

FOKUS REPORT

NFC im Spital

Vereinfachte Patientenübergabe
und -übernahme

Seite 5

Agile Methoden

Unterrichten von agilen Software-
Entwicklungsmethoden

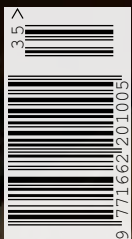
Seite 15

Indoor Tracking

Technik und Kunst in engem
Zusammenspiel

Seite 21

2013



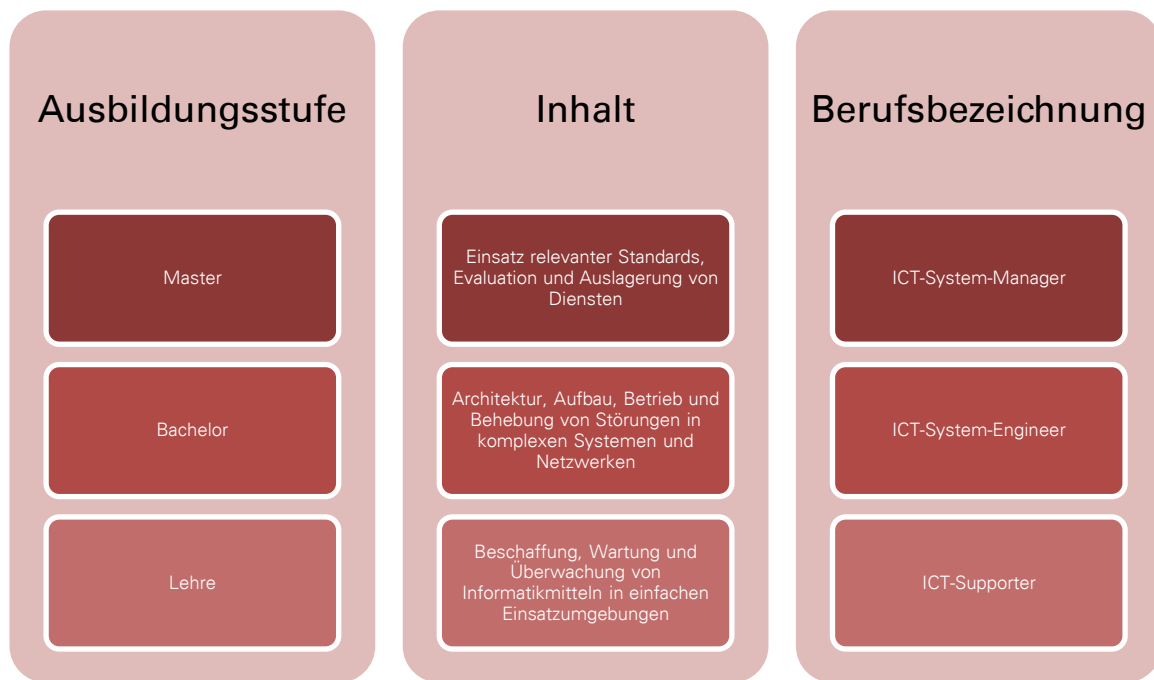


Abbildung 1: Berufsbild ICT-System-Management

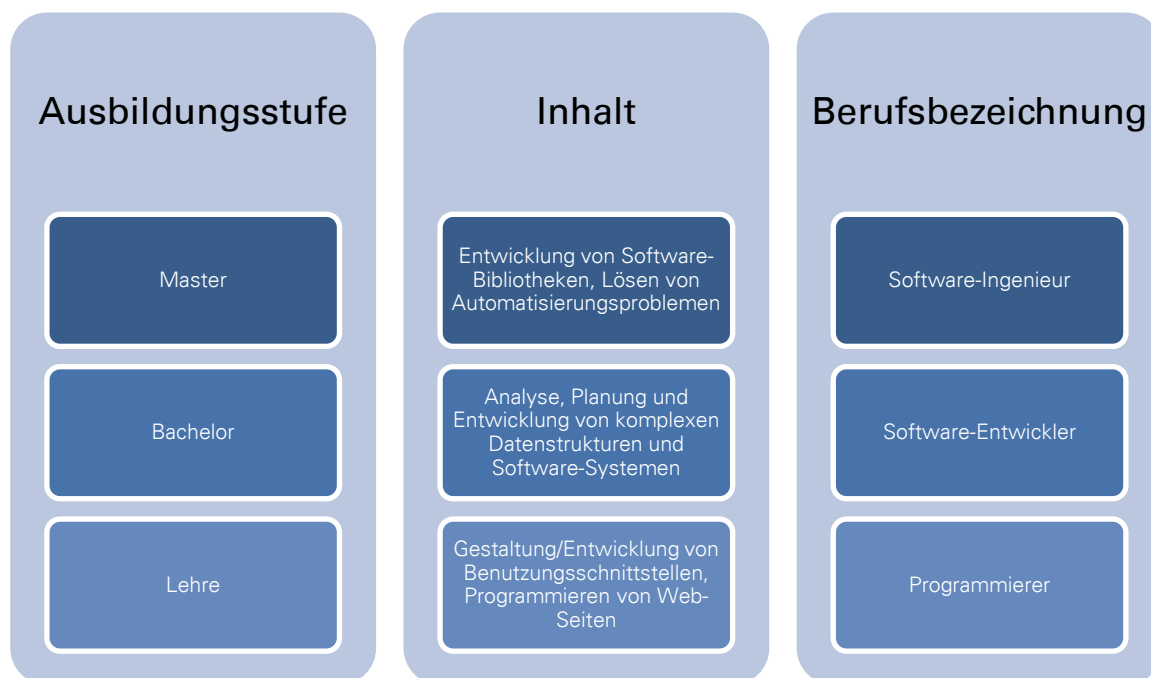


Abbildung 2: Berufsbild Software-Entwicklung

Editorial

Seit gut fünfzehn Jahren reden Fachleute von einem drohenden Informatik-Fachkräftemangel in der Schweiz – und die Presse verkürzt salopp auf Informatikermangel und manifestiert dadurch unbewusst eines der Hauptprobleme, mit dem die Informatik heute zu kämpfen hat: es fehlen klar unterscheidbare, verständliche Berufsbilder im Informatikbereich. Während zum Beispiel in der Medizin zwischen medizinischen Praxisassistentinnen, praktischen Ärztinnen und Herzspezialisten unterschieden wird, so spricht man in der Informatik schlicht und einfach von Informatikern oder Informatikerinnen (mit Schwerpunkt xy).

Unter dem Begriff Informatiker fasst die Gesellschaft heute fast alle Personen zusammen, welche sich mehr oder weniger gut mit Computern auskennen. Das führt zu diffusen und schwer vermittelbaren und somit auch unattraktiven Berufsbildern. Bloss weil in vielen Fachgebieten und Berufen Mathematik verwendet wird, heisst das noch lange nicht, dass alle, die Mathematik einsetzen, auch Mathematiker sind. Das Gleiche sollte auch für Informatik und Informatiker gelten.

Das Problem liegt aber nicht alleine in der Gesellschaft, welche kaum zu unterscheiden mag zwischen der Benutzung von Computern und Computer nutzbar machen. Das Problem liegt bei den Informatikern selber. Sie wissen selber kaum mehr, wo Informatik anfängt und endet oder welche Fähigkeiten auf welcher Altersstufe vermittelt werden sollten. Wenn bereits Absolventen einer Informatiker-Berufslehre mit Schwerpunkt Applikationsentwicklung sich mit Anforderungsanalyse, Software-Konzeption und objektorientierter Programmierung auseinandersetzen, welche Aufgaben bleiben dann noch für Bachelor- und Master-Absolventen? Hier braucht es mehr Klarheit!

Eine zweckmässige, altersgerechte Unterscheidung könnte sein: Mit 16 wissen alle Jugendliche, wie man Computer und andere digitale Endgeräte professionell verwendet und sie verstehen, wie Informationen gespeichert und ausgetauscht werden. Maturandinnen und Lehrabsolventen mit Informatikschwerpunkt können dann im Alter von 20 Jahren den Computer für eigene Zwecke, im Sinne eines einfachen digitalen Werkzeugs programmieren. Und ab dem 24. Lebensjahr sind ausgebildete Informatiker schliesslich im Stand, den Computer für komplexere, intelligente Automatisierungsaufgaben zu programmieren und somit die volle Mächtigkeit des Computers für verschiedenste Anwendungsgebiete gewinnbringend anzubieten. Plakativ heisst das: zuerst Office richtig bedienen, dann Texteditor selber programmieren und schliesslich im Zusammenspiel mit anderen Disziplinen eine Grammatikprüfung entwickeln.

