' can be invoked (1) to verify that it returns the expected value. This call is forwarded (2) to the real component C1 on the test client. C1 requires functionality of C2 to complete the request. Since C2 is replaced by a mock, it calls a method on the injected proxy M2' (3). M2' forwards the call (4) to the mock, M2, which answers with its preconfigured return value. After all the invocations of the first call have returned (1) the test compares the returned result with the expected value and validates that the mock is being used in the intended way.

Case Study

To validate the testing approach we implemented and tested a flashcards application on two mobile platforms. In this section we demonstrate the process described in this paper on the basis of the flashcards application.

A flashcards application can manage multiple card boxes which can contain any number of cards. Card boxes are downloaded from the internet and stored on the mobile device. If the application is closed or interrupted, it must be possible to resume the learning session. The user interface should follow the platform specific guidelines.

The Abstract Architecture

Based on these requirements we defined the AA as shown in Fig. 2. The interface IFlashcardsCompReg is the component registry of the application. Starting from this interface it is possible to access the IStorageService and the IDownloadService. As the names imply the first is responsible for persisting and querying card boxes whereas the second downloads them from the web. The remaining interfaces – except IList - represent the domain model of the application.

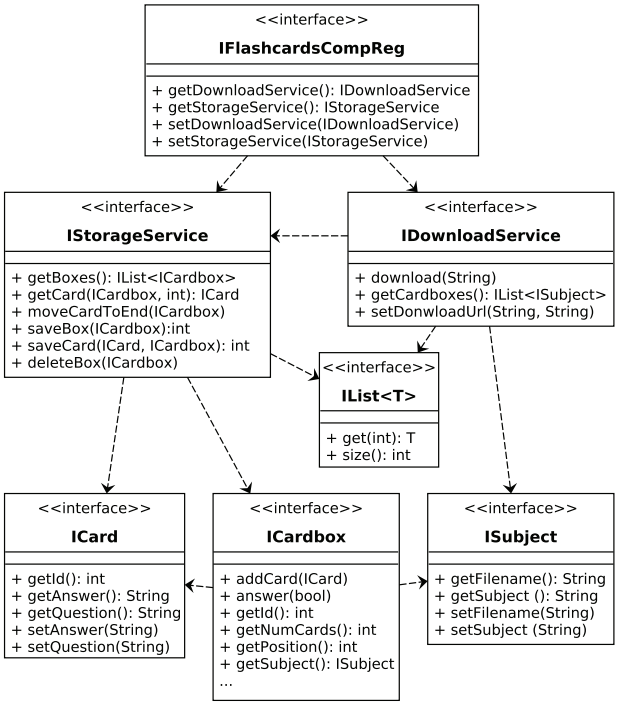


Figure 2: Abstract architecture of flashcards

This architecture satisfies the criteria described in the previous sections. It is platform independent because there are no references or assumptions to concrete target platforms. For example, the assignment of identifiers to cards and card boxes is ceded to the implementation. Another example is the IList interface. This custom interface defines a platform- independent way to work with lists. In addition a component registry makes it possible to replace real components before test runs.

All important parts of the flashcards application are covered by the AA. Tests can now verify the correctness of the download and the persistence of the card boxes and cards independent of the platform. Furthermore, the correct handling of cards during a learn session is testable (moveCardToEnd(...)). UI aspects did not affect the abstract architecture as they are handled platform specific.

Behavior Specification

So far, only the static structure of the flashcards application has been defined through the AA. What is missing is the specification of the behavior. In order to implement the AA on several platforms the developers need an accurate specification of the components, their responsibilities and how they should interact amongst each other.

We prefer test cases to specify the desired behavior. Compared to a specification document where the behavior is specified in natural language, tests have several benefits. They are executable and verify quickly if an implementation fulfills the tested behavior or not. Moreover tests positively affect software design in regards to reusability and coupling. As tests are, anyway, necessary we suggest using tests as specifications wherever possible. This also avoids the unnecessary inconvenience of having to keep the tests and the documentation in sync.

One benefit of a document in which the requirements are formulated in natural language is readability. People with a non-technical background, i.e. the customer, will be able to read such a document. Behavior driven design libraries such as specs [9] for Scala [10], fill this gap. With specs it is possible to write software specifications, which a technical layman can understand, by combining natural language with test code. The test results are presented in a comprehensible way, too. Examples will follow in the next section.

Writing Specifications

Now let us take a look at the flashcards application specifications. As our test server is based on Scala, we used specs to write specifications. In addition Mockito [11] is used for mocking. Any

```
class DownloadServiceSpec extends FlashcardsSpec {

  "DownloadService" should {
    val compReg = entryPoint[IFlashcardsCompReg]

    "invoke storage of box and cards" in {
      //init mocks
      val storage = mock[IStorageService]
      compReg.setStorageService(storage)

      //stub the mocks
      storage.saveBox(any[ICardbox]) returns 1

      //do actions
      val ds = compReg.getDownloadService()
      ds.setDownloadUrl("http://.../boxes/index", http://.../boxes/{0}")
      ds.download("Test_Card_Box")

      //verification
      there was one (storage).saveBox(any[ICardbox])
      there was atLeastOne(storage).saveCard(any[ICard], any[ICardbox])
      noMoreCallsTo(storageMock)
    }
  }
}
```

Listing 1: DownloadService specification example

other testing or mocking library running on the JVM (e.g. JUnit [12] or EasyMock [13]) would have been possible, too.

Lst. 1 shows a specification for the DownloadService. It specifies the behavior triggered by a call to the download method.

FlashcardsSpec, the base class of this specification, performs setup and teardown tasks such as cleaning up the mobile's internal storage where the card boxes and cards are stored. The StorageService methods are used for this. Besides, it provides the entryPoint method which is used to access the component registry of the application being tested. Once we have the IFlashcardsCompReg instance, we can access the defined services. The rest of the listing does not differ from a usual specification written in specs and supported by Mockito. First a mock for the StorageService is created and injected in the component registry. The mock is then programmed to return the integer 1 whenever the saveBox method is called. After that the DownloadService's download method is used to retrieve a prepared card box from a web address. If the implementation follows the defined behavior, it will connect to the webserver, download the card box and use the StorageService to persist the box and its cards. The specification ends with verifying the mock. It is expected that one card box and multiple cards have been saved.

In addition we might have checked if the card box was downloaded and parsed correctly. Special characters in the questions and answers could motivate this test. Further verifications were skipped to keep the example short. Still this specification contains all relevant parts and hopefully showed the capabilities of equivalence testing.

A comprehensive explanation of each detail and further test cases would go beyond the scope of this paper. Therefore, we would like to refer to our project site [14]. In addition the flashcards application implemented on Android and Windows Mobile is available there.

Related Work

There are different approaches to ensure that a ported application does the same as the original one. Basically these approaches are listed in [1] and can be split into three categories:
Single adaptive implementation:

- *Web applications* seem to provide an ideal solution to the porting problem but unfortunately the available browser engines interpret the HTML, CSS and JavaScript code differently. A major disadvantage is a lack of access to device features as they are shielded by browser security. Performance of JavaScript often lags behind native implementations.
- *Cross-platform frameworks* are often based on web technologies, too. Deployment is done natively on each platform. Instead of the native web browser, a web view is executing the application. This concept enables cross-platform frameworks to access device features over a JavaScript-to-Native-Bridge. A typical representative of this category is PhoneGap [7].
- *Cross-compiling* approaches translate the code of a high level programming language to native code of the target platform. Qt Mobility is a representative of this category [15].

  For single adaptive implementations, testing does not substantially differ from our approach. As tests are part of the single code base,

they can be executed on each target platform without adaption.

Deriving multiple implementations from a common model:

- A similar framework to ours is [6], where a Domain-Specific Modeling Language (DSML) is introduced to specify test scenarios. The DSML abstracts from the variability between platforms and devices. In combination with a test bed, this approach is able to automatically black box test mobile applications. A test specification interacts with the application on the same level a user does. Therefore, it is not possible to verify the system on the component level.
- Equivalence testing also follows a model driven approach: The AA represents the model against which tests are implemented. It contains the interface definitions which are used for generating proxies for mock objects.
  A pure MDA approach would supersede testing as there is only generated code. In practice only code templates are generated. These have to be extended manually with detailed code. Production and test code has to be adapted for each platform.

Manually porting an application:

- Basically this is what our approach supports: manually porting the application and then use equivalence testing to do integration tests and ensure equivalent functionality on all platforms. The "overhead" of defining an AA is amortized by the reduced effort that has to be put into testing.

Conclusion

Equivalence testing is not suitable for each kind of application. Applications that are merely GUIs for web services do not profit from equivalence testing. Equivalence testing pays off for applications with a high amount of A-components (i.e. application logic).

Seen from a testing perspective, the single adaptive implementation approach is closest to our approach. It only needs one test suite that can be run on all target platforms. Unfortunately this approach does not suit the need of all mobile development projects. This is the case if runtime performance is an issue, if the application uses features that are not supported by the cross-platform framework or if it shall adhere closely to the native look and feel.

We encourage native application development and provide a process as well as tools to pragmatically support multi-platform development. In order to have implementations with equal behavior, our tools enable component and integration testing with a single test set. All of the components that are defined in the common abstract architecture can be tested. Compared to other approaches the effort of integration test development remains constant and does not grow with each target platform.

Finally we would like to explicitly point out that equivalence testing does not replace other tests that usually are conducted during a software project. GUI or unit tests still need to be provided per platform. Equivalence testing as presented here can only be applied to components, but not to classes within these components. However, platform independent tests are written and maintained in only one place avoiding porting errors and consuming less time.

References

[1] Rajapakse D. C., 2008, Techniques for De-fragmenting Mobile Applications: a Taxonomy. Proceedings of 20th International Conference on Software Engineering and Knowledge Engineering (SEKE '08).San Francisco, USA, pp. 923-928

[2] Allen S. et al, 2010. Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile, and Android Development and Distribution. Apress, New York, USA

[3] Haft M. et al, 2005, The Architect's Dilemma – Will Reference Architectures Help? Quality of Software Architectures and Software Quality (QoSA-SOQUA 2005), Lecture Notes in Computer Science 3712. Erfurt, Germany, pp. 106-122

[4] C. Liu, 2000, Platform-independent and tool-neutral test descriptions for automated software testing. Proceedings of the 22nd international conference on Software engineering (ICSE '00), Limerick, Ireland, pp. 713-715

[5] Miravet P. et al, 2009 DIMAG: A Framework for Automatic Generation of Mobile Applications for Multiple Platforms. Proceedings of the 6th International Conference on Mobile Technology, Application & Systems (Mobility '09), Nice, France, pp. 23:1-23:8

[6] Ridene Y. and Barbier F., 2011, A model-driven approach for automating mobile applications testing. Proc. of the 5th European Conference on Software Architecture: Companion Volume (ECSA '11), Essen, Germany, pp. 9:1-9:7

[7] PhoneGap site, http://www.phonegap.com (28. June 2012)

[8] Sencha Touch site, http://www.sencha.com/products/touch (28. June 2012)

[9] Specs site, http://code.google.com/p/specs (28. June 2012)

[10] Scala site, http://www.scala-lang.org (28. June 2012)

[11] Mockito site, http://code.google.com/p/mockito (28. June 2012)

[12] JUnit site, http://www.junit.org (28. June 2012)

[13] EasyMock site, http://easymock.org (28. June 2012)

[14] Eq-Testing project site, http://www.assembla.com/space/eq-testing (28. June 2012)

[15] Qt-Mobility a sub-project of Qt. Site: http://qt-project.org/ (28. June 2012)

Eine Kategorisierung mobiler Applikationen

Mobile Applikationen lassen sich auf verschiedene Arten entwickeln. Je nachdem welche Anforderungen an eine Applikation gestellt werden, sollte zwischen einer nativen Umsetzung, einer Webapplikation oder einer Mischung aus beidem gewählt werden. Die Wahl der am besten passenden dieser drei Kategorien ist nicht immer einfach und sollte im Vorfeld genau abgeklärt werden. In diesem Artikel zeigen wir die Eigenheiten dieser drei Kategorien auf und vergleichen sie miteinander anhand von Beispielen und einer fiktiven Ticketing-Applikation.

Andreas Hildebrandt, Jürg Luthiger, Christoph Stamm, Chris Yereaztian | juerg.luthiger@fhnw.ch

Mobile Applikationen für Smartphones, so genannte Apps, lassen sich grob in drei Kategorien einteilen: native Apps, Webapplikationen und eine Mischung aus beidem, die hybriden Apps. Nicht alle Anwendungen lassen sich gleich gut mit allen drei Kategorien umsetzen, da jede Kategorie ihre Vorzüge und Nachteile mit sich bringt.