Inhalt

Editorial: Die Informatik braucht klare Berufsbilder	3
NFC Einsatz im Spital	5
Stateful View Controllers in iOS	10
Teaching Agile Software Development at University Level	15
Large-Scale Indoor Tracking	21
Parsing Graphs: Applying Parser Combinators to Graph Traversals	31
A space- and time-efficient Implementation of the Merkle Tree Traversal Algorithm	35

Impressum

Herausgeberin:
 Fachhochschule Nordwestschweiz FHNW
 Institut für Mobile und Verteilte Systeme
 Bahnhofstrasse 6
 CH-5210 Brugg-Windisch
www.fhnw.ch/technik/imvs
 Tel +41 56 202 99 33

Kontakt: Prof. Dr. Jürg Luthiger
juerg.luthiger@fhnw.ch
 Tel +41 56 202 78 23
 Fax +41 56 462 44 15

Redaktion: Prof. Dr. Christoph Stamm
 Layout: Claude Rubattel
 Erscheinungsweise: jährlich
 Druck: jobfactory Basel
 Auflage: 150

ISSN 1662-2014 (Print)
 ISSN 2296-4169 (Online)

In den Diskussionen zur Erschliessung von mehr Informatikfachkräften votieren viele Exponenten für verschiedene (auch sozioinformatische) Ausweitungen des Berufsbildes des Informatikers. Sie erhoffen sich dadurch eine Attraktivierung, um weitere Personenkreise und vor allem vermehrt junge Frauen als Informatikerinnen zu gewinnen. Ich bin überzeugt, dass die Ausweitung nur eine Scheinlösung ist. Obwohl man dadurch tatsächlich mehr Leute für Informatikbelange sensibilisieren kann, so weitet man gleichzeitig das Tätigkeitsfeld des Informatikers noch weiter aus und erhöht die entscheidende Dichte, also die Anzahl Personen pro Berufsbild, nicht wirklich.

Anstatt einfach das Berufsbild des Informatikers ins nebulöse auszuweiten, sollten zuerst klar verständliche und unterscheidbare Berufsbilder mit informatischen Grundkenntnissen definiert werden. In den Abbildungen 1 und 2 (Umschlaginnenseite) sind konkrete Vorschläge für das ICT-System-Management und die Software-Entwicklung aufgeführt. Weitere solche Berufsbilder können und sollen definiert werden. Damit lassen sich die Diskussionen über den Fachkräftemangel im Informatikbereich versachlichen: es kann besser zwischen informatischen Grundkenntnissen und verschiedensten Fachkompetenzen von Informatik-Fachkräften unterschieden werden!

Immer mehr Arbeitsabläufe werden digitalisiert und Unmengen an Software entwickelt. Aus Eile oder Unvermögen wird aber übersehen, dass viel Software schlicht überflüssig wäre, wenn sie nicht einfach herkömmliche Arbeitsabläufe an den Computer umsiedeln, sondern diese tatsächlich vereinfachen und automatisieren würde.

Nehmen wir als Beispiel den Zahlungsverkehr. Zum Begleichen der Rechnung schickt man den Einzahlungsschein zur Bank, welche diesen einscannt und die relevanten Informationen der maschinell lesbaren Referenznummer entnimmt oder man betreibt zuhause Online-Banking und tippt die Referenznummer entweder mühsam ab oder schafft sich einen passenden Scanner mit Schrifterkennung an. Die in der Referenznummer codierten Daten werden also mindestens zweimal erfasst: zuerst beim Rechnungssteller und danach zur Ausführung der Zahlung. Die digitalen Daten werden beim Gläubiger gehortet und fliesen nicht digital zum Schuldner. Nun gut, dieses Problem ist längst erkannt worden und es gibt eine einfache und professionelle Abhilfe, die sogenannte E-Rechnung, welche im Online-Banking mit einem Mausklick bezahlt werden kann.

Dieses Beispiel veranschaulicht schön, dass es nicht reicht, etablierte Geschäftsprozesse eins-zu-eins abzubilden, sondern dass dies allenfalls nur ein Zwischenschritt zum medienbruchfreien, digitalen Datenfluss darstellt. Durch falsch verstandenen Datenschutz wird dieser digitale Datenfluss jedoch oft verunmöglicht, obwohl er

die beste Voraussetzung für eine echte Automatisierung darstellen würde, bei der die Menschen von stupiden Arbeiten entlastet werden können. Automatisierung kann aber auch Kontrollverlust bedeuten und zu Ohnmacht führen, wenn sie zum Ausschalten von unerwünschter Einflussnahme missbraucht wird. Hier sind Software-Entwickler echt gefordert und werden es in naher Zukunft noch viel stärker sein.

Das Internet der Dinge steht vor der Tür. Neben Geschäftsprozessen und sozialen Netzwerken werden auch Gegenstände des täglichen Bedarfs im digitalen Konzert mitspielen. Zum Betrieb, zur Verwaltung und zur Einbindung dieser Gegenstände in aktuelle und neue Datenflüsse sollen wieder Unmengen an neuer Software entwickelt werden und die Rufe nach mehr Informatik-Fachkräften werden kaum abnehmen. Doch es wird nicht viel nützen. Was ist also zu tun?

Die grössten Aufwände bei der Digitalisierungen der Geschäftsprozesse und sozialen Netzwerke sind in den hochentwickelten Ländern bereits getätigt worden. Dabei handelt es sich oft um Einzellösungen, zuerst für den Grossrechner, dann für den PC, danach für den Web-Browser und nun fürs Smartphone. Anstatt klassische Arbeitsabläufe mit digitalen Tools vollzupflastern, brauchen wir Computerprogramme, welche die Prozessoren des Computers für mehr als Datenkonvertierung benutzen, aus Daten(müll)halden Informationen extrahieren, zusammentragen, konsolidieren und sinnvoll präsentieren – und das bitte automatisch!

Dazu braucht es gewiefte Software-Entwickler, die in der Lage sind, Abläufe und Daten zu analysieren, Regelmässigkeiten zu erkennen und zu formalisieren und komplexe Datenflüsse zu vereinfachen. Es braucht hochintelligente Software-Ingenieure, welche Systeme automatisieren, mit Lernfähigkeit und ganz wichtig mit Transparenz ausstatten. Denn nur wenn ein automatisiertes System in der Lage ist, seine gemachten Entschiede seinen Benutzern zu kommunizieren, ist es möglich, dass die Benutzer ihrer berechtigten Einflussnahme nicht beraubt werden. So kann verhindert werden, dass Automatisierung zur Ohnmacht und somit zur Ablehnung führt. Hier gibt es noch sehr viel zu tun!

Die Ausbildungsstätten werden gut daran tun, vor allem die intelligentesten jungen Menschen für die Informatik zu motivieren. Dazu braucht es aber klare Berufsbilder, intellektuelle Herausforderungen und nicht zuletzt auch kompetitive Entlohnungen. Der Abwärtstrend der Durchschnittsgehälter im letzten Jahrzehnt ist symptomatisch für die diffusen Entwicklungen in der Informatik, dass jeder, der ein Computerprogramm schreiben kann, auch ein Informatiker sein soll.

Prof. Dr. Christoph Stamm

NFC Einsatz im Spital

Obwohl Near Field Communication eine relativ neue Technologie ist und noch nicht von allen grossen Smartphone Herstellern unterstützt wird, werden immer neue Anwendungsfälle entdeckt, wo der Einsatz dieser Technologie eine sinnvolle Verwendung findet. In einem Forschungsprojekt mit der Firma SenTec haben wir NFC im Spitalumfeld evaluiert und zwei konkrete Anwendungsfälle gefunden, wo dank NFC die Bedienung einer mobilen Applikation viel einfacher gestaltet werden konnte. In diesem Artikel geben wir zuerst einen Überblick über NFC und dessen Anwendungsgebiete und zeigen dann, wie wir NFC für unser Projekt konkret eingesetzt haben.

Emanuel Hediger, Jürg Luthiger | juerg.luthiger@fhnw.ch

Die Abkürzung NFC steht für „Near Field Communication“. Dabei handelt es sich um eine Weiterentwicklung der schon weit verbreiteten RFID (Radio Frequency Identification) Technologie, einem System zur Identifizierung und Lokalisierung von Objekten mittels Transponder und Magnetwellen. NFC ermöglicht eine Datenübertragung mit einer Übertragungsfrequenz von 13,56 MHz und einer Datenübertragungsrate von maximal 424 kBit/s, dies bei einem Abstand von bis zu 10 cm. Im Vergleich zu anderen drahtlosen Technologien wie WLAN oder Bluetooth weist NFC einen geringeren Energiebedarf auf. Das ist ein wichtiger Vorteil vor allem beim Einsatz in einem Smartphone. Die Übertragung von kleinen Datenmengen im Bereich von wenigen Megabytes kann mit NFC ressourcenschonender umgesetzt werden. Das IMVS verfolgt seit längerer Zeit mit grossem Interesse die Entwicklung dieser Technologie [DG11].

NFC unterscheidet drei verschiedene Betriebsmodi [LR10]. Diese sind in der Tabelle 1 zusammengefasst.

Eine einfache Sicherheit ist vor allem dadurch gegeben, dass sich die NFC-Mobiltelefone wäh-

Reader/Writer-Modus	Das NFC-Mobiltelefon wird zum Leser und kann passive NFC-Tags auslesen und mit Daten beschreiben.
Card-Emulation-Modus	Das NFC-Mobiltelefon ist passiv und emuliert ein NFC-Tag, typischerweise eine Smartcard. Ein RFID-Leser, z.B. ein Kassensystem oder ein Türschloss, greift auf das im NFC-Mobiltelefon emulierte Smartcard-Tag zu.
Peer-to-peer-Modus	In dieser Betriebsart können Informationen zwischen zwei aktiven NFC-Mobiltelefonen ausgetauscht werden. Es kann sich dabei um einen Gutschein handeln oder um gegenseitige Identifikationen, damit grössere Datenmengen danach über Bluetooth oder über das Internet kommuniziert werden können.

Tabelle 1: NFC Betriebsmodi

rend der Kommunikation nahe aneinander befinden müssen. Bei Bedarf sollte man die Daten aber verschlüsseln und z.B. bei Kreditkarten auf die minimal nötigen Daten zur Zahlung beschränken. Trotzdem sind „Man in the Middle“ Attacken möglich, falls das Signal abgefangen und mit veränderten Daten an das korrespondierende Gerät weitergeschickt werden.

iPhone unterstützt NFC soweit noch nicht. Jedoch gibt es diverse Android Smartphones, welche mit NFC ausgerüstet sind wie zum Beispiel das Samsung Galaxy SIII oder Samsung Google Nexus. Ab Version 2.3.3 (Gingerbread) wird von Android eine API zur Verfügung gestellt, um mit NFC zu kommunizieren. Dabei können Daten im standardisierten NDEF (NFC Data Exchange Format) [NDEF] oder benutzerspezifischen Format aus Tags gelesen werden oder mittels NDEF-Nachrichten von einem Gerät an ein anderes geschickt werden [BEAM].

In der Tabelle 2 sind typische Anwendungsszenarien aufgelistet wo die NFC-Technologie zum Einsatz kommen kann.

Pairing	Paarung und/oder Zuordnung von verschiedenen Geräten, Orten oder Objekte
Verbindungsaufbau	Initialer Aufbau einer bidirektionalen Bluetooth oder WLAN-Verbindung zwischen zwei Geräten
Zugangskontrolle	Authentifizierung eines Benutzers
Kryptografisch gesicherte Übertragung	Übertragung von sensiblen Daten (z.B. bargeldloses Zahlen)
Nahbereichskommunikation	Übertragung/Aufnahme von kleineren Datenmengen auf Kurzdistanz

Tabelle 2: Typische Einsatzszenarien der NFC Technologie

NFC Anwendungsbeispiele

Kreditkartenfirmen wie Mastercard mit payPass [MAPP] und VISA payWave [VIPW] bauen NFC-Tags in ihre neuen Kreditkarten ein und unterstützen die Bezahlung kleiner Beträge per Smartphone. Bezahlssysteme wie GoogleWallet [GGWA], Paypal [PNFC], GiroGo [GIRG] (deutsche Sparkasse) oder MyWallet (deutsche Telekom) bieten ebenfalls NFC-Unterstützung. Somit ist es möglich, Bezahlungen ohne PIN-Eingabe zu tätigen, indem man lediglich das Gerät oder die Karte an das Lesegerät hält.

Ausserdem soll es in Zukunft möglich sein, vom Smartphone aus Daten via NFC an den Fernseher zu übertragen. So kann man bequem Musik oder Fotos auf dem Fernseher betrachten und muss dazu lediglich das Smartphone an den Fernseher halten.

Interaktive Werbeflakate können durch einfaches Berühren zu Zusatzinformationen wie Links auf Webseiten, Adressdaten oder sogar Gutscheine und Eintrittskarten führen.

Kfz-Hersteller Daimler, General Motors, Honda, Hyundai, Toyota und Volkswagen arbeiten mit Mobiltelefon-Herstellern wie Microsoft oder Samsung zusammen, um NFC in das Auto zu integrieren. Man will z.B. Eintrittskarten für eine Veranstaltung mit einem NFC-Tag ausrüsten, so dass man die Eintrittskarte auf einem Hotspot im Auto auflegen kann, um das Navigationsgerät zu aktivieren, welches dann den Weg zum Veranstaltungsort automatisch einstellt. Auch könnte das Mobiltelefon auf dem Hotspot kabellos aufgeladen werden oder es findet ein Austausch von multimedialen Inhalten zwischen Fahrzeug und Gerät statt.

Generell kann NFC als Autoschlüssel [CAKN] oder auch als Schlüssel für Hotels verwendet werden.

Kulturstätten oder Vergnügungsparks können NFC Schranken beim Eingang bzw. Ausgang installieren, so dass die Bezahlung einfacher ausgeführt werden kann. Oder sie stellen interaktive Karten zur Verfügung, welche die Kulturstätte mittels NFC-Tags interaktiv erklären und Zusatzinformationen bereitstellen [TUKU].

Der Spielkonsolen Hersteller Nintendo will die Spielkonsole Wii U mit NFC [NINF] ausrüsten, um z.B. Spielfiguren per NFC auf dem Spielfeld platzieren zu können. Activision hat NFC schon sehr erfolgreich für die Spielreihe Skylanders eingesetzt [SKLA]. Auch Disney baut mit dem Projekt Disney Infinity auf die NFC Technologie [DYIN]. Dabei hat man ein Portal (Platte zum Aufsetzen von Tags), Spielfiguren und sogenannte Powerdisks entwickelt. Die Spielfiguren wandern durch Platzieren auf dem Portal per NFC in das Spiel oder man kann per NFC neue Welten freischalten. Powerdisks geben den Spielfiguren Zusatzkräfte.



Abbildung 1: SenTec Digital Monitor (SDM)

NFC im Spitalumfeld

In einem Spital braucht man für eine sinnvolle Überwachung eines Patienten lückenlose Informationen über seine Ventilation und seine Sauerstoffversorgung. Deshalb entwickelte die Firma SenTec AG das Digital Monitoring System SDMS (Abb. 1). Das System ist mit einem innovativen Sensor ausgerüstet, um eine kontinuierliche und nicht-invasive Überwachung des Kohlendioxid-Partialdrucks (pCO₂), der Sauerstoffsättigung (SpO₂) sowie des Pulses in Echtzeit zu ermöglichen.

Innerhalb eines KTI-Projektes haben wir zusammen mit der Firma SenTec diese Überwachungsmöglichkeiten erweitert und ein Client-Server-System entwickelt, das dank einer mobilen Applikation eine kontinuierliche Beobachtung des Patienten jederzeit und überall erlaubt.