In diesem Artikel charakterisieren wir die drei Kategorien und geben für jede Kategorie ein bekanntes Beispiel aus der Praxis an. Zudem versuchen wir anhand einer fiktiven Ticketing-App abzuschätzen, welche Probleme bei der Umsetzung dieser App in den drei Kategorien auftreten könnten. Die Ticketing-App soll das Bestellen und Verwalten von Tickets vereinfachen. Ähnlich der Ticketing-App der SBB sollen von möglichst vielen Benutzern Tickets über das Web bestellt und lokal auf dem Gerät gespeichert werden können. Nach dem Ticket-Download soll keine weitere Internet-Verbindung mehr notwendig sein. Dies erleichtert und beschleunigt den Abruf des Tickets bei der Ticketkontrolle. Durch einfaches Schütteln des Smartphones soll das letzte Ticket zum Vorschein gebracht werden, ohne dass die Benutzerin das Gerät mit den Fingern bedienen muss.

Native Anwendungen

Native Anwendungen werden in vielen Fällen speziell für die entsprechende Zielplattform geschrieben und direkt auf dem Gerät ausgeführt. So werden beispielsweise iPhone-Apps hauptsächlich in Objective-C und Android-Apps in Java entwickelt. Daneben gibt es aber auch vermehrt Ansätze, native Anwendungen für mehrere Plattformen gleichzeitig zu entwickeln. Die Firma Xamarin[1] bietet zum Beispiel die Möglichkeit, den Quellcode des Logikteils für alle angebotenen Plattformen in C# zu schreiben. Das User-Interface wird danach plattformspezifisch gestaltet und entwickelt.

Weil native, mobile Applikationen plattformspezifisch implementiert sind, kann das plattformtypische Look-and-Feel vollständig umgesetzt werden. Dadurch fühlt sich die Benutzungsschnittstelle der App für eine mit der Plattform vertraute Benutzerin sehr intuitiv an, während ein plattformfremder eine gewisse Eingewöhnungszeit benötigt. Die Bedienung ist zudem bestens auf die Hardware abgestimmt und flüssig. Native Anwendungen können über die betriebssystemspezifische Schnittstelle (API) alle Funktionen des Gerätes direkt nutzen. Durch das API hat der Entwickler auch vollen Zugriff auf alle Komponenten und Sensoren des Gerätes. Somit lassen sich die Vorteile des jeweiligen Gerätes voll ausnutzen.

Typische native Applikationen sind Kamera-Anwendungen, Spiele oder VoIP-Anwendungen. Abbildung 1 zeigt exemplarisch die unterschiedliche Umsetzung der Chatfunktion von Skype auf iOS und Android. Auf dem rechten Bild der iPhone-App ist ein Back-Button oben in der Navigationsleiste vorhanden und unten ist die Tab-Bar gemäss Apple Guidelines platziert. Auf dem linken Bild der Android-App ist die typische Action Bar des Android-Systems zu sehen. Einen Back-But-

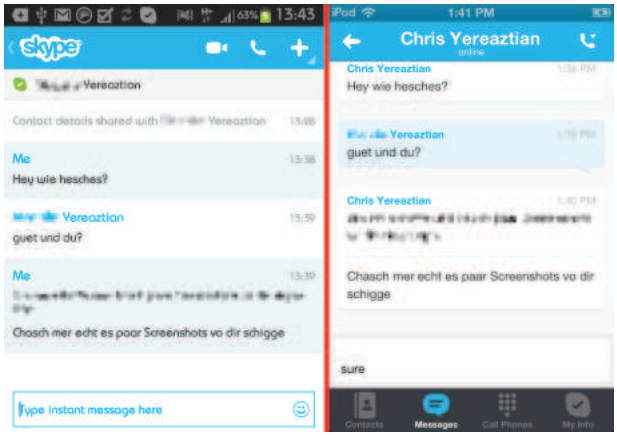


Abbildung 1: Plattformspezifische Umsetzung der Skype-Chatfunktion unter iOS (links) und Android (rechts). Namen und privater Text unleserlich gemacht.

1 Xamarin: http://xamarin.com/

```
<activity android:label="@string/app_name" android:name="com.phonegap.DroidGap"
      android:configChanges="keyboardHidden|orientation»>
  <intent-filter />
</activity>
```

Listing 1: Manifest-Datei mit PhoneGap-typischem Eintrag

ton braucht es bei Android-Geräten nicht, da ein Hardware Button genutzt wird.

Der Vertrieb nativer, mobiler Anwendungen wird meistens über einen entsprechenden App-Store realisiert, zum Beispiel den *App Store* für iPhone oder den *Google Market* für Android. Dabei ist zu beachten, dass bei einigen App-Stores der Genehmigungsprozess viel Zeit beanspruchen kann.

Eine native Realisierung der skizzierten Ticketing-App ist grundsätzlich möglich, da alle notwendigen Funktionalitäten und Sensorzugriffe möglich sind. Performance-Probleme wird es keine geben und das Look-and-Feel entspricht demjenigen anderer nativer Apps. Für den Entwickler bzw. Anbieter der App erhöht sich der Aufwand mit der Anzahl verschiedener Plattformen, die unterstützt werden sollen. Die App und spätere Anpassungen müssen über die App-Stores verteilt werden und stehen somit erst mit einer gewissen Verzögerung von ein paar Tagen zur Verfügung.

Hybride Anwendungen

Hybride Apps bestehen aus zwei unterschiedlichen Komponenten: Die native Komponente kann direkt auf die betriebssystemspezifischen Funktionalitäten zugreifen, während die Web-Komponente in einem entsprechenden Container läuft. Dabei wird angestrebt, möglichst grosse Teile der Applikationslogik in der plattformunabhängigen Web-Komponente zu implementieren. Nur dort wo auf plattformspezifische Funktionen zugegriffen werden muss, wird die native Komponente eingesetzt. So bietet die native Komponente Zugriff auf Kamera, Kompass oder andere Sensoren, für die es keine standardisierten Schnittstellen gibt.

Zur Realisierung hybrider Apps kommen oft Frameworks, wie zum Beispiel PhoneGap oder Titanium, zum Einsatz. Diese Frameworks stellen die Kommunikation zwischen dem nativen und dem Web-Teil sicher. Die zu verwendende Programmiersprache wird dabei durch das Framework festgelegt. Einige Frameworks nutzen die Websprachen HTML, JavaScript und CSS, andere wiederum verwenden C#.

Ein typisches Beispiel für eine hybride, mobile Anwendung ist die Wikipedia-App von Wikimedia Foundation[2]. Wikipedia bietet für mobile Geräte, seien dies Tablets oder SmartPhones, eine optimierte Variante ihrer Webseite (inklusive aller Artikel) an, die mit den üblichen mobilen Browsers kompatibel ist. Die Applikation verwendet auf Android sowie auf iOS PhoneGap[3], um die auf beiden Plattformen verfügbare SQLite-Datenbank verwenden zu können[4]. Einmalig heruntergeladene Artikel werden in der SQLite-Datenbank zwischengespeichert, so dass die Applikation auch ohne Internet-Verbindung verwendet werden kann. Artikel zu bereits gesuchten Begriffen können somit auch unterwegs ohne Internetverbindung gelesen werden.

Da die Wikipedia-App keine speziellen grafischen Elemente wie Check-Boxen oder Auswahllisten verwendet, lässt sich nicht so einfach erkennen, dass es sich um eine Applikation handelt, die mit PhoneGap umgesetzt worden ist. Jedoch lässt sich anhand des Quellcodes oder der Installationsdatei (bei Android das Application Package File) schnell feststellen, ob eine Applikation ein bestimmtes Framework verwendet oder nicht. Die Installationsdatei einer Android-Applikation kann mit dem *Android Asset Packaging Tool* entpackt werden. Im Hauptverzeichnis der Applikation befindet sich die Manifest-Datei, in der die einzelnen Bestandteile der Applikation definiert sind. PhoneGap-Applikationen besitzen üblicherweise einen Eintrag für die Start-Activity, welche die index.html Datei samt dazugehörigen JavaScript lädt (siehe Listing 1).

Das hybride Modell bietet den Vorteil, dass die gleiche Applikationslogik für mehrere Plattformen genutzt werden kann, da sie nicht spezifisch für eine Plattform geschrieben wird, sondern gegen ein Framework wie z.B. PhoneGap oder Titanium[5]. Das Framework liefert den In-Between-Code, der den plattformunabhängigen Code in die native API übersetzt. Je umfangreicher die mobile Anwendung ist und je spezieller die Funktionen sind, desto aufwendiger werden jedoch diese Anpassungen.

Bei UI-Elementen ist speziell zu beachten, dass die häufigsten Frameworks im Look-and-

Abbildung 2: PhoneGap Demo-App auf einem Android SmartPhone mit iPhone Look-and-Feel

2 Wikimedia Foundation: http://www.wikimedia.org/
3 PhoneGap: http://phonegap.com/
4 Wikipedia nutzt PhoneGap: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/customer-success/pdfs/wikimedia-foundation-case-study.pdf
5 Appcelerator Titanium: http://www.appcelerator.com/

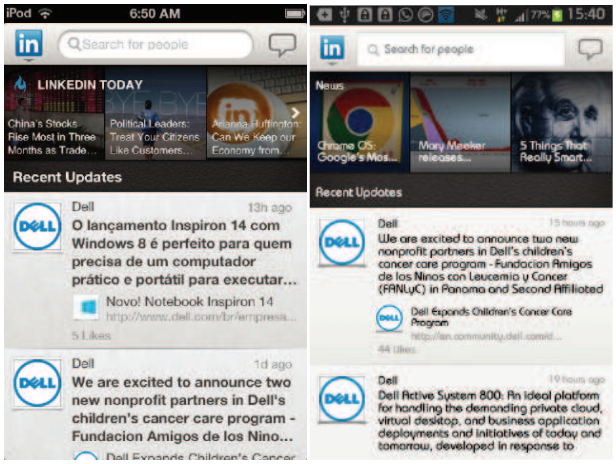


Abbildung 3a: Gegenüberstellung der Webapplikation LinkedIn unter iOS (links) und Android (rechts)

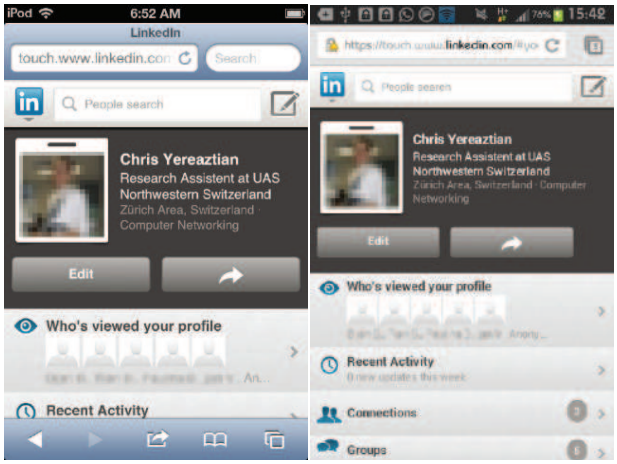


Abbildung 3b: Gegenüberstellung der Webseite LinkedIn im Safari (links) und Chrome Mobile (rechts). Profilbild und Namen unleserlich gemacht

Feel das native iOS nachahmen, so dass derartig umgesetzte UI-Elemente auf einem Android- oder Windows-Gerät nicht wie native Applikationen wirken. Abbildung 2 zeigt eine PhoneGap Demo-Applikation, welche Daten aus dem Beschleunigungssensor ausliest und darstellt. Für Android und Windows Phone bringt der Back-Button oben Links keinen zusätzlichen Nutzen, da diese Funktionalität über einen Hardware-Button abgedeckt wird. Die iOS-Geräte jedoch besitzen nur einen einzigen Hardware-Button, den sogenannten Home-Button, und deshalb sind die iPhones auf solche UI-Buttons angewiesen. Die dargestellten Listenelemente sehen den Listenelementen auf iOS sehr ähnlich, nutzen aber nicht die von der Plattform angebotenen Implementierungen.

Der Vertrieb hybrider, mobiler Anwendungen wird wie bei nativen Applikationen über einen entsprechenden App-Store realisiert.