Bei der Entwicklung dieser mobilen Applikation ist grosser Wert auf die Benutzbarkeit gelegt worden. Da im Zentrum die Anforderungen und die Wünsche der Benutzergruppen stehen, ist es wichtig, dass die neue Applikation die existierenden Arbeitsabläufe des Spitalpersonals effizient unterstützt und nicht behindert. Mit einem benutzerzentrierten Design-Ansatz sind die verschiedenen Anforderungen ermittelt und in charakteristischen Nutzerprofilen zusammengefasst worden. Ebenso sind die Aufgaben und Ziele der Nutzer, Arbeitsabläufe und die Arbeitsumgebung, zu der auch die technischen Rahmenbedingungen zählen, in sieben Spitälern der USA und der Schweiz analysiert worden. Insgesamt resultiert daraus ein klar besseres Verständnis der Arbeitsabläufe beim Schichtwechsel, der Kommunikationskanäle und der Verantwortlichkeiten und dieses bessere Verständnis hat die Gestaltung der mobilen Applikation stark beeinflusst.

Aus den diversen Interviews identifizierten wir fünf verschiedene Benutzergruppen (Personas) mit der Hauptnutzerin „Xenia“. Xenia ist eine Pflgende. Sie ist auf ihrer Schicht für mehrere Patienten verantwortlich und braucht deshalb eine einfache Möglichkeit, um den Status der Patienten zu überblicken. Vor allem ist sie an den Alarmmeldungen interessiert. Das Benutzerszenario von Xenia kann wie folgt zusammengefasst werden:

1. Start der Schicht (Rapport mit Übernahme von Patienten, Login in das System).

2. Überwachung der Patienten, Übernahme neuer Patienten und Übergabe von Patienten, die aus der Station austreten.
3. Massnahmen in Alarmsituationen einleiten
4. Interaktion zwischen Patient-Arzt-Pflege koordinieren
5. Ende der Schicht (Patientenübergabe)

Die Integration der mobilen Applikation in dieses Szenario wird durch folgende Massnahmen umgesetzt:

- *Optimierte Informationen:* Es werden nur die Informationen angezeigt, die für den aktuellen Arbeitsschritt auch notwendig sind (Abb. 2), die der jeweiligen Benutzergruppe entsprechen und welche die bekannte Visualisierung aus dem Spitalumfeld aufnehmen.
- *Optimierte Interaktionen:* Durch Login über die Kamera mittels QR-Code, durch NFC Unterstützung bei der Übernahme von Patienten, durch NFC Unterstützung bei der Patientenübergabe während dem Schichtwechsel sind manuelle Eingaben auf ein Minimum reduziert worden.

Die Integration der NFC Technologie in den Arbeitsablauf von Xenia wird in den folgenden Abschnitten detailliert vorgestellt.

NFC Einsatz bei der Patientenübernahme

Xenia authentisiert sich bei Schichtbeginn auf dem Server. Während ihrer Schicht ist sie für die Betreuung von fünf Patienten zuständig. Xenia muss einen neuen Patienten überwachen. Der Patient liegt im Doppelzimmer 305 auf der rechten Seite. An der Wand neben dem Bett ist ein NFC-Tag angebracht und als Bettenplatz bzw. Raum „305 re“ auf dem Server registriert.

Xenia positioniert zuerst den SDM neben dem Patientenbett und schliesst danach die Sensoren an den Patienten an. Der SDM ist ebenfalls mit ei-

nem NFC-Tag ausgerüstet und ebenfalls im Server hinterlegt. Sie liest nun mit ihrem NFC-fähigen Smartphone zuerst das NFC-Tag auf dem SDM aus und danach das NFC-Tag an der Wand. Durch das Scannen der beiden NFC-Tags wird eine Paarung zwischen SDM und Bettenplatz durchgeführt. Dank dieser Zuordnung von SDM zu Bettenplatz kann der Server die entsprechenden Patienteninformationen für Xenia bereitstellen und diese über ihre mobile Applikation anzeigen. Dieser Ablauf ist auch in Abbildung 4 dargestellt.

NFC Einsatz bei der Patientenübergabe

Wenn Xenia nun ihre Schicht beendet, muss eine Übergabe der Patienten an eine andere Pflegeperson stattfinden. Auf ihrem Smartphone hat Xenia immer noch die Liste ihrer überwachten Bettenplätze bzw. Patienten (Abb. 2). Um diese Patienten zu übergeben wählt Xenia zunächst die Patienten in ihrer Liste an und hält ihr Smartphone mit dem Rücken an das Smartphone der anderen Pflegeperson. Beide Smartphone sind mit NFC ausgerüstet. Xenia muss einen Dialog auf ihrem Smartphone quittieren, so dass der Transfervorgang gestartet wird. Die andere Pflegeperson bestätigt als Empfängerin ebenfalls über einen entsprechenden Dialog die Annahme der zu transferierenden Patienten. Jetzt kann der Server die neuen Zuordnungen vornehmen und Xenia aus der Verantwortung für ihre Patienten lösen. Danach kann Xenia ihre Schicht abtreten. Dieser Ablauf ist auch in Abbildung 5 dargestellt.

Remote Monitoring System V-CareNeT mobile

Die Software Lösung besteht grundsätzlich aus einem Server und mehreren Client Applikationen. Der Server kann eine beliebige Anzahl von SDMs integrieren. Er ist in der Lage mit dem SDM zu interagieren, Befehle zu senden, die gemessenen Patientendaten auszulesen und diese zentral abzuspeichern. Dazu ist der Server in verschiedene Module aufgeteilt (Abb. 3).

Der Server kann über seine Schnittstellen verschiedenen Client Applikationen Zugang bieten:

- *Mobile Monitor App (SMMA):* Mobile Applikation auf dem Smartphone. Sie erlaubt die mobile Überwachung von Patienten, die an einen SDM angeschlossen sind.
- *Monitor (SM):* Rich Internet Application (RIA) stellt eine Übersicht über alle aktiven SDMs zur Verfügung. Diese Anwendung dient zur Visualisierung verschiedener SDMs.
- *Administration App (SDA):* Administrationskonsole als Web Applikation. Hier wird die Gesamtapplikation konfiguriert.

Weitere Client Applikationen können jederzeit über die REST-Schnittstelle in das System integriert werden.



Abbildung 2: Übersichtsseite der mobilen Applikation. Pro Zeile ist ein Bettenplatz aufgelistet. Zusätzliche Informationen werden nur im Alarmfall eingeblendet.



Abbildung 3: Software Architektur mit den wichtigsten Modulen. Der SenTec Digital Monitor (SDM) wird hier nur als Datenquelle betrachtet, obwohl das Gerät seine eigene Benutzerschnittstelle besitzt.

NFC und Mobile Monitor App SMMA

Wie man anhand der beschriebenen Anwendungsszenarien erkennt, wird NFC auf zwei verschiedene Arten eingesetzt. Im ersten Beispiel wird im NFC Betriebsmodus Reader/Writer ein Pairing zwischen SDM und Bettenplatz durchgeführt. Dadurch wird eine implizite, eindeutige Zuordnung zwischen dem Patient bzw. seinem Bettenplatz und dem SDM, an dem der Patient angeschlossen ist, erstellt. Implementierungsdetails dazu werden im nächsten Abschnitt vorgestellt.

Im zweiten Beispiel wird im Peer-to-Peer Modus eine Nahbereichskommunikation aufgebaut, um die „digitale“ Übergabe der betreuten Patienten zwischen zwei Pflegepersonen zu unterstützen. Dank NFC Einsatz spart das Pflegepersonal bei diesen Arbeitsschritten erheblich Zeit. Mühsames Eintippen oder Auswählen von SDM-Nummer und Bettenplatznummern entfallen und auch die Übergabe von überwachten Patienten geht schnell und braucht nur eine Bestätigung der beiden teilnehmenden Pflegenden. Auch zu diesem Anwendungsfall werden noch Implementierungsdetails angegeben.

Implementierungsdetails zur Patientenübernahme

In Abbildung 4 sind die Interaktionen zwischen den Software-Modulen dargestellt, die bei der Übernahme eines neuen Patienten involviert sind. NFC wird in den Sequenzen „SDM Tag lesen“ und „Raum Tag lesen“ eingesetzt. Mit Raum-Tag ist

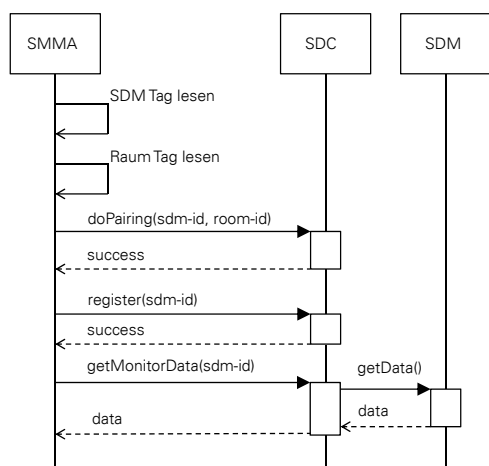


Abbildung 4: Sequenzdiagramm bei der Übernahme eines Patienten zwischen mobiler Applikation SMMA, Server SDC und SenTec Monitor SDM

hier das Bettenplatz-Tag gemeint. Beim Lesen eines NFC-Tags wird Android einen sogenannten *Intent* auslösen. Jede Applikation kann diesen *Intent* verarbeiten, sofern die entsprechenden Einstellungen vorgenommen sind. In unserem Falle ist es jedoch wichtig, dass das Android System den *Intent* an die SMMA Applikationen weiterleitet. Es ist deshalb notwendig, dass die SMMA Applikation entsprechende Massnahmen trifft. Die SMMA Applikation ist beim Lesen eines Tags aktiv und im Vordergrund. Mit dem Flag *FLAG_ACTIVITY_SINGLE_TOP* kann das NFC-Subsystem nun so konfiguriert werden (Listing 1), dass es den NFC *Intent* direkt an die Anwendung im Vordergrund leitet, also an die SMMA Applikation. Der NFC *Intent* löst in der Activity einen Statuswechsel von Running auf Paused zurück zu Running aus. Deshalb kann über die Lifecycle-Methode *onResume()* die entsprechende Applikationslogik für das Lesen der beiden NFC-Tags gestartet werden.

Um das Handling der NFC-Tags, die auf den SDMs und bei den Patientenbetten verteilt werden müssen, möglichst einfach zu gestalten, ist bewusst auf das explizite Beschreiben der NFC-Tags verzichtet worden. Die Tags haben deshalb lediglich eine Identifikationsnummer und keine Meldung im NDEF-Format.

Implementierungsdetails zur Patientenübergabe

In Abbildung 5 ist die Sequenz bei der Übergabe der Patienten dargestellt. Mit „SDM auswählen“ ist gemeint, dass die mit den Bettenplätzen und Patienten assoziierten SDMs selektiert werden.

Im Peer-to-Peer Modus können NDEF-Meldungen mit der Android Beam Technologie von einem Smartphone zum anderen übertragen werden. In Listing 2 sieht man, dass die NDEF-Meldung alle SDM-IDs im JSON Format enthält. In #1 werden die selektierten SDMs ausgelesen, um in #2 die NDEF-Meldung zu erzeugen. Die JSON-Formatierung der SDM-IDs startet auf Zeile #3.

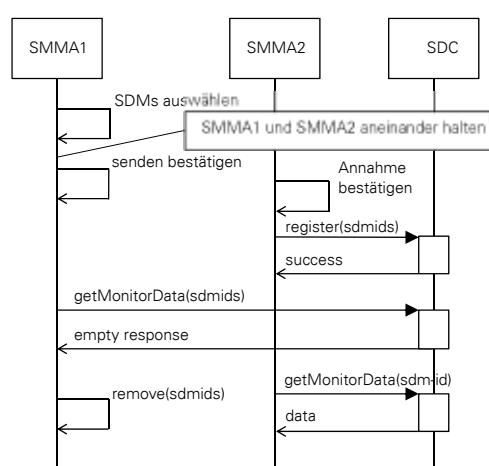


Abbildung 5: Patientenübergabe durch den Transfer der entsprechenden Patienten bzw. der assoziierten SDMs


```

...
// Create and prepare intent to signal "singleTop" for the android runtime system
nfcIntent = PendingIntent.getActivity(activity, 0, new Intent(activity,
    activity.getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
...
// Give priority to the foreground activity when dispatching a discovered nfc tag
nfcAdapter.enableForegroundDispatch(activity, nfcIntent, null, null);
...

```

Listing 1: Die relevanten Code-Sequenzen für das Dispatching beim Einlesen eines NFC-Tags

In Abbildung 5 erkennt man, dass das empfangende SMMA (SMMA2) die erhaltenen SDMs mittels ihren IDs bei der Serverapplikation SDA registriert. Durch den Registrierungsprozess werden die entsprechenden Zuordnungen für das sendende SMMA (SMMA1) entfernt. Bei der nächsten Datenabfrage wird SDA keine Daten zurücksenden können, da der entsprechende SDM nicht mehr für das SMMA registriert ist. Nun kann das SMMA die betroffenen SDMs aus der Beobachtungsliste entfernen.

Zusammenfassung

Dank Einsatz der NFC-Technologie können wichtige Arbeitsschritte beim Patienten-Monitoring benutzerfreundlicher gestaltet werden. Mühsames manuelles Eintippen entfällt sowohl bei der Patientenübernahme wie auch bei der Patientenübergabe komplett. Die NFC-Unterstützung in Android ist gut und stabil. Die bereitgestellten APIs vermindern den Programmieraufwand deutlich. Noch fehlt aber die Erfahrung dieser NFC-Anwendung im täglichen Alltagseinsatz, da der Prototyp bisher keinem grösseren Praxistest ausgesetzt werden konnte.

Referenzen

- [BEAM] <http://developer.android.com/guide/topics/connectivity/nfc/nfc.html>
- [CAKN] <http://www.nfcworld.com/2011/02/15/35998/morpho-announces-nfc-key-fob-that-connects-to-any-wifi-phone/>
- [DG11] Dominik Gruntz; NFC mit Android, IMVS Fokus-Report, 2011.
- [DYIN] <https://infinity.disney.com/de-ch>
- [GGWA] <http://www.google.com/wallet/buy-in-store/>
- [GIRG] <http://www.girogo.de/>
- [LR10] Josef Langer, Michael Roland, Anwendungen und Technik von Near Field Communication (NFC), Springer Verlag, Sept 2010
- [MAPP] http://www.mastercard.com/de/privatkunden/innovationen_paypass.html
- [MYWA] <http://www.my-wallet.com/>
- [NDEF] <http://www.nfc-forum.org/specs/>
- [PNFC] <http://www.androidpit.de/PayPal-Android-App-jetzt-mit-NFC>
- [SKLA] <http://www.skylanders.com/>
- [TUKU] <http://www.touch2go.biz/nfc-tourismus-kultur.html>
- [VIPW] http://www.visaeurope.com/en/cardholders/visa_paywave.aspx

```

public NdefMessage getNdefMessage(NfcEvent event) {
    final String appName = "application/ch.imvs.sentec.client.mobile";
    if (selectedMonitors.size() == 0) {
        return null;
    }
    byte[] msgContent = getSelectedMonitors();
    return new NdefMessage(createMimeRecord(appName, msgContent));
}

private NdefRecord createMimeRecord(String mimeType, byte[] payload) {
    byte[] mime = mimeType.getBytes(Charset.forName("US-ASCII"));
    return new NdefRecord(NdefRecord.TNF_MIME_MEDIA, mime, new byte[0], payload);
}

private byte[] getSelectedMonitors() {
    try {
        JSONObject monitors = new JSONObject();
        monitors.put(TRANSFER_OBJECT_HEADER, TOT_MONITOR_LIST);
        monitors.put(TRANSFER_OBJECT_DEVICE_ID,
            SettingsManager.getInstance().getDeviceId());
        monitors.put(TRANSFER_OBJECT_COUNT, selectedMonitors.size());
        JSONArray mlist = new JSONArray();
        for(int i = 0; i < selectedMonitors.size(); i++) {
            mlist.put(selectedMonitors.get(i).getSdmId());
        }
        monitors.put(TRANSFER_OBJECT_DATA, mlist);
        return monitors.toString().getBytes();
    } catch (JSONException ex) {
        ...
    }
}

```

Listing 2: Code um eine NDEF-Message mit den SDM-IDs als JSON-Array zu erzeugen

Stateful View Controllers in iOS

Für die Entwicklung einer zustandsbasierte Benutzungsschnittstelle eines mobilen Gerätes zeigen wir drei verschiedene Ansätze auf. Wir vergleichen die Ansätze miteinander und führen einen Ansatz genauer aus, welchen wir im Rahmen der Entwicklung eines Störmeldesystems auf dem iPhone entwickelt haben. Dieser Ansatz basiert auf dem State Pattern und verwendet eine einzige einfache tabellarische Darstellung, welche dynamisch mit den zustandsabhängigen Elementen eingefüllt wird.