Eine hybride Realisierung der skizzierten Ticketing-App ist grundsätzlich möglich, da alle notwendigen Funktionalitäten und der Zugriff auf den Bewegungssensor möglich sind. Mit Performance-Problemen ist vorliegenden Fall kaum zu rechnen, da die Benutzungsoberfläche recht einfach gehalten werden kann. Das Look-and-Feel wird höchstens auf einer Plattform demjenigen nativer Apps entsprechen. Weil eine gemeinsame Code-Basis für mehrere Plattformen verwendet werden kann, bleibt der Aufwand für den Entwickler mehrheitlich unabhängig von der Anzahl der unterstützten Plattformen, solange das verwendete Framework diese Plattformen unterstützt. Diese Abhängigkeit eines Frameworks kann zudem nachteilig sein, wenn die Weiterentwicklung des Frameworks aus irgendwelchen Gründen ins Stocken gerät. Wie bei einer nativen App müssen die App und spätere Anpassungen über die App-Stores verteilt werden und stehen somit erst mit einer gewissen Verzögerung von ein paar Tagen zur Verfügung.

Webapplikationen

Eine klassische Webapplikation wird auf einem Web-Server ausgeführt und verwendet üblicherweise den Web-Browser des Clients als Benutzungsschnittstelle. Mit HTML5 und der Verwendung von JavaScript verschiebt sich jedoch wieder ein grosser Teil des ausführbaren Codes vom Server auf den Client. Somit übernimmt der Browser die Aufgabe eines standardisierten Applikationsfensters mit den dazugehörigen UI-Elementen. Anpassungen beschränken sich dadurch hauptsächlich auf die unterschiedlichen Bildschirmdarstellungen bei den verschiedenen Display-Grössen.

Mit dem Verzicht der Nutzung nativer Schnittstellen, gehen gewisse Einschränkungen bezüglich der Nutzung von Sensoren und Kommunikationsschnittstellen einher. Um damit zusammenhängende Performance-Einbussen bei der Ausführung von sequentiellem JavaScript Code zu umgehen, definiert HTML5 einen neuen Web-Worker-Standard, welcher es erlaubt, JavaScript Code in einem separaten Thread auszuführen[6].

Ein Beispiel einer Web-Anwendung ist die LinkedIn-App[7]. Diese verwendet unter Android und iOS nur HTML5 und JavaScript und verzichtet dabei ganz auf ein Framework. Die Applikation nutzt eine sogenannte Web-View, um HTML-Seiten in einer Applikation mit der Rendering-Engine des System-Browsers anzuzeigen, ohne dass dafür spezieller Code benötigt wird.

In Abbildungen 3a und 3b ist zu sehen, wie sich die Grösse der Schrift und die Anordnung der dazugehörigen Elemente unterscheiden. Mit HTML lässt sich die Schriftgrösse ohne weiteres an die Bildschirmauflösung angepasst.

Der Vertrieb von reinen Webapplikationen beschränkt sich auf die Aktualisierung der Applikation auf einem Web-Server. Korrekturen und Ak-

6 Einführung Web-Worker: http://www.html5rocks.com/en/tutorials/workers/basics/

7 Link zur LinkedIn-App: https://www.linkedin.com/

tualisierungen können daher in viel kürzerer Zeit veröffentlich werden. Zudem ist sichergestellt, dass alle Benutzerinnen die neuste Version der Applikation einsetzen.

Eine Realisierung der skizzierten Ticketing-App als Webapplikation ist für die Bestellung und Bezahlung des Tickets problemlos möglich und sogar sinnvoll, da dadurch möglichst viele potentielle Benutzer angesprochen werden können. Sogar das Abrufen des aktuellen Tickets ist mittlerweile möglich, da mittels HTML5 und JavaScript der Bewegungssensor abgefragt werden kann. Mit Performance-Problemen ist allenfalls bei der Detektion der Schüttelbewegung zu rechnen, da der Bewegungssensor in kurzen Abständen mehrfach hintereinander aufgerufen werden muss. Das Look-and-Feel wird sich von demjenigen nativer Apps unterscheiden. Weil eine gemeinsame Code-Basis für mehrere Plattformen verwendet werden kann, bleibt der Aufwand für den Entwickler mehrheitlich unabhängig von der Anzahl der unterstützten Plattformen. Die Verteilung der Webapplikation ist denkbar einfach und Anpassungen oder neue Funktionen lassen sich schnell und einfach verbreiten.

HTML5-Hype

Wie bereits angesprochen, verschiebt sich mit dem Einsatz von HTML5 und JavaScript wieder ein grosser Teil des ausführbaren Codes vom Server auf den mobilen Client. Dadurch verwischen sich die Grenzen zwischen nativen Apps und mobilen Webapplikationen immer mehr. Nicht zuletzt deswegen gewinnt die Entwicklung von HTML5-Apps immer mehr an Bedeutung. Viele der Applikationen, die heute in den App-Stores angeboten werden, sind in HTML5 und JavaScript geschrieben. Weitere Gründe für diesen Hype können folgende Punkte sein:

- Mobile Webapplikationen sind einfacher und schneller zu implementieren. Um eine native Applikation für ein iPhone oder Android Gerät zu schreiben sind jeweils eigene Entwicklungsumgebungen notwendig. Die iPhone Entwicklung kann zusätzlich nur auf einen Computer von Apple erfolgen. Neben diesen Voraussetzungen braucht es Kenntnisse in einer Programmiersprache der entsprechenden Plattform (Android: Java, IPhone: Objective-C, …). Da HTML sowie JavaScript einfacher zu erlernen sind, dürfte die Hürde, eine eigene App zu schreiben, sehr viel tiefer liegen.
- Die momentan starke Fragmentierung der mobilen Betriebssysteme wirkt sich negativ auf die Entwicklung mobiler Apps aus. Unter Android gibt es die komplette Palette von der Version 1.5 bis hin zu 4.1[8]. Unter iOS und Windows Phone sieht es zwar nicht ganz so extrem aus, aber es gibt auch hier verschiedene Versionen. Mit HTML5 kann eine Anwendung geschrieben werden, die unabhängig von der aktuellen und der genutzten OS Version lauffähig ist und keine Einschränkungen besitzt.
- Durch die Plattformunabhängigkeit von Webapplikationen ist es sehr viel einfacher, eine App zu veröffentlichen. Eine App, die in HTML und JavaScript geschrieben ist, kann leicht von einem Android- auf ein iOS- oder Windows-Gerät portiert werden. Damit lassen sich sehr viel mehr Benutzer erreichen, während der Vertriebsaufwand minimal gehalten werden kann.
- Schliesslich sind Webapplikationen nicht an Restriktionen der jeweiligen Systeme gebunden. Das heisst, dass bei einem Release einer neuen Version die App praktisch sofort den Benutzern zugänglich gemacht werden kann. Auf iOS oder Windows Phone muss eine App erst eine längere und kostenpflichtige Prozedur durchlaufen, bevor sie für den Store freigegeben wird.

Native Apps versus (hybride) Webapplikationen

Im eingangs aufgezeigten Szenario mit der mobilen Ticketing-Anwendung spielt die möglichst grosse Verbreitung eine sehr zentrale Rolle und daher ist die Plattformunabhängigkeit von grösster Wichtigkeit. Reine oder hybride Webapplikationen haben in einer solchen Situation einen klaren Vorteil gegenüber nativen Apps. Die Verwendung einer reinen Webapplikationen hängt jedoch nicht zuletzt davon ab, ob Sensoren benötigt werden und sich diese von der Webapplikation ansteuern lassen. Der zweite Knackpunkt ist die Effizienz. Es stellt sich also die Frage, ob die Performance einer Web-Anwendung ausreicht oder ob einzelne Teile oder eventuell sogar die ganze Anwendung nativ realisiert werden müssen.

Die einst klaren Grenzen zwischen nativen Apps und Webapplikationen verwischen sich immer mehr. Speziell die neuen HTML5-Standards[9] ermöglichen es, dass sich eine mobile Webapplikation immer mehr wie eine richtige native Applikation verhält. So erlaubt z.B. der Standard Web-Storage[10], Daten über mehrere Browser-Sessions hinweg offline zu sichern. Dem Benutzer kann so ein Gefühl einer nativen App vermittelt werden auch wenn keine Internet-Verbindung zur Verfügung steht. Hingegen besteht aber immer noch das Problem, dass die Grafikelemente in den HTML5-Applikationen wie Buttons, Listen etc. sich stark von den nativ entwickelten Elementen unterscheiden. Eine Ausnahme dazu liefert aber

8 Fragmentierung Android OS: http://developer.android.com/about/dashboards/index.html, http://mashable.com/2012/05/16/android-fragmentation-graphic/

9 Übersicht der mobilen HTML5-Standards: http://mobilehtml5.org

10 Browser Storage Support: http://www.html5rocks.com/de/features/storage, http://csimms.botonomy.com/2011/05/html5-storage-wars-localstorage-vs-indexeddb-vs-web-sql.html

	nativ	hybrid	Webapp
Grafiken	native API	HTML, Canvas, SVG	HTML, Canvas, SVG
Video/Audio	Ja	Ja	Ja
3D Grafiken (WebGL)	Ja	Ja	Ja
Kamera	Ja	Ja	Nein
Notifikationen	Ja	Ja (1)	Nein
Kontakt	Ja	Ja	Nein
Kalender	Ja	Ja (2)	Nein
Offline Storage	Ja	Ja	Ja
Geolokation	Ja	Ja	Ja
Beschleunigungssensor	Ja	Ja	Ja

Tabelle 1: Unterstützung mobiler Komponenten in den drei Applikationskategorien. (1) Nicht mit jedem Framework und/oder mit Urban Airship. (2) Zum Teil über Third-Party Module.

JQuery[11] mit der JavaScript-Bibliothek JQuery-Mobile. Das Design der Elemente ist dort stark an iOS angelehnt.

Der grosse Vorteil einer hybriden gegenüber reinen Webapplikation liegt darin, dass die erstere über die native Komponente des Frameworks auf Teile der nativen API zugreifen kann, während die letztere Bibliotheken in JavaScript benötigt, die wiederum nur eingeschränkten Zugriff auf die Hardware ermöglichen. In Tabelle 1 ist die Unterstützung der wichtigsten Elemente von mobilen Plattformen in Abhängigkeit der drei Kategorien von mobilen Applikationen aufgelistet. Beispielsweise lässt sich auf die Kontakt- und Kalender-APIs[12] und die diversen Sensoren nur von nativen und hybriden Apps zugreifen. Da nicht alle Frameworks immer sämtliche Funktionen aller Plattformen unterstützen, spielt die Wahl des gewählten Frameworks sowie des Betriebssystems eine entscheidende Rolle über die Integrationsmöglichkeiten nativer Funktionalität.

Werden APIs angepasst oder gar neu bereitgestellt (z.B. Notifications für iOS oder NFC für Android 2.3.x), können diese bei hybriden, mobilen Apps oft nicht gleich genutzt werden. Die Framework-Entwickler brauchen dafür erst Zeit um diese Schnittstellen in das Framework zu integrieren und anschliessend freizugeben.

Performance-Betrachtung

Die Performance einer hybriden, mobilen Applikation mit einer schlicht gestalteten Benutzeroberfläche kann durchaus mit einer nativen App mithalten. Ist das User Interface komplexer aufgebaut, reagiert die Anwendung schwerfälliger, weil die Darstellung von nicht nativen UI-Elementen durch die CPU aufwendig berechnet werden muss[13]. Im Vergleich mit einer reinen mobilen Webapplikation ist eine hybride, mobile Anwendung nicht an die Sandbox des Browsers gebunden, sondern unterliegt den gleichen Einschränkungen (Anzahl Threads, Speicherbedarf) wie eine native App.

Abbildung 4: Aktualisierung der Beschleunigungssensordaten im Vergleich. Links: native Test-App; Mitte: Test-App mit PhoneGap unmodifiziert; rechts: Test-App mit PhoneGap modifiziert, ohne Beschränkung der Aktualisierungsrate.

Bei einer Webapplikation hängt die Performance stark von der Gestaltung der Benutzeroberfläche ab. Je aufwändiger diese gestaltet ist, desto langsamer läuft die Anwendung. Dabei kann die Performance-Einbusse an der Geräteleistung liegen oder auch durch technische Massnahmen künstlich hervorgerufen werden. Beispielsweise begrenzt Apple bei ihrem Betriebssystem die Leistung aller Browser, damit sichergestellt wird, dass für das OS genügend Leistung bereitgestellt werden kann. Diese Einschränkung führt beim Scrollen zu ruckeln und fühlt sich sehr störend an.

Bei der Facebook-App haben solche Performance-Probleme (Zeichnen des Layouts, Nachladen von News) unter iOS den Ausschlag gegeben, von einer Webapplikation auf eine native Lösung umzustellen. Innerhalb drei Wochen hat sich das Rating von 1.5 auf 4 Sterne im App-Store verbessert[14].

Zugriff auf Sensoren

Für die fiktive Ticketing-App haben wir gefordert, dass das aktuelle Ticket durch eine Schüttelbewegung des mobilen Gerätes zur Ansicht gebracht werden kann. Um eine Schüttelbewegung erkennen zu können, muss der Bewegungssensor in kurzer Folge mehrfach abgefragt werden. Dies gilt auch dann, wenn für die Schüttelbewegung eine vordefinierte Gestenerkennung vorhanden ist. In dem Fall läuft der Sensorzugriff einfach im Hintergrund ab. Aus diesem Grund haben wir eine einfache Test-App entwickelt, welche die maximale Aktualisierungsrate des Bewegungssensors ermittelt (siehe Abb. 4).

In der nativen und der hybriden App wird eine Callback-Methode registriert, die aufgerufen wird, sobald ein Update mit neuen Sensordaten zur Verfügung steht. In Android kann bei der Registrierung des Callbacks die Aktualisierungsfrequenz über einen Parameter spezifiziert werden.

11 jQuery: http://jquery.com/
12 PhoneGap Features: http://phonegap.com/about/feature
13 Browser Performance: https://www.scirra.com/blog/85/the-great-html5-mobile-gaming-performance-comparison
14 Quelle: http://techcrunch.com/2012/09/13/facebook-for-ios-review/

Gerät	nativ	hybrid ohne Modifikation	hybrid mit Modifikation
Samsung Galaxy S (Android 2.3.6)	33 Hz	10 Hz	11 Hz
Samsung Galaxy S (Android 4.1.2 - CM10)	85 Hz	10 Hz	15 Hz
Samsung Nexus S (Android 4.1.2)	45 Hz	10 Hz	—
Samsung Galaxy S III (Android 4.1.1)	155 Hz	10 Hz	15 Hz

Tabelle 2: Ausführung der Test-App auf mehreren Geräten und Betriebssystemversionen

Diesen Parameter haben wir bei unseren Messungen auf SENSOR_DELAY_FASTEST gesetzt, um die Rate nicht künstlich zu beschränken. In PhoneGap ist die API für den Zugriff auf den Beschleunigungssensor äquivalent aufgebaut wie in Android.

Auf der Android-Plattform haben Messungen mit unserer Test-App ergeben, dass auch die Verwendung von Frameworks wie PhoneGap den Performance-Unterschied von HTML5 gegenüber nativen Applikationen nicht markant verringern kann. Der Zugriff auf die Sensor-API ist mittels PhoneGap sieben- bis zehnmal langsamer gegenüber einer nativen Lösung. Die Spannweite hängt damit zusammen, dass PhoneGap die Aktualisierungsfrequenz künstlich auf ca. zehn Aktualisierungen pro Sekunde beschränkt. Einer der Gründe für diese Beschränkung könnte die Schonung der Akkuleistung sein. Durch eine Modifikation im PhoneGap-Framework ist es jedoch möglich, die Aktualisierungsfrequenz als Parameter zuübergeben. So kann die Anzahl Aktualisierungen nochmal gesteigert werden. Selbst mit diesem Trick wird keine ähnlich hohe Aktualisierungsrate wie bei der nativen Applikation erreicht. Die Aktualisierungsrate ist zudem stark von der JavaScript/HTML-Performance des Browsers abhängig und kann deswegen von Gerät zu Gerät variieren, wie in Tabelle 2 zu sehen ist.

Fazit

Die Wahl der adäquaten Kategorie einer mobilen Applikation hängt stark vom Typ der Anwendung, dem gewünschten Funktionsumfang und dem bereits vorhanden Know-how ab. Wird eine Applikation verlangt, die stark auf native APIs oder auf die Hardware zugreifen muss oder deren Performance bei einer aufwendig gestalteten Benutzungsschnittstelle kritisch ist, dürfte eine Webapplikation kaum geeignet sein. Die wohl beste Wahl würde hier eine native App darstellen.

Anders sieht es bei einer Applikation aus, die zur Konsumation von News genutzt wird und die auf möglichst vielen verschiedenen Plattformen lauffähig sein soll. Hier kann eine Webapplikation eine gute Alternative darstellen, sofern keine speziellen Sensoren gebraucht werden. Das grosse Plus der Webapplikation ist die hohe Plattformunabhängigkeit und der geringe Aufwand bei der Verbreitung von neuen Versionen.

Wird eine App gewünscht, welche mit wenig mehr Aufwand auf verschiedenen Plattformen lauffähig ist und dazu noch auf vereinzelte native Funktionen und Sensoren wie zum Beispiel die Kontaktdaten oder die Kamera zugreifen kann, dann ist eine hybride Applikation eine gute Lösung. Dabei gilt es jedoch die Performance im Auge zu behalten, denn laufende Erweiterungen und Anpassungen an Kundenwünsche können zu unerwünschten Nebenwirkungen bei der Leistungsfähigkeit führen.

Modulare domänenspezifische Sprachen

Für ein neuartiges Überwachungssystems von Lieferketten entwickeln wir mit Groovy und Java eine modulare DSL-Engine, welche es ermöglicht, eine domänenspezifische Sprache zu implementieren, deren Syntax und Semantik jederzeit durch neue Module ergänzt werden kann. Durch dieses modulare Konzept lässt sich die Funktionalität eines Systems, wie auch die DSL, sehr einfach an die sich ständig verändernden Bedürfnisse der Domänenspezialisten anpassen.

Jürg Luthiger, Markus Knecht | juerg.luthiger@fhnw.ch

Für die Überwachung der Zulieferer und deren Produkte wird im Forschungsprojekt APPRIS (Advanced Procurement Performance and Risk Indicator System) ein neuartiges Überwachungssystem entwickelt. Die Überwachung geschieht auf der Basis einer frei definierbaren Regelmenge. Wird eine Regel verletzt, so generiert das System ein Warnsignal und macht die Benutzerin auf eine mögliche Störung in einer der überwachten Lieferketten aufmerksam, damit sie frühzeitig Gegenmassnahmen einleiten kann. Die bei der Überwachung verwendeten Regeln müssen Informationen aus unterschiedlichen Datenquellen nutzen, wie zum Beispiel ERP-Systeme, Web-Services verschiedener Anbieter oder unstrukturierte Inhalte aus dem Internet.

Problemstellung

Die Definition von Regelmengen ist eine domänenspezifische Aufgabe. Beispielsweise ist es das Team aus dem Einkauf, welches die Zusammenhänge in der Lieferkette am besten kennt. Deshalb braucht es für dieses Team eine adäquate Schnittstelle zum Monitoring-Tool, über welche die Teammitglieder einfach und effektiv die Regeln des Überwachungssystems festlegen können. Oft wird für eine solche Aufgabe eine formale Sprache eingesetzt, welche speziell für ein bestimmtes Problemfeld (die Domäne) entworfen und implementiert wird. Eine solche Sprache wird als *Domain-Specific Language* (DSL) bezeichnet. Beim Entwurf einer DSL wird man bemüht sein, einen hohen Grad an Problemspezifität zu erreichen. Dadurch ist die DSL durch Domänenspezialisten ohne besonderes Zusatzwissen einsetzbar.

Mit dem ständigen Einsatz eines Überwachungssystems wachsen auch die Ansprüche der Benutzerinnen und Benutzer daran. Daher muss es leicht möglich sein, die Regelmengen ständig den Bedürfnissen anzupassen und dabei auch neue Datenquellen zu erschliessen.

Die Anforderungen in APPRIS gehen über die üblichen Anforderungen an DSLs heraus. So müssen in unserem Fall die einzelnen Module des Überwachungssystems über die DSL kontrolliert und konfiguriert werden können und jedes Modul kann wieder neue Funktionen einführen, die wiederum über die DSL genutzt werden sollen. Um ein derartiges Verhalten zu ermöglichen, muss die DSL möglichst flexibel erweitert[1] werden können. Jedes Modul, das neue Funktionen einbringt, muss in der Lage sein, die DSL so zu erweitern, dass die neuen Funktionen auch über die DSL nutzbar werden. Die DSL ist somit modular, d.h. sie wird aus unterschiedlichen Modulen zusammengesetzt. Herkömmliche DSL-Ansätze müssen demnach um eine Modularisierungsmöglichkeit erweitert werden.

Beispiel einer DSL

In einer DSL kann zum Beispiel eine Regel zur Überwachung des Silberpreises konfiguriert werden (siehe Listing 1). Es wird eine Warnung ausgegeben, falls der Silberpreis über 1.15 Schweizer Franken steigt.

```
when price > 1.15 CHF then
produce warning
```

Listing 1: Beispiel einer einfachen Regel zur Überwachung eines Preises

Soll nun diese Regel erweitert werden, um den Silberpreis mit dem Lagerbestand eines Lieferanten zu kombinieren, so kann die Regel wie in Listing 2 ergänzt werden. Damit kann ein maximales Limit überwacht werden.

```
when price * stock > 10000 CHF
then produce warning
```

Listing 2: Beispiel einer erweiterten Regel zur Überwachung eines Lagerbestandes

Diese Erweiterung führt dazu, dass eine neue Datenquelle integriert werden muss, welche bei-

1 In APPRIS ist es nicht erforderlich, dass die Erweiterung zur Laufzeit erfolgen muss, im Gegensatz zu einem Vorgängerprojekt [Kne12].

spielsweise das ERP-System eines amerikanischen Silberlieferanten nutzt, um den aktuellen Silberlagerstand abzufragen. In Kombination mit dem Silberpreis kann so der aktuelle Wert des Silberlagers des Lieferanten ermittelt werden.

Dieses Beispiel zeigt, wie durch Hinzunahme eines neuen Moduls die vorhandene DSL erweitert wird. Das neue Modul bietet neue Eigenschaften an (stock), fügt eventuell neue Regeln hinzu und greift allenfalls auf neue Datenquellen zu (ERP-System des Silberlieferanten).

Implementierung einer DSL

Es gibt es zwei grundlegend unterschiedliche DSL-Typen: die interne und die externe DSL. Eine externe DSL ist eine formale Sprache, die analog zu einer generellen Programmiersprache (*General Purpose Language*) einen eigenen Übersetzer (Compiler) oder Interpreter besitzt, welcher den DSL-Ausdruck in eine ausführbare Anweisung überführt oder ihn direkt interpretiert.

Bei einer internen DSL hingegen wird der Compiler oder Interpreter einer bestehenden Programmiersprache verwendet, der sogenannten Host-Sprache. In dieser Host-Sprache wird eine Engine für die DSL entwickelt, welche die Möglichkeiten der Hostsprache nutzt, um eine Umgebung zu schaffen, in der jeder DSL-Ausdruck auch zu einem gültigen Ausdruck in der Host-Sprache wird. Nicht alle Host-Sprachen sind dafür geeignet. Hervorragende Kandidaten sind Sprachen wie Scala, Groovy und Ruby, da diese eine Syntax haben die sehr flexibel ist, wodurch der Charakter der Host-Sprache nicht oder nur eingeschränkt in einem DSL Ausdruck zu erkennen ist.

In APPRIS haben wir uns für die Implementierung einer DSL-Engine in der Host-Sprache Groovy entschieden, weil interne DSLs in einem modularen Umfeld gegenüber externen DSLs einige Vorteile bieten:

- Viele Host-Sprachen haben Sprach-Features, wie zum Beispiel Interfaces und Erweiterbare Klassen, welche ein modulares Konzept effizient unterstützen.
- Ein Modulentwickler muss nur die Host-Sprache sowie die Funktionsweise der DSL-Engine verstehen und nicht eine ganz neue Compiler-Infrastruktur.
- Es existieren diverse Frameworks, wie zum Beispiel OSGi (*Open Services Gateway initiative*), die es erlauben, Anwendungen und ihre Dienste per Komponentenmodell zu modularisieren und zu verwalten.

Groovy ist eine moderne, dynamische Sprache für die Java Virtual Machine (JVM). Groovy bietet viele Techniken, die zur DSL-Entwicklung verwendet werden können und die mit relativ geringem Aufwand in eine modulare Umgebung übertragbar sind. Da Groovy auf der JVM läuft und eine sehr gute Java-Integration hat, kann man das umfangreiche Java Ökosystem problemlos nutzen. Dies erlaubt eine Implementierung der DSL-Engine, welche nicht nur Groovy-, sondern zum Beispiel auch Java-Komponenten enthält.

Modulare DSL-Engine

Die Kernstücke unserer DSL-Engine sind die Groovy Shell (ein Groovy-Interpreter) und das OSGi-Framework. Über das OSGi-Framework werden Module geladen, welche Dienste zur Verfügung stellen, um die DSL zu erweitern. Die Engine baut aus diesen Diensten eine Umgebung auf, in deren Kontext DSL-Statements ausgeführt werden können.