Moritz Dietsche | moritz.dietsche@fhnw.ch

Das in diesem Artikel beschriebene User Interface einer iOS App ist im Rahmen des KTI-Projekts „App für Störmeldesysteme“ entwickelt worden¹. Diese mobile Anwendung ermöglicht berechtigten Personen den Empfang von Statusmeldungen von Störmeldevorrichtungen. Die erforderliche Zugriffsberechtigung erfolgt über codebasiertes Geräte-Pairing zwischen dem Störmeldesystem und der App.

Problemstellung

Um das oben angesprochene Pairing zweier Geräte durchzuführen, wird eine Benutzeroberfläche benötigt (Abb. 1), welche auf Desktopsystemen oft in Form eines Assistenten oder „Wizards“ umgesetzt wird. Ein solcher Assistent bietet eine Möglichkeit, Schritt für Schritt einen Automaten zu durchlaufen, wobei die Zustandsübergänge entweder vom Benutzer (Auswählen einer Option) oder durch das System (zum Beispiel eine Netzwerkanfrage) ausgelöst werden.

Abbildung 2 zeigt einen Automaten, welcher als Grundlage für einen Assistenten dienen könnte. Er verfügt über Verzweigungen und über Zu-



Abbildung 1: View von PairingStateWaitForResponse nach Annahme durch die Gegenseite

standsübergänge, welche – im Gegensatz zu rein benutzergesteuerten Assistenten – entweder durch den Benutzer oder durch äussere Einflüsse wie eingehende Push Notifications ausgelöst werden. Wichtig zu beachten ist hierbei, dass die Zustände verschiedene User Interfaces repräsentieren, auch wenn diese gemäss der Geschäftslogik einem einzigen Zustand entsprechen. Je nach Ausführungspfad endet der Automat in einem anderen Zustand.

Bei der Umsetzung eines Assistenten der zuvor geschilderten Art auf einer Mobilplattform werden üblicherweise folgende Arbeitsschritte realisiert:

- Erstellung der einzelnen Views;
- Navigation zwischen den Views;
- Definition der Logik, d. h. welche Zustandsübergänge sind möglich;
- Kommunikation zwischen den Zuständen und der übergeordneten Einheit;
- Erweiterbarkeit des Automaten.

Lösungsansätze

Wir beschränken uns im Folgenden auf zwei Kriterien, nach welchen wir die verschiedenen Lösungsansätze beurteilen. Diese sind einerseits die Platzierung der Logik des Automaten, zentral oder auf verschiedene Zustände verteilt. Andererseits unterscheiden wir zwischen Lösungen mit einer und mit mehreren Views. Wir erhalten die vier Ansätze in Tabelle 1.

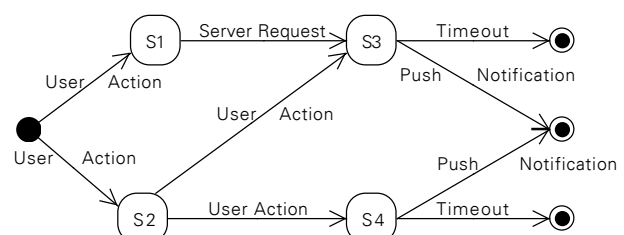


Abbildung 2: Ein Automat, welcher in Form eines Assistenten implementiert werden könnte

	eine View	mehrere Views
verteilte Logik	Lösungsansatz 1: einfach	hier nicht sinnvoll
zentrale Logik	Lösungsansatz 2: grosse View	Lösungsansatz 3: TableView

Tabelle 1: Verschiedene Lösungsansätze in Abhängigkeit der Anzahl Views und der Verteilung der Logik der Zustandsmaschine

Wir beginnen mit einem ersten einfachen Ansatz. Dieser besteht darin, die einzelnen Schritte als separate Views zu implementieren. Diese könnten nun manuell gestaltet werden. Für die Geschäftslogik fügen wir jeder View noch einen View Controller hinzu. Dieser übernimmt die Steuerung der View und verfügt über eine Methode *getNext()*, welche den nächsten Zustand zurückgibt.

In einem nächsten Schritt fügen wir die erstellten Controller einem Container Controller hinzu, welche die Aufgabe hat, die Zustandsübergänge auszuführen. Dies reicht eigentlich schon, um die oben genannten Anforderungen zu erfüllen. Wir benötigen für *n* Zustände lediglich *n* Views und *n+1* Controller. Die Vorteile dieses Ansatzes sind:

- simple Implementierung der Zustände;
- überschaubarer Aufwand für kleinere Anwendungen (Anzahl Views und Controller steigt linear mit der Anzahl Zustände);
- grafische Umsetzung der Views ermöglicht einfaches Ausdrucken des Storyboards unter anderem für Usability-Testing.

Dennoch bleiben ein paar mögliche Probleme:

- Die Logik des Automaten ist auf die verschiedenen View Controller verteilt. Dies vermindert die Übersichtlichkeit ohne weitere Vorteile zu bieten.
- Der Aufwand für eine grosse Anzahl Zustände ist sehr gross, da alle Views von Hand erstellt werden müssen.

In dieser Auflistung nicht berücksichtigt ist die Benachrichtigung des Assistenten darüber, dass nun ein Zustandsübergang ausgeführt werden soll. Diese Problematik besteht allerdings bei allen Ansätzen und kann unterschiedlich gelöst werden. Wir verzichten deshalb auf die Betrachtung dieses Aspekts.

Ein zweiter Lösungsansatz besteht in der Verwendung einer einzigen View. Wir stellen uns diese als eine lange Ansicht aller Schritte vor. Auf einem Smartphone ist allerdings immer nur ein Abschnitt der View sichtbar. Bei einem Zustandsübergang wird lediglich vertikal gescrollt, damit der korrekte Abschnitt sichtbar ist.

Gegenüber dem einfachen Ansatz bietet diese Lösung folgende Vorteile:

Es kann auf die Erstellung mehrerer einfacher Views inklusive ihrer View Controller verzichtet werden. An die Stelle des Container Controllers

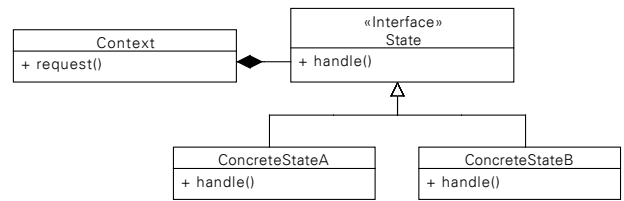


Abbildung 3: UML Klassendiagramm des State Patterns

rückt ein einziger View Controller mit der langen View.

Da nur ein Controller verwendet wird, ist die gesamte Logik darin gekapselt. Leider entsteht bei diesem Ansatz auch ein neuer Nachteil:

- Während die Vorstellung einer langen View sehr einfach wirkt, ist ihre Umsetzung nicht trivial. Auf beiden grossen Smartphone-Plattformen wäre dies mit viel Handarbeit verbunden.

In unserem dritten Ansatz bauen wir auf dem State Pattern auf (Abb. 3). Die verschiedenen Zustände bieten ein einheitliches Interface gegen aussen an. Bei einem Zustandsübergang wird lediglich die implementierende State-Klasse ausgetauscht. In unserem Beispiel könnten wir folgendes State-Interface anbieten:

- *+ getUIComponents()* liefert eine Liste der darzustellenden Komponenten;
- *+ isStepComplete()* gibt an, ob der Schritt abgeschlossen ist;
- *customStatusInfo* sind benutzerdefinierte Informationen.

Wenn wir nochmals zum ersten Ansatz mit den einzelnen Views zurückgehen und das State Pattern anwenden, enden wir wiederum bei *n+1* Controllern für *n* Zustände. Die Controller für die Zustände müssen aber nun das oben definierte Interface implementieren. Die Views ersetzen wir durch eine neue View, welche beim jeweiligen State-Controller mit *getUIComponents()* abfragt, welche UI-Komponenten angezeigt werden sollen.