Anhand des Groovy Sprach-Features *Category* zeigen wir, wie entsprechende Groovy Sprach-Features auch in einer modularen Umgebung eingesetzt werden können. Eine Kategorie ist eine Klasse mit statischen Methoden. Diese stellen *Extension-Methods* dar, d.h. sie erweitern eine bestehende Klasse um diese pseudo-statischen Methoden. Eine Kategorie, die zum Beispiel alle Integer-Instanzen um die Methode *printWithUnit()* erweitert, ist in Listing 3 aufgeführt. Der erste Parameter einer solchen Methode wird jeweils als Receiver-Objekt interpretiert (siehe *self* in Listing 3).

```
class IntegerCategory {
  static printWithUnit(Integer self,
          String unit) {
    println("""${self} ${unit}""")
  }
}
```

Listing 3: Beispiel einer Groovy Category

Eine Kategorie kann für einen Codeblock mit Hilfe der *use()*-Methode aktiviert werden. Die Groovy-Laufzeitumgebung sorgt dafür, dass die statischen Methoden der aktiven Kategorien beim dynamischen Binden miteinbezogen werden. In Listing 4 wird gezeigt wie die Kategorie aus Listing 3 verwendet wird.

```
use(IntegerCategory) {
  5.printWithUnit "Meter"
}
```

Listing 4: Verwendung der *IntegerCategory* führt zur Ausgabe «5 Meter»

Bei einer Erweiterung der DSL kann zum Beispiel ein neues Modul einen OSGi-Dienst zur Verfügung stellen. Das Modul wird eine entsprechende Erweiterungsschnittstelle, d.h. ein Interface zur Erweiterung der DSL, beinhalten müssen. Auf diesem Interface gibt es eine Methode *getCategories()*, die eine Liste aller *Category*-Klassen des Moduls zurückgibt. Die DSL-Engine selber aktiviert nun alle Kategorien von jeder DSL-Erweiterung. So hat die DSL-Engine über die aktivierten Kategorien Zugang zu neuer Funktionalität, welche in den *Extension-Methods* implementiert ist. Dies erlaubt

es jederzeit neue Methoden zu definieren, zu implementieren und zu integrieren, welche dann aus der DSL heraus angesprochen werden können.

Die Erweiterungsschnittstelle definiert weitere Methoden, die es erlauben andere Groovy Sprach-Features zu verwenden, die in einem modularen DSL-Umfeld wichtig sind [Dae10], wie zum Beispiel: *Metaclass*, *Builders* und *Expandos*. Die DSL-Erweiterungsschnittstelle ist eine herkömmliche Java-Schnittstelle, weshalb sie nicht nur mit Groovy-Komponenten verwendet werden kann, sondern zum Beispiel auch mit normalen Java-Komponenten.

Praxisbeispiel APPRIS

Kehren wir zu unserem eingangs gezeigten Beispiel mit dem Silberpreis zurück. Um die Regel aus Listing 1 ausführen zu können, müssen folgende OSGi-Module existieren:

1. Ein Basismodul, welches die DSL-Struktur und die Schlüsselwörter „when“, „>“, „then“, „produce“ und „warning“ zur Verfügung stellt.
2. Ein Price-Modul, welches die Methode „price“ zur Verfügung stellt und Abfragen auf unterschiedlichen Datenquellen zur Preisermittlung durchführt.
3. Ein Money-Modul, welches Währungen verwalten und Umrechnungen zwischen den Währungen durchführen kann und welches auch den Ausdruck „CHF“ bereitstellt.

Um die Regel aus Listing 2 aufsetzen zu können, muss die bestehende DSL um ein neues Modul ergänzt werden. Mit dem neuen Modul sollen die Lagerbestände eines Lieferanten miteinbezogen werden können, vorausgesetzt eine entsprechende Datenquelle ist angebunden. Die OSGi-Modullandschaft für diese erweiterte DSL ist in Abbildung 1 dargestellt. Wir sehen, dass der Funktionsumfang der DSL aus unterschiedlichen Modulen kommt. Es gibt dabei Module, welche die DSL direkt erweitern und solche, welchen bestimmten Modulen neuen Datenquellen hinzufügen. Die Module können auch Abhängigkeiten untereinander haben. Das Price-Modul ist zum Beispiel vom Money-Module abhängig. Dadurch können Preise in der korrekten Währung ausgegeben werden.

Implementierung einer DSL-Erweiterung

Anhand des Stock-Modules wollen wir nun zeigen, wie eine DSL-Erweiterung aussehen könnte. Das Money-Module führt den Multiplikationsoperator „*“ ein, um den Preis einer Einheit (z.B. Gramm) mit der Anzahl Einheiten multiplizieren zu können. Damit sich jedoch der Ausdruck „price * stock“ verarbeiten lässt, muss zudem eine Eigenschaft „stock“ vorhanden sein, welche die Lagermenge (in Gramm) für einen Lieferanten und ein Produkt ausfindig macht. Diese Lagermenge wird über einen Dienst eines OSGi-Modul geliefert. Damit kann das Stock-Modul die entsprechenden

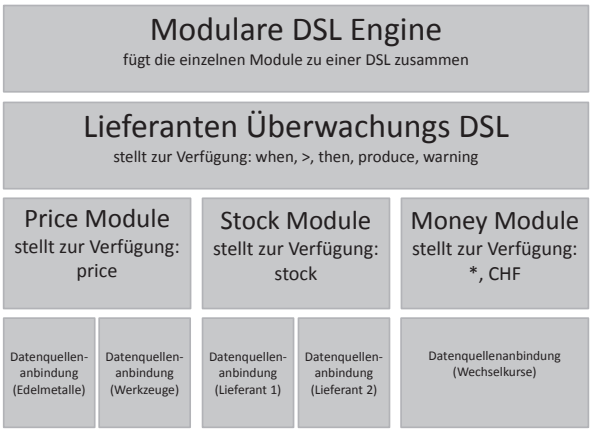


Abbildung 1: Module für den DSL-Ausdruck in Listing 2

Daten durch die „stock“-Eigenschaft der DSL zur Verfügung stellen.

Die DSL-Engine führt alle Ausdrücke innerhalb einer Instanz einer Klasse namens Global aus. Das heisst, dass die *this*-Referenz während der DSL-Ausführung vom Typ Global ist. In Groovy wird der Ausdruck „stock“ als *this.getStock()* interpretiert. Wir müssen also sicherstellen, dass die Methode *getStock()* auf der *this*-Instanz vorhanden ist. Dies kann durch eine Kategorie erreicht werden, welche die Global-Klasse um die Methode *getStock()* erweitert (siehe Listing 5).

```
class StockCategory {
  static getStock(Global self) {
    return …
  }
}
```

Listing 5: Implementierung der Kategorie „Stock“

Die neue Kategorie muss anschliessend in der DSL-Engine registriert werden. Dazu wird das *IDslExtensionInterface* implementiert. Für unser Beispiel ist hier nur die Methode *getCategories()* von Interesse (siehe Listing 6).

```
class StockExtension implements
          IDslExtensionInterface {
  …
  public List<Class> getCategories() {
    return [StockCategory]
  }
  …
}
```

Listing 6: Implementierung der Stock-Extension über die Erweiterungsschnittstelle

Damit die Erweiterung als OSGi-Dienst verwendet werden kann, braucht es noch eine Registrierung, die je nach verwendetem OSGi-Framework und der gewählten Technik unterschiedlich aussehen kann.

Wird nun ein DSL-Ausdruck ausgeführt, holt sich die DSL-Engine alle *IDslExtensionInterface*-Dienste, sammelt alle Kategorien und aktiviert sie. Dann wird der Ausdruck interpretiert und dabei wird die Eigenschaft „stock“ ausgeführt, wo-

bei das dynamische Binden nach einer Methode namens *getStock()* auf der *this*-Instanz sucht und unsere *Category*-Methode findet und ausführt.

Fazit und Ausblick

Die vorgestellte DSL-Engine erlaubt es eine DSL zu entwickeln, die im Einsatz wachsen kann. Groovy hat sich als eine sehr gute Hostsprache für eine modulare DSL herausgestellt. Dank der dynamischen Natur von Groovy können einzelne DSL-Module unabhängig voneinander kompiliert und in die DSL-Engine integriert werden. Die Sprach-Features, die in Groovy für eine DSL verwendet werden können, sind grösstenteils leichtgewichtig und bieten trotzdem viele Einsatzmöglichkeiten.

Unsere DSL-Engine unterstützt die Möglichkeit, eine DSL zu definieren, die je nach Einsatzgebiet aus unterschiedlichen Modulen besteht. Eine sinnvolle Erweiterung unserer DSL-Engine wäre beispielsweise ein Rollenkonzept, welches es erlauben würde, je nach Rolle andere DSL-Funktionalitäten zur Verfügung zu stellen. Jede Rolle nutzt dabei eine andere Menge von Modulen. Darauf basierend könnte auch ein Sicherheitskonzept realisiert werden, welches sensitive Daten und Funktionen nur bestimmten Rollen zur Verfügung stellt.

Referenzen

[Dae10] Daerle, F.: Groovy for Domain-Specific Languages. Packt Publishing, 2010.

[Kne12] Knecht, M.: Modulare DSL zur Lieferantenüberwachung. Projektbericht, 2012. http://www.fhnw.ch/personen/markus-knecht/dateien/Modulare%20DSL%20zur%20Lieferantenüberwachung.pdf

Badminton-Smash Geschwindigkeitsmessung

Im Gegensatz zu internationalen Tennisturnieren, wo die Ballgeschwindigkeitsmessung bei Aufschlägen seit vielen Jahren zum Standard gehört, gibt es bei entsprechenden Badmintonturnieren keine vergleichbaren Messungen. Solche Geschwindigkeitsmessungen sind primär als Bereicherung für die Zuschauer gedacht, können darüber hinaus aber auch den Spielern, Trainern und Ausrüstern wichtige Informationen liefern. Wir beschreiben in diesem Artikel unseren Ansatz zur Geschwindigkeitsmessung von Schmetterbällen in Badminton und berichten über erste Erfahrungen mit unserem Messsystem an den Badminton Swiss Open 2012 in Basel.

Christoph Stamm | christoph.stamm@fhnw.ch

Grosse, internationale Badmintonturniere ziehen weite Bevölkerungsgruppen an und zeigen den sehr schnellen, dynamischen Sport von seiner eindrücklichsten Seite. In Basel finden seit mehr als 20 Jahren die Badminton Swiss Open[1] statt, welche zwischenzeitlich zu den allerbesten, internationalen Badminton Turnieren weltweit gehörten. In den Jahren 2008 und 2009, beispielsweise, berichteten 25 Fernsehstationen mit 250 Mio. potentiellen Haushalten von diesem Grossanlass. Die Raffinesse und erforderliche Leistung in diesem Sport erschliesst sich jedoch oft nur unzureichend breiteren Zuschauerkreisen, vor allem dann, wenn schnelle, hochpräzise aber weniger kämpferische Ballwechsel ein Spiel dominieren. Hier sind die Turnierveranstalter gefordert, die eindrücklichen Bilder mit kompetenten Moderationen zu versehen und die kurzen Pausen zwischen Ballwechseln mit attraktivitätssteigernden Zusatzinformationen, wie zum Beispiel Superzeitlupenaufnahmen von Hochgeschwindigkeitskameras oder Geschwindigkeitsangaben bei Schmetterbällen zu nutzen.

Bei internationalen Tennisturnieren gehört die Geschwindigkeitsmessung von Aufschlägen seit vielen Jahren zum Standard. Daneben haben sich weitere Messverfahren etabliert, wie zum Beispiel die Überwachung der Netzkante bei Aufschlägen oder Hawk-Eye[2], welche nicht nur der Erhöhung der Attraktivität für die Zuschauer dienen, sondern darüber hinaus dem Schiedsrichter helfen, unklare Situationen zu klären. Hawk-Eye basiert auf der Verwendung von mindestens vier Hochgeschwindigkeitskameras, die das Spielfeld aus verschiedenen Blickwinkeln heraus erfassen und somit die Flugbahn des Balls präzis festhalten können.

Im Gegensatz zu Tennis ist der Bedarf nach automatisierten Messinstrumenten in Badminton weniger gross, da die Ballgeschwindigkeit beim Spielfeldrand deutlich geringer ist als bei Aufschlägen im Tennis und die Linienrichter zudem noch näher beim Spielfeldrand sitzen. Daher kommt es viel seltener zu umstrittenen Situationen, was der primäre Grund ist, dass elektronische bzw. computerbasierte Messsysteme noch keinen Einzug bei internationalen Badmintonturnieren gehalten haben. Trotzdem sind vereinzelte Turnierdirektoren an solchen Messsystemen interessiert, jedoch primär zur Attraktivitätssteigerung für die Zuschauer.

Anforderungen

Im vorliegenden Artikel befassen wir uns mit der Geschwindigkeitsmessung von Schmetterbällen im laufenden Spielbetrieb eines Badmintonturniers[3]. Eine solche Geschwindigkeitsmessung darf aus naheliegenden Gründen das Spiel in keinerlei Hinsicht beeinträchtigen, muss beinahe in Echtzeit (< 2 Sekunden) ausgeführt werden können und sollte mit einem Minimum an Geräten auskommen, um den Aufwand für die Turnierausrichter gering zu halten. Die quasi Echtzeitanforderung rührt daher, dass ein solches Messsystem der Attraktivitätssteigerung für die Zuschauer dient und demzufolge eine möglichst zweifelsfreie Zuordnung der Messresultate zu den Schmetterbällen möglich sein sollte, auch dann, wenn viele Smashs in kürzester Zeit hintereinander erfolgen. Für Messsysteme, die hauptsächlich der Weiterentwicklung der Schläger dienen und bei den Schlägerherstellern eingesetzt werden, gelten selbstverständlich wieder andere Anforderungen.

Lösungsansätze

Die Messung der Federballgeschwindigkeit (shuttlecock speed) eines Schmetterballes (Smashs) in einem laufenden Badmintonspiel ist mit verschiedenen Schwierigkeiten konfrontiert: Schmetter-

1 Badminton Swiss Open: http://www.swissopen.com

2 Hawk-Eye: http://www.hawkeyeinnovations.co.uk/

3 Das Projekt „Badminton-Smash Geschwindigkeitsmessung“ ist von der Haslerstiftung finanziell gefördert worden: http://www.haslerstiftung.ch

bälle können fast von überall, in unterschiedliche Richtungen und in sehr verschiedenen Spielsituationen gespielt werden. Die im Strassenverkehr und Tennis angewandte Radartechnik zur Messung der Fahrzeug- bzw. Aufschlagsgeschwindigkeit erfolgt in Bewegungsrichtung. In beiden Situationen können die bewegten Objekte recht gut angepeilt werden, weil sie eine ausreichende Grösse und vor allem eine bekannte Position und Bewegungsrichtung haben. In einem laufenden Badmintonspiel sind weder die Schlagposition noch die Flugrichtung im Voraus bekannt. Zudem bewegt sich der kleine, filigrane und nur fünf Gramm leichte Federball[4] kurz nach dem Schlag nicht nur in der neuen Flugrichtung, sondern muss darüber hinaus noch sein konisches Federhinterteil durch eine extrem schnelle transversale Rotation der neuen Flugrichtung angleichen. Diese Angleichung ist als rasch abklingende transversale Wellenbewegung in Abbildung 1 ersichtlich.

Auf dem ersten Meter nach dem Schlag beträgt die durchschnittliche Fluggeschwindigkeit des Federballs rasch 100 km/h und mehr. Bei Schmetterbällen von Spitzenspielern sind Geschwindigkeiten von 250 km/h keine Seltenheit. Der aktuelle, inoffizielle Geschwindigkeitsrekord stammt von einem chinesischen Badmintonspieler[5] aus dem Jahr 2005 und liegt bei 332 km/h. Im Vergleich dazu beträgt der aktuelle, offizielle Geschwindigkeitsrekord im Tennis[6] aus dem Jahr 2012 263 km/h. In Testlabors von Badminton-Racket-Herstellern (Yonex) sind anscheinend sogar schon 421 km/h gemessen worden.

Aus diesen genannten Gründen ist eine spontane, präzise Radarpeilung aus gut 15 Metern Abstand in einem laufenden Badmintonspiel impraktikabel[7].

Als Alternative zur Radarpeilung bietet sich ein optisches Messverfahren auf Basis von Videoaufnahmen an. Heutige, digitale Hochgeschwindigkeitskameras sind in der Lage 1000 Bilder pro Sekunde bei VGA-Videobildgrössen zu produzieren und können einerseits bei kleineren Auflösungen sogar mehrere Zehntausend Bilder und anderseits bei Bildraten von 500 Bilder/s und weniger auch grössere Auflösungen aufnehmen. Dadurch lassen sich sehr schnelle Bewegungen filmen und nachträglich auswerten. Neben der hohen Bildrate ist aber auch eine sehr kurze Belichtungszeit notwendig, um Bewegungsunschärfe zu vermeiden, welche die Distanzmessung und somit die Geschwindigkeitsmessung unnötig erschweren würden. Kurze Belichtungszeiten in der Grösse von 0.1 ms sind mit lichtstarken Objektiven und guter Badmintonfeldausleuchtung infolge der weissen Federbälle vor dem dunklen Hallenhintergrund durchaus realisierbar.

Für die Auswertung der Videosequenzen kann ein handelsüblicher, leistungsfähiger Rechner eingesetzt werden, welcher über schnelle Datenleitungen mit dem Kamerasystem verbunden wird. Die Echtzeitanforderung an das Gesamtsystem kann jedoch nur dann erfüllt werden, wenn es gelingt, die riesigen Datenmengen einer Hochgeschwindigkeitskamera in Echtzeit an das Auswertungssystem zu übertragen. Die meisten Hochgeschwindigkeitskameras sind bei Bildraten von 500 und mehr dazu aber nicht mehr in der Lage. Stattdessen speichern sie eine kurze Videosequenz fortwährend in einem kameraeigenen Ringpuffer und übertragen nach einem (manuell) ausgelösten Trigger einen ersten vordefinierten Anteil des Ringpuffers vor dem Triggersignal und einen zweiten Anteil nach dem Triggersignal an den Zielrechner. Eine solche Übertragung kann dann schnell mehrere Sekunden dauern und somit eine Echtzeitauswertung verunmöglichen.

Mittlerweile gibt es auch digitale Videokameras mit zusätzlicher CPU und Schnittstellen, so dass das ganze Kamerasystem als Standard-PC unter Windows oder Linux verwendet werden kann. Ein solch integriertes System löst zwar das Problem der Datenübertragung, enthält aber oft eine leistungsschwache CPU und ist nicht für hohe Bildraten ausgelegt. Einen anderen Ansatz wählen oft Hersteller von Hochgeschwindigkeitskameras. Sie bauen in ihre Kameras einen digitalen Signalprozessor (DSP) ein, um gewisse Bildaufbereitungsaufgaben direkt in der Kamera durchzuführen. Je nach Kameraarchitektur könnte ein solcher DSP auch für erste Bildauswertungsaufgaben eingesetzt werden, um beispielsweise eine inhaltsgesteuerte Sequenzauslösung und -übertragung zu starten.

Optisches Messsystem

Für eine präzise Geschwindigkeitsmessung ist die Rekonstruktion des dreidimensionalen Flugwegs des Federballs anhand der gemachten Videoaufnahmen notwendig. Aus dem dreidimensionalen Flugweg lässt sich dann die Flugdistanz und aus der Bildrate der Videokamera die verstrichene Zeit und insgesamt die Durchschnittsgeschwindigkeit bestimmen. Wie schon angedeutet, wird für die Geschwindigkeitsmessung eines Smashs nicht die Durchschnittsgeschwindigkeit über die gesamte Flugbahn, sondern lediglich über den ersten Meter nach dem Schlag verwendet. Einerseits vereinfacht dies den ganzen Vorgang, weil nur ein kleiner Ausschnitt der gesamten Flugbahn rekonstruiert werden muss. Anderseits sagt der gemessene Geschwindigkeitswert wenig über die

4 An internationalen Badmintonturnieren kommen nur Naturfederbälle mit einem Kopf aus Kork und echten Federn zum Einsatz. Daher beziehen sich alle hier gemachten Angaben auf Naturfederbälle.

5 Fu Haifeng: http://de.wikipedia.org/wiki/Fu_Haifeng

6 ATP: http://www.atpworldtour.com/News/Tennis/2012/05/Features/Groth-Fast-Serve.aspx

7 Das Spielfeld ist 13.4 m lang. Es muss mit einem Abstand von ca. drei Metern zum Spielfeldrand gerechnet werden, um die Spieler und die Linienrichter nicht zu stören.

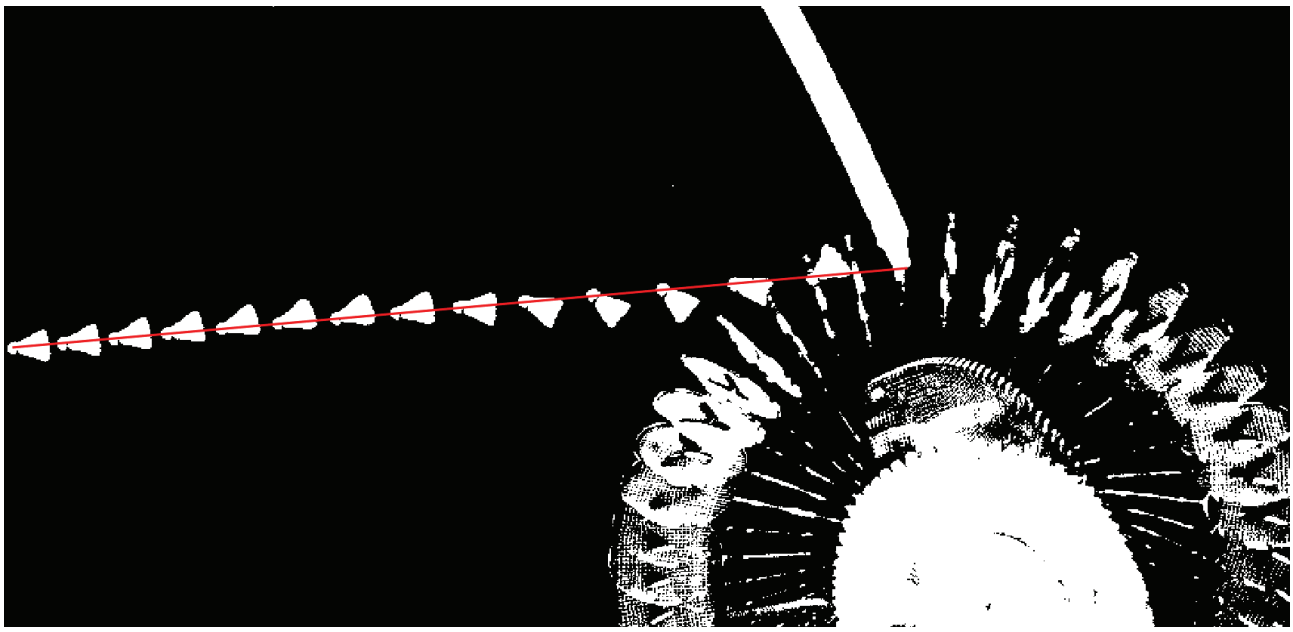


Abbildung 1: Lineare Flugbahn eines langsamen Smashs kurz nach dem Schlag

Durchschnittsgeschwindigkeit des vollständigen Smashs beginnend beim Treffpunkt und endend beim Berühren des Bodens oder beim Treffpunkt des Rückschlags aus. Vereinfacht man die tatsächliche Flugbahn des ersten Meters zu einer linearen[8], so kann auf eine Rekonstruktion der tatsächlichen Flugbahn verzichtet werden und nur noch der Start- und Endpunkt der Messstrecke sind relevant. Auf alle Fälle bleibt aber die Rekonstruktion des genauen Treffpunkts im Raum, dort wo der Federball das Racket des schlagenden Spielers verlässt, und des Endpunkts der Messstrecke.

Die Photogrammetrie befasst sich mit der Bestimmung der räumlichen Lage eines Objektes aus Fotografien oder Messbildern. Dazu wird das Objekt von verschiedenen bekannten Standorten mit einer Kamera aufgenommen oder es werden gleichzeitig mehrere Kameras bzw. Stereokameras eingesetzt. Der Bedarf an mehrere Aufnahmen desselben unbekannten Objektes aus verschiedenen Positionen zur Bestimmung seiner räumlichen Lage hängt mit der zweidimensionalen Projektion des Kamerabildes zusammen. Für die Positionsbestimmung eines Federballs bei einem Smash bieten sich vor allem zwei Kamerapositionen an: eine Kamera an der Seitenlinie und eine Kamera oberhalb des Spielfelds. Während eine Kamera an der Seitenlinie gänzlich ausserhalb des Spielfelds und mit wenig Aufwand platziert werden kann, ist eine Kamerapositionierung oberhalb des Spielfeldes schon einiges aufwendiger, weil sie mindestens sieben Meter über dem Spielfeld platziert werden muss, um kein Störfaktor zu sein. Damit einhergehen lange Kabelverbindungen zum Rechner, welcher die Kamerabilder erfassen, eventuell zusammenführen und auswerten soll. Beide Kamerapositionen eignen sich jedoch gut, die Flugbahn des Federballs (oder zumindest eine räumliche Projektion davon) festzuhalten.