Die Methode *isStepComplete()* und die Statusinformationen sind auch in den beiden vorherigen Ansätzen vorhanden, wir haben sie hier ledig-

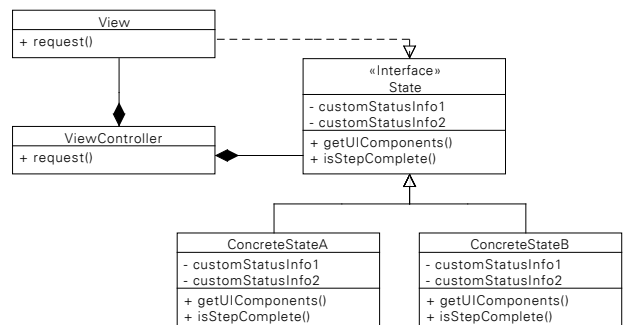


Abbildung 4: Klassendiagramm des implementierten State Patterns

```
// Header-File UITableViewDataSource
- tableView:cellForRowAtIndexPath: // required method
- numberOfSectionsInTableView:
- tableView:numberOfRowsInSection: // required method
- tableView:titleForHeaderInSection:

// Header-File UITableViewDelegate
- tableView:heightForRowAtIndexPath:
- tableView:didSelectRowAtIndexPath:
```

Listing 1a: Auszug aus dem Header-File von UITableViewDataSource. Das DataSource-Objekt ist für die Inhalte der TableView verantwortlich und bietet dazu unter anderem die Methode tableView:cellForRowAtIndexPath: an, welche die angegebene Zelle der Tabelle tableView zurückgibt.

Listing 1b: Die zwei wichtigsten Methoden aus dem Header von UITableViewDelegate. Das Delegate-Objekt ist für die Darstellung und die Interaktion verantwortlich. Es liefert unter anderem die Höhe für jede Zelle (tableView:heightForRowAtIndexPath:) und reagiert auf Benutzereingaben (tableView:didSelectRowAtIndexPath:)

lich formalisiert. Abbildung 4 zeigt das erweiterte Klassendiagramm.

Wie schlägt sich der dritte Ansatz im Vergleich zu den beiden anderen Ansätzen? Die Vorteile des dritten Ansatzes sind:

- Die Logik des Automaten ist zentral im Controller des Assistenten implementiert, womit die Implementation der Zustände einfach gehalten werden kann.
- Die Views lassen sich wiederverwerten (z. B. durch vordefinierte Components-Sets für *get-UIComponents()*).

Wir handeln uns als direkte Konsequenz aber mindestens das Problem der erschwerten Testbarkeit der Benutzeroberfläche ein, da die Views erst zur Laufzeit erzeugt werden. Zudem bestehen auch bei dieser Lösung noch technische Herausforderungen:

- Wie kann eine solch flexible View implementiert werden?
- Wie kann die View bei einem Zustandsübergang neu gezeichnet werden?

Während die Gewichtung der einzelnen Aspekte der vorliegenden Lösungsansätze von Plattform zu Plattform und von Problemstellung zu Problemstellung variieren kann, erscheint uns die Lösung mit Hilfe des State Patterns auf Grund der vielen ähnlichen Zustände am sinnvollsten.

Im nächsten Abschnitt werden wir genauer auf die Umsetzung dieser Variante auf iOS eingehen. iOS bietet Möglichkeiten, welche die beiden zuvor genannten Herausforderungen elegant lösen lässt. Im Grundsatz kann eine solche Lösung aber auch auf Android oder Windows Phone angewandt werden.

```
#pragma mark - Table view data source
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}
#pragma mark - Table view data source
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    return 1;
}
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section {
    return @"Request Code";
}
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    // #1
    PairingRequestCodeCell *requestCodeCell = (PairingRequestCodeCell *)
[tableView dequeueReusableCellWithIdentifier:@"RequestCodeCell" forIndexPath:indexPath];
    // Konfiguration der Zelle
    return requestCodeCell;
}

#pragma mark - Table view delegate
- (void)tableView:(UITableView*)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    // Wird hier nicht benötigt.
}
```

Listing 2: Auszug aus einem View Controller State



Abbildung 5: Der erste Zustand in unserem Pairing-Assistenten. Der Benutzer wird aufgefordert einen Pairing-Code einzugeben.

Als Beispiel soll uns die entwickelte App für Störmeldesysteme dienen. Wir werden aber nicht näher auf ihre Funktionalität eingehen.

Der Einsatz einer *UITableView* für ein beliebiges UI ist keine derart abenteuerliche Wahl und wird in vielen iOS-Apps verwendet. Wir verweisen auf die Mail-App für den Posteingang, die Kalender-App für die Stunden in der Tagesansicht oder die Nachrichten-App für den Verlauf der Konversation, um nur die offensichtlichsten Beispiele zu nennen. Der Benutzer nimmt diese Table Views unter Umständen gar nicht als Table Views wahr.

Stateful UITableView

Wir beginnen gleich bei der wichtigsten Komponente unseres Assistenten: der *UITableView*. Eine *UITableView* ist eine Liste von Einträgen, also eine einspaltige Tabelle.² Eine *UITableView* verfügt über zwei wichtige Referenzen: eine auf eine *DataSource* und eine auf ein *Delegate*. Beide Referenzen zeigen auf Objekte, welcher der *TableView* zur Laufzeit Verhalten hinzufügen.

Es dürfte nun klar werden, weshalb sich eine *TableView* gut zur Darstellung eines Assistenten eignet. Einmal erzeugt bezieht sie ihre Inhalte aus der *DataSource* mit Hilfe der Methode *tableView:cellForRowAtIndexPath:* aus Listing 1a und verhält sich wie in der *Delegate*-Methode *tableView:didSelectRowAtIndexPath:* implementiert (Listing 1b). Es reicht also aus die Implementierenden Klassen der beiden Interfaces³ *UITableViewDelegate* und *UITableViewDataSource* auszutauschen um Inhalt und Verhalten der *TableView* komplett zu ändern. Folglich sind es nun auch diese beiden Klassen, welche wir in verschiedene Zustände implementieren möchten.⁴

² Ihr Name ist eine Anlehnung an ihren grossen Bruder *NSTableView* auf OS X, welche mit mehreren Spalten umgehen kann.

³ Bei *UITableViewDataSource* und *UITableViewDelegate* handelt es sich genau genommen um Objective-C-Protokolle. Diese Unterscheidung ist für uns aber nicht relevant, weshalb wir bei der allgemein besser bekannten Bezeichnung Interface bleiben[TV].

⁴ Die hier vorgestellten Ideen lassen sich auch auf eine *UICollectionView* anwenden.

```
@interface PairingViewController :
    UITableViewController

@property (nonatomic, strong)Pairing
    *pairing;

@property (nonatomic)BOOL isEstablishing;

@property (nonatomic, strong)
    PairingViewControllerState *state;

@end
```

Listing 3: Header des Container View Controllers

Wir betrachten zur Veranschaulichung der Verwendung von *Delegates* und *DataSources* einen Auszug aus dem State, welcher einen Pairing-Code entgegennimmt. Wie in Abbildung 5 ersichtlich besteht die *TableView* aus einer *Section* mit einer einzigen Zelle. In Listing 2 ist der entsprechende Code ersichtlich.

Wie in Listing 2 ersichtlich, ist die Implementierung eines View Controller States sehr übersichtlich. Wir benötigen lediglich drei Methoden aus *UITableViewDataSource* und eine aus *UITableViewDelegate*. Auch wenn komplexere Schritte im Assistenten mehr Aufwand erfordern, so bleibt die Grundstruktur dennoch bei jedem State dieselbe. Spezielle Erwähnung verdient der Code-Ausschnitt #1: dort wird eine mit dem Interface Builder grafisch erstellte Zelle erzeugt. Somit erfüllt diese Lösung auch eine weitere unserer Anforderungen, die Wiederverwendbarkeit von UI-Elementen. Dieses Vorgehen schont ausserdem die beschränkten Systemressourcen des Smartphones, da von einem Zellentyp nur so viele Instanzen erzeugt werden, wie angezeigt werden können. Scrollt der Benutzer nach unten, werden die nicht mehr sichtbaren Objekte rezykliert.

Als Container für unsere View dient ein View Controller. Dieser muss in der Lage sein, den Automaten auf Grund der Zustandsinformationen zu durchlaufen und dabei die korrekten Informationen anzuzeigen. In Listing 3 ist der Header eines Container Controllers angegeben. Wie die erste Zeile

```
@interface PairingViewController :
    UITableViewController
```

bereits zeigt, ist der Container View Controller von *UITableViewController* abgeleitet und erbt somit auch dessen Properties *dataSource* und *delegate*. Bei *state* handelt es sich um eine Referenz auf den aktuell gültigen Zustand. *isPairing* und *pairing* sind die angesprochenen Zustandsinformationen. Sie sind in unserem Fall ausreichend, um in jedem Fall den korrekten nächsten Zustand anzusteuern.

```

- (void)setState:(PairingViewControllerState *)state {
    _state = state; // _state bezeichnet das Property state aus dem Header
    // #1
    self.tableView.delegate = _state;
    self.tableView.dataSource = _state;
    [self.tableView reloadData];
}

```

Listing 4: Die Methode `setState:` führt eine Zustandsänderung aus.

Um den aktuellen Zustand zu ändern, ist es ausreichend das Delegate und die DataSource der TableView zu aktualisieren und die TableView zu aktualisieren. Listing 4 zeigt die Implementierung des Setters von `state` in der Klasse des Container Controllers.

Wir haben nun fast alle Elemente zusammen. Es bleibt die Frage zu klären, wie der Container erfährt, dass er eine Zustandsänderung ausführen soll, das heisst, dass er die Methode `setState:` (Listing 4) ausführen soll. Dies lässt sich in Objective-C beispielsweise mittels Key-Value-Observing [KVO] lösen. Dazu ergänzen wir die Methode `setState:` an der Stelle #1 mit dem folgenden Aufruf.

```

[_state addObserver:self
 forKeyPath:@"stateComplete"
 options:0 context:NULL]

```

Hier senden wir dem aktuellen Zustand `_state` die Message `addObserver:forKeyPath:options:context:` mit den benannten Parametern `forKeyPath`, `options` und `context`. Als Argumente geben wir eine Referenz auf uns selbst (`self`) und den zu observierenden Property-Name als String an. Wir benötigen keine Optionen und keinen unterscheidbaren Kontext, weshalb wir diese Parameter nicht setzen.

Nun wird der Container bei jeder Änderung des Property `stateComplete` benachrichtigt und kann, falls der neue Wert `YES` ist, die Zustandsinformationen auswerten und dann die Methode `setState:` mit dem entsprechenden Zustand aufrufen.

Somit haben wir alle Anforderungen der Problemstellung gelöst, ohne die Nachteile der anderen Ansätze in Kauf zu nehmen. Konkret haben wir eine einfache Methode gesehen, um Zustandsänderungen auszuführen, wir können grosse Teile des UI an verschiedenen Stellen einsetzen und dennoch sind alle implementierten Klassen so knapp, dass die Übersicht nicht verloren geht. Des Weiteren können wir neue Zustände hinzufügen und müssen dabei lediglich die Zustandslogik entsprechend erweitern.

Zusammenfassung

Wir haben nun einen kleinen Einblick in die Entwicklung eines zustandsbasierten UI auf dem iPhone erhalten. Wir haben weder Bibliotheken von Drittherstellern, noch sonderlich fortschrittliche Funktionen der iOS-Plattform eingesetzt. Die verwendeten Funktionen werden in den meisten iOS-Apps angewendet. Dies gilt auch für Key Value Observing, obwohl einigen Entwicklern vermutlich nicht bewusst ist, dass dieses Prinzip unter der Haube für viele Annehmlichkeiten der Plattform verantwortlich ist.

Referenzen

- [KVO] Key-Value Observing Programming Guide/ Introduction to Key-Value Observing Programming Guide: <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/KeyValueObserving/KeyValueObserving.html>, 07.11.2013.
- [TV] Table View Programming Guide for iOS/ About Table Views in iOS Apps: http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/tableview_iphone/AboutTableViewsiPhone/AboutTableViewsiPhone.html, 07.11.2013.

Teaching Agile Software Development at University Level¹

Several recent surveys show that agile methodologies like Scrum, Extreme Programming and Kanban have been successfully adopted by many companies for software development. However, the same surveys show that only few of the agile practices are applied consequently and thoroughly. This is to a great extent due to the lack of skilled personnel. In this paper we propose a more holistic approach for teaching agile software development, in which the required agile practices and values are not only integrated theoretically into our courses but also practically applied. The proposed concept was realized in a new course at Zurich University of Applied Sciences during 2012. The evaluation shows very encouraging results.

Martin Kropp, Andreas Meier | martin.kropp@fhnw.ch

Recent surveys [1,2] show that agile methodologies in many respects deliver better outcomes than plan-driven ones. As a result, agile software development has been adopted by many IT companies and IT departments. In the Swiss Agile Study (SAS), a survey conducted by the authors, these findings have been confirmed [2]. More than half of the participating companies are using an agile methodology like Scrum [14] or XP [15] – Agile has become mainstream!

Unfortunately, this also has a significant impact on the agile team constitution. The early adopters of agile approaches were all highly mature and technically skilled experts in their fields. They had internalized the agile philosophy, were very productive and produced high quality results. Today's agile teams, however, are "normal" software teams, with architects, seniors and juniors in one team, and many of them are not yet familiar with the agile philosophy. Even though those teams have improved in software development to some extent, they are far less productive than the early adopter expert teams. Survey results show that quality has partially even gone down and overall costs increased. One reason for this may be that many of the important agile practices are not applied as thoroughly as the agile pioneers proposed [13].

In this paper we will analyze the situation on the industry side in more detail to find out, which skills are missing and make a proposal how education on university level can help to improve this. We will suggest a holistic teaching approach, which integrates the necessary agile engineering and managements skills together with the core agile values, into the education of agile software development.

In the next two sections we give an overview of related work to set our paper in context and

we analyze the reasons for the rather poor performance of today's agile teams and contrast this with the current state of software development education. Then we present the Agile Competence Pyramid as a model for the required competences for agile software development. In the rest of the article we present the layout and the evaluation of a new Software Engineering course, which was held at the Zurich University of Applied Sciences in the 5th semester of the undergraduate Computer Science program. We conclude with an outlook on further work.

Related Work

Though agile software development has been around for more than a decade (even before the famous Agile Manifesto [13]), teaching agile software development has only drawn some attention in educational and research conferences in the last few years. A reason for this might be that agile development is not based on a green-field theory but has been developed from practice. In [3] the authors discuss reasons why software engineering programs should teach agile software development. They emphasize that software engineers not only need technical skills but also social and ethical ones, which are both corner stones of agile development. In [4] the authors emphasize that theoretical lectures about agile development are not enough, but that students have to apply agile methods to really internalize them. The authors present a case study with 80 students working on a large project. There are several recent case study papers and experience reports [5-8] in which the authors report about their experiences teaching agile software engineering courses.

Motivation

The recent Swiss Agile Study, in which 140 Swiss IT companies and almost 200 IT professionals participated, shows very clear results. IT com-

¹ The original version of this paper has been published at the 26th Conference on Software Engineering Education and Training, May 19–21, 2013. CSEE&T '13, San Francisco, USA.

Aspect	much worse	worse	unchanged	improved	significantly improved	don't know
Changing priorities	1%	0%	9%	45%	44%	1%
Development process	0%	2%	17%	58%	22%	1%
Time to market	1%	2%	19%	53%	23%	2%
Alignm.btw. IT and business	0%	1%	25%	46%	23%	6%
Project visibility	0%	2%	25%	39%	28%	6%
Team morale	0%	4%	25%	42%	24%	5%
Requirements management	0%	2%	29%	51%	13%	5%
Productivity	0%	2%	33%	47%	15%	4%
Risk management	0%	5%	32%	42%	17%	4%
Software quality	0%	2%	45%	35%	16%	2%
Software maintainability	0%	7%	55%	23%	12%	3%
Development cost	1%	12%	52%	22%	7%	6%
Engineering discipline	0%	4%	42%	42%	9%	4%

Table 1: How has agile software development influenced the following aspects?

panies and IT professionals following the agile methods are much more satisfied with their methodologies than their plan-driven counterparts. The study also shows very clearly, that major goals of introducing agile development have been reached: A significant improvement in the ability to manage changing priorities, improvement of the development process in general and a much faster time-to-market.

Table 1 summarizes