Sobald mehrere Kameras gleichzeitig zum Einsatz kommen, stellt sich bei dynamischen Objekten das Synchronisationsproblem. Je schneller sich die zu fotografierenden Objekte bewegen, desto wichtiger ist eine genaue zeitliche Synchronisation der Aufnahmen. Nur dann lassen sich aus den einzelnen Aufnahmen genaue, zeitabhängige Ortsangaben zu den Objekten machen. Bei Hochgeschwindigkeitskameras wird die Synchronisation mehrerer Kameras zum echten Problem und wird wenn immer möglich vermieden.

Die notwendige Bildrate bei der Aufnahme eines Objektes, welche sich durch den Bildraum der Kamera hindurch bewegt, ergibt sich aus der Bewegungsgeschwindigkeit und der gewünschten Anzahl Bilder, auf denen das Objekt abgebildet sein soll. Im Fall eines Badminton-Smashs müssen wir von einer maximalen Geschwindigkeit (ca. 400 km/h) und von mindestens drei Aufnahmen ausgehen, welche den Federball auf seinem ersten Meter nach dem Schlag darstellen. Zwei Aufnahmen genügen in der Praxis kaum, weil beim eigentlichen Treffpunkt der Federball quasi im Racket verschwindet und infolge Verformung des Federballs und des Rackets kaum sichtbar ist. Unter diesen Annahmen resultiert eine Bildrate von mindestens 333 Bilder/s, welche auf die nächst höhere und verfügbare Bildrate der Hochgeschwindigkeitskamera (z.B. 500 Bilder/s) angehoben wird.

Swiss Open 2011

Im März 2011 filmten wir in Absprache mit den Verantwortlichen der Badminton Swiss Open im Rahmen eines Studentenprojekts mit einer einzigen Hochgeschwindigkeitskamera der Firma

8 Für Schmetterbälle ist diese Annahme hinreichend genau, wie das Beispiel in Abbildung 1 exemplarisch zeigt.

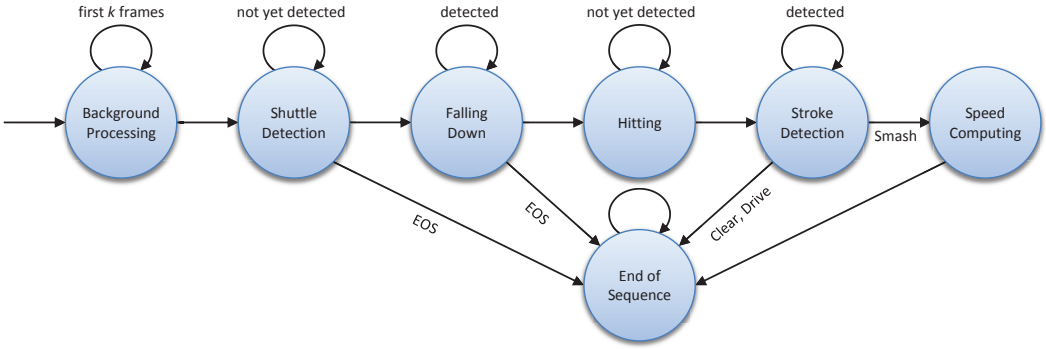


Abbildung 2: Zustandsmaschine zur Shuttle-Detektion und Geschwindigkeitsmessung

AOS Technologies AG[9] in den Finalspielen eine sehr grosse Anzahl von Smashs mit unterschiedlichsten Kameraeinstellungen. Die Kamera wurde in gebührendem Abstand zum Spielfeld an dessen Seitenlinie aufgestellt. Bei der genauen Positionierung achteten wir darauf, dass (Sprung-) Smashs von verschieden grossen Spielern und Spielerinnen in genügend hoher Auflösung und dennoch ausreichend langer Flugbahn erfasst werden konnten. Die daraus entstandenen Videoaufnahmen[10] dienten danach als realistische Testdaten zur Entwicklung einer ersten Software zur Smash-Geschwindigkeitsmessung.

An früherer Stelle haben wir ausgeführt, dass in der Photogrammetrie mehrere Aufnahmen des gleichen Objekts aus verschiedenen Blickwinkeln verwendet werden, um die räumliche Lage des Objekts zu bestimmen. In unserem Fall haben wir jedoch nur eine einzige Kamera zur Verfügung und müssen daher die fehlende Tiefeninformation im Kamerabild aus dem Abbildungsmassstab der Kamera und der bekannten Objektgrösse des Federballs abschätzen. Hierin liegt jedoch eine grosse Ungenauigkeit, welche von der Kameraauflösung, der Kameraoptik und der Distanz zwischen Kamera und Objekt abhängt. Gemäss einer Fehlerabschätzung liegt der Messfehler bei ca. 6%, solange die Flugbahn parallel zur Bildebene liegt. Sobald die Flugbahn diese Ebene verlässt, was bei Cross-Smashs häufig vorkommt, steigt der Messfehler zusätzlich noch an. Daher sind alternative Lösungen zur Distanzmessung zwischen Kamera und Federball gesucht.

Bereits in diesem Vorprojekt ist eine erste einfache Version einer Auswertungssoftware entstanden. Diese Software ist in der Lage, die Geschwindigkeit eines Smashs parallel zur Bildebene und offline (also nicht in Echtzeit) mit einem gut definierten Messfehler zu bestimmen. Der dabei eingesetzte Algorithmus ist in C++ unter Verwendung von OpenCV[11] entwickelt worden, verwendet verschiedene Arten von Mustererkennung und entspricht im Wesentlichen der Zustandsmaschine in Abbildung 2.

Echtzeittauglichkeit

Fürs Hauptprojekt haben wir uns zwei primäre Ziele gesetzt: erstens die Echtzeittauglichkeit zu verbessern und zweitens eine Anpassung der Geschwindigkeitsberechnung bei Flugbahnen, welche nicht parallel zur Bildebene verlaufen.

Betrachten wir zuerst die Echtzeittauglichkeit. Der naheliegendste Ansatz die Echtzeittauglichkeit zu erhöhen, wäre der Einsatz einer Hochgeschwindigkeitskamera, welche in der Lage ist, die aufgenommenen Videobilder in Echtzeit an einen angeschlossenen Rechner zu übertragen und dort zu verarbeiten. Verschiedene solche Streaming-Systeme sind auf dem Markt und auch unser Projektpartner AOS Technologies AG bietet mit PROMON[12] ein entsprechendes System an [SS11]. Da die meisten dieser Highspeed-Streaming Systeme jedoch nicht die Anforderung erfüllen, bei einer Mindestgrösse von 1000×600 eine Bildrate von 333 und mehr zu liefern, was einer Übertragungsrate von 1.6 GBit/s entspräche, verfolgen wir einen anderen Ansatz, nämlich die Reduktion der Datenübertragung von der Kamera zum Rechner.

Wie im Vorprojekt verwenden wir eine Schwarzweisskamera, welche problemlos 500 Bilder pro Sekunde bei einer Auflösung von 1280×600 liefern, diese aber nicht in Echtzeit an einen Rechner übermitteln kann. Da für die Berechnung der Smash-Geschwindigkeit nur ganz wenige Bilder notwendig sind, beginnend beim Treffpunkt des Federballs und endend zum Zeitpunkt, wo der Federball den Bildausschnitt verlässt, besteht die Idee darin, möglichst nur zwanzig relevante Bilder an den Rechner zu übertragen und dadurch die Dauer der Datenübertragung auf ein Minimum zu reduzieren. Ein solcher Ansatz lässt sich jedoch nur dann realisieren, wenn bereits innerhalb der Kamera über die Relevanz der Bilder entschieden oder der Trigger sehr präzise mit dem Schlag ausgelöst werden kann. Bei einer manuellen Auslösung des Triggers lassen sich mit etwas Übung 0.2 Sekunden dauernde Videosequenzen (= 100

9 AOS Technologies AG: http://www.aostechnologies.com
10 Slowmotion Badminton Smashes: http://www.youtube.com/watch?v=haRSzUPdCzg
11 Open Source Computer Vision: http://opencv.org
12 AOS PROMON: http://www.aostechnologies.com/

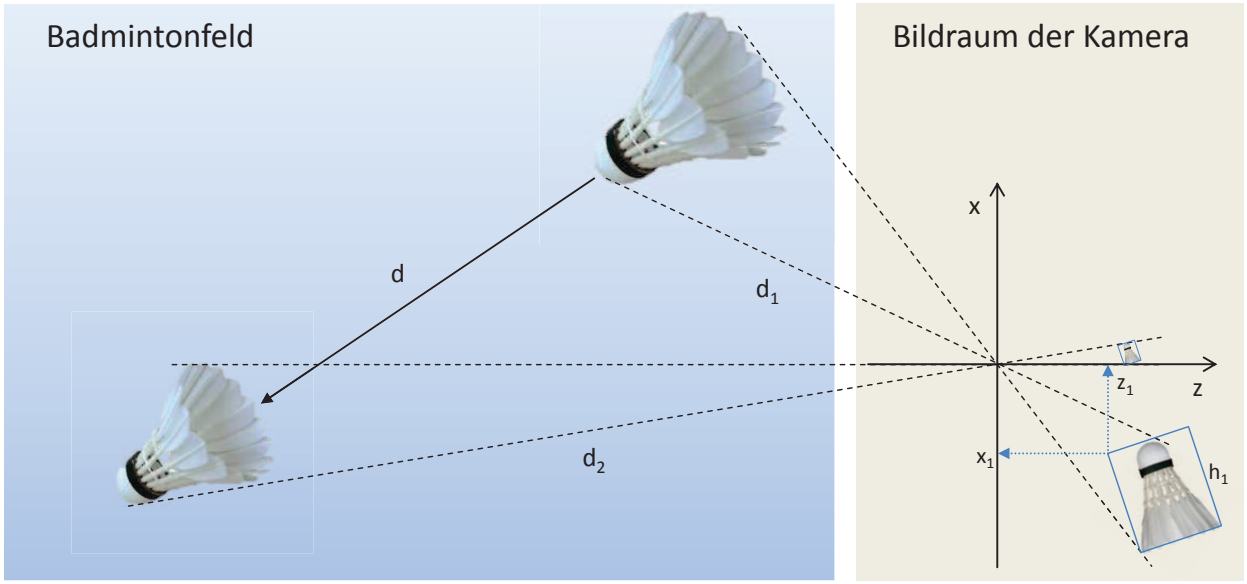


Abbildung 3: Abbildungsgeometrie zur Bestimmung der räumlichen Distanz

Bilder) aufnehmen, welche die relevanten Bilder zur Geschwindigkeitsmessung eines Smashs beinhalten. Ein ganz anderer Ansatz wäre eine akustische Trigger-Auslösung, welche das typische Schlaggeräusch verwenden und dadurch die Genauigkeit gegenüber der manuellen Auslösung erhöhen würde.

Digitaler Signal Prozessor

Da die von uns verwendete Hochgeschwindigkeitskamera über einen eingebauten DSP verfügt, versuchen wir diesen für die Auslösung des Triggers einzusetzen. Dabei werden die an den DSP übertragenen Bilder mit einem speziell optimierten Programm nach einem Federball in einem vordefinierten Bereich abgesucht. Die ursprüngliche Idee, für dieses DSP-Programm den gleichen Algorithmus wie im Vorprojekt zu verwenden, scheitert daran, dass der DSP nur mit einer Bildrate von ca. 30 Bildern/s (nur jedem 17. Bild) versorgt wird und somit der Federball zwischen zwei DSP-Bildern wesentlich grössere Distanzen zurücklegt als dies beim in Abbildung 2 vorgestellten Algorithmus angenommen wird. Im genannten Algorithmus gehen wir davon aus, dass der Federball innerhalb eines schmalen Streifens am oberen Bildrand ins Bild eintritt. Diese Annahme vereinfacht und beschleunigt die Detektion wesentlich, weil der Suchbereich dadurch deutlich eingeschränkt ist. Eine Detektion am oberen Bildrand reduziert auch die Fehleranfälligkeit, weil sich in unteren Bildbereichen die Spielerin und das Racket bewegen und dadurch dem Federball ähnliche Muster auftreten können. Zwischen der ersten Detektion und dem Schlag besteht eine unterschiedlich lange Flugphase (die Länge hängt von der Position und Grösse des Spielers ab), welche mit wenig Aufwand bis zum Schlag überwacht werden kann. Durch die geringe Fallgeschwindigkeit des Federballs (ca. 4 m/s) können innerhalb dieser Flugphase zwischen zehn und mehr als hundert Bilder liegen. Auf den DSP übertragen bedeutet dies, dass zur Federballdetektion vor dem Schlag nur zwischen null und fünf Bilder verwendet werden können. In den Fällen, wo gar keine Bilder verwendet werden können, muss auf eine Auswertung gänzlich verzichtet werden. In den anderen Fällen muss der Suchbereich so stark ausgeweitet werden, dass sowohl der Suchaufwand als auch die Fehleranfälligkeit sich dermassen erhöhen, dass eine zuverlässige Detektion in der für den DSP-Algorithmus zur Verfügung stehenden Ausführungszeit nicht umsetzbar ist.

Es ist uns also nicht gelungen, die Datenmenge zwischen Kamera und Rechner zuverlässig und signifikant zu reduzieren, so dass nur die für die Geschwindigkeitsmessung relevanten Bildsequenzen übertragen werden. Um dennoch in nützlicher Zeit eine Geschwindigkeitsmessung durchführen zu können, bleibt nur die Reduktion der Datenmenge pro Video-Bild und der Einsatz einer Hochgeschwindigkeitskamera mit Echtzeitdatenübertragung. Darauf gehen wir weiter unten noch ein.

Geschwindigkeitsberechnung

Das zweite verfolgte Ziel ist eine verbesserte Geschwindigkeitsmessung bei Federballflugbahnen, welche nicht parallel zur Bildebene der Kamera verlaufen. Bei einer parallelen Flugbahn lässt sich aus der realen Federballgrösse G und seiner Bildgrösse h der Abbildungsmassstab $A = h/G$ bestimmen. Zusammen mit der Bildrate der Kamera und dem zweidimensionalen Bewegungsvektor des Federballs im Bild lässt sich dann die Geschwindigkeit berechnen. Um die räumliche Geschwindigkeit berechnen zu können, bedarf es zusätzlicher Tiefeninformation. In einer Bildsequenz eines Smashs wird dazu im i-ten Bild die räumliche Distanz $d_i = ||(x_i, b, z_i)||/A$ zwischen Kamera

und Federball aus der Bildgrösse h_i des Federballs und den Abbildungseigenschaften der verwendeten Kamera und Optik (Abbildungsmassstab A, Brennweite f, Bildweite b, Auflösung)[13] berechnet (siehe Abb. 3). Aus diesen räumlichen Distanzen lassen sich dann der räumliche Flugweg d und somit auch die räumliche Geschwindigkeit des Shuttles berechnen.

Die Genauigkeit der Distanzberechnung hängt also wesentlich von der Genauigkeit der Grössenmessung h des Shuttles im Bild ab. Infolge der starken Verformungen des Shuttles unmittelbar nach dem Schlag ist eine solche Grössenmessung nicht immer mit gleicher Qualität möglich. Erschwerend kommt hinzu, dass infolge der hohen Shuttle-Geschwindigkeit nur wenige, relevante Einzelbilder nach dem Schlag zur Verfügung stehen und dass aus den daraus gemessenen Shuttle-Grössen möglichst exakt die räumliche Flugbahn interpoliert werden muss. Dabei führen kleinste Messabweichungen rasch zu einem Geschwindigkeitsmessfehler von +/-10%.

Swiss Open 2012

An den diesjährigen Badminton Swiss Open 2012 haben wir unsere neue Software getestet, laufend überarbeitet und in den Finalspielen am Sonntag auch erste Messresultate den Zuschauern präsentiert. Die Geschwindigkeitsmessung ist auf noch verhaltenes Interesse gestossen. Das hängt primär damit zusammen, dass wir das gesteckte Ziel der Echtzeittauglichkeit noch nicht erfüllt haben. Infolge der fehlenden automatischen Bildauslösung in der Kamera haben wir die Kamera manuell ausgelöst und dadurch eine wesentlich grössere Anzahl Bilder für die Übertragung und Analyse in Kauf genommen. Mit etwas Übung sind wir in der Lage, in einer Sequenzlänge von 0.2 Sekunden (100 Bilder) die für die Geschwindigkeitsmessung relevanten Bilder einzufangen. Bis zur Anzeige der Geschwindigkeit sind dann jedoch bis zu 10 Sekunden verstrichen, anstatt der erhofften zwei Sekunden. Dadurch ist die Geschwindigkeitsmessung nur noch bei einem Ballwechsel abschliessenden Smash aussagekräftig einsetzbar.

PROMON und Ausblick

Die vertiefte Auseinandersetzung mit der Problematik hat uns gezeigt, dass wir im Wesentlichen darauf angewiesen sind, dass die von der Kamera erzeugten Bilder in kürzester Zeit beim Analysesystem ankommen. Das bedeutet, dass wir eine Hochgeschwindigkeitsvideokamera benötigen, welche in der Lage ist, die riesige Datenflut bei einer Bildrate von 500 Bildern pro Sekunde in möglichst kurzer Zeit (wenn möglich in Echtzeit) an den PC zu übertragen. Bei einer Bildgrösse von 1280×1024 wäre somit eine Transferrate von 5.24 Gbit/s notwendig. Nur so erhalten wir alle notwendigen Informationen rechtzeitig, die wir für unsere Geschwindigkeitsanalyse im PC benötigen. Transferraten von 10 Gbit/s und mehr sind mit heutigen Interface-Technologien tatsächlich realisierbar. Mit dem Wechsel auf eine Kamera mit entsprechender Streaming-Technologie sollte es somit möglich sein, das Gesamtsystem zu vereinfachen und zu beschleunigen.

Rückschläge	100
Smashs mit Geschwindigkeitsmessung	20
nicht erkannte Smashs	6
andere Rückschläge mit Geschwindigkeitsmessung	4
andere Rückschläge korrekt erkannt	70

Tabelle 1: Auswertung von Badmintonsequenzen in einer Trainingshalle

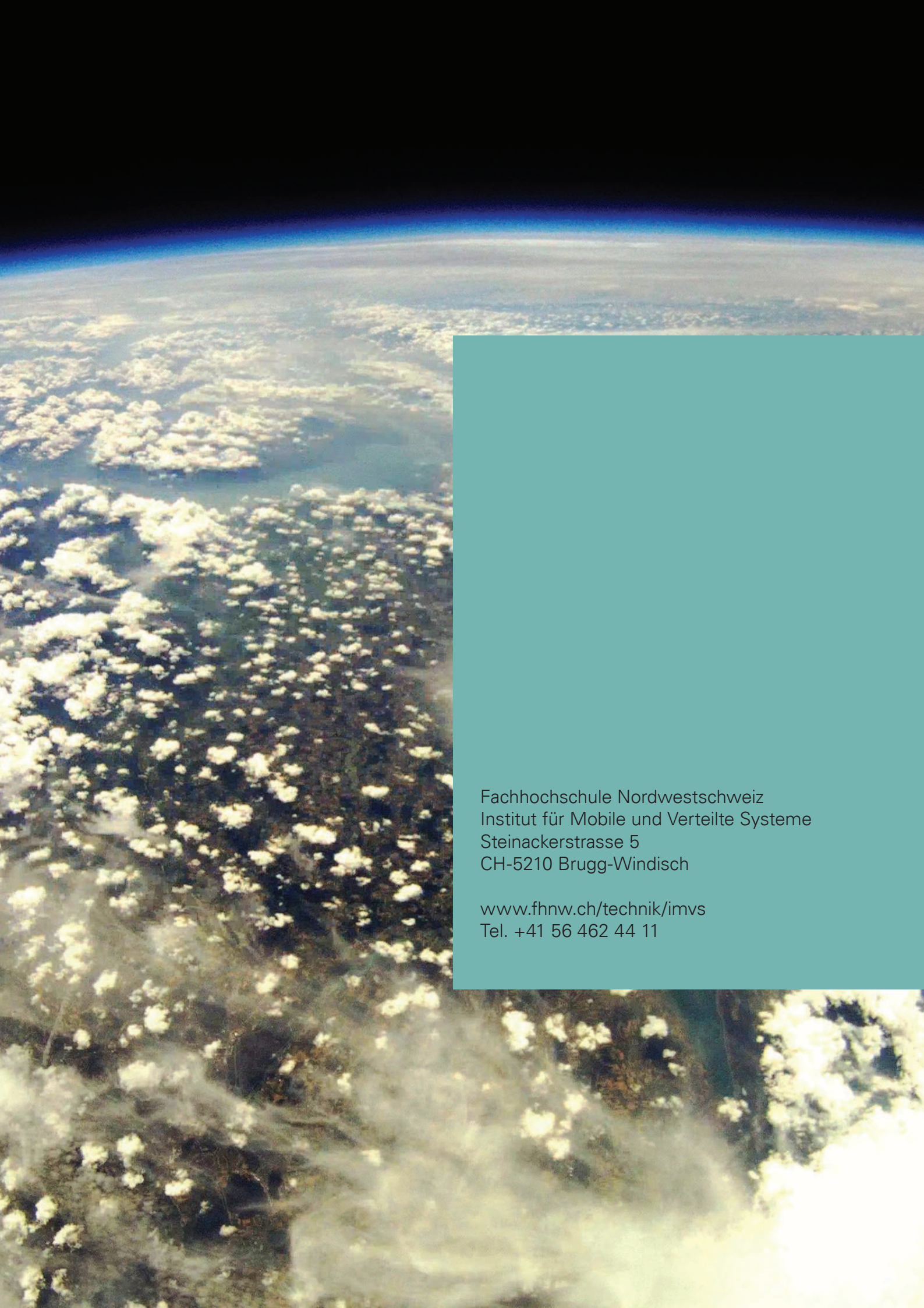
Da wir momentan keinen Zugriff auf ein entsprechendes Kamerasystem haben, versuchen wir die Auswirkungen einer geringeren Bild- und Zeitauflösung auf die Smash-Detektion und -Geschwindigkeitsmessung mit Hilfe des Systems PROMON von AOS zu ermitteln. Daher haben wir in einer Trainingshalle einen insgesamt 42-minütigen Mitschnitt von mehreren Trainingsspielen mit einer Auflösung von 1280×720 und einer Bildrate von 200 erstellt. Diese Testdaten weisen leider nicht die gleich hohe Qualität wie die an den Swiss Open unter professionellen Bedingungen erstellten Videosequenzen auf. Daher sind die nachfolgenden Ergebnisse in Tabelle 1 mit grosser Vorsicht zu interpretieren. Dennoch geben Sie uns Hinweise darauf, welche Schwierigkeiten bis zu den nächsten Badminton Swiss Open 2013 noch gelöst werden müssen, um den Zuschauern zuverlässige Smash-Geschwindigkeitsangaben liefern zu können.

Die Werte in Tabelle 1 sagen natürlich noch nichts über die Güte der Smash-Geschwindigkeitsmessung aus. Eine echte Validierung der Geschwindigkeitsmessung hat bis jetzt noch nicht stattgefunden, wäre aber sicherlich wünschenswert. Bei langsameren Schlägen und unter klar vorgegebenen Bedingungen könnte eine solche Validierung beispielsweise mithilfe einer Radarmessung durchgeführt werden, wobei ganz klar sein müsste, zu welchem exakten Zeitpunkt die Geschwindigkeitsmessung stattgefunden hat, da die Shuttle-Geschwindigkeit im Verlauf der Smash-Flugbahn schnell absinkt. Denkbar sind aber auch ganz andere Ansätze basierend auf Sensoren zur Validierung oder Plausibilisierung unserer Geschwindigkeitsmessung.

Referenz

[SS11] Schindler, M., Stamm, C. Highspeed Videoverarbeitung. IMVS Fokus Report, 2011.

13 Die Bildweite b lässt sich aus der einfachen Linsengleichung $1/g + 1/b = 1/f$ und dem Abbildungsmassstab $A = b/g = h/G$ durch $b = f\,(A + 1)$ bestimmen, wobei g die räumliche Distanz zwischen Objekt und Hauptebene der Linse darstellt und G die Grösse des Objekts und h die Bildgrösse des Objekts sind.



Fachhochschule Nordwestschweiz
Institut für Mobile und Verteilte Systeme
Steinackerstrasse 5
CH-5210 Brugg-Windisch

www.fhnw.ch/technik/imvs
Tel. +41 56 462 44 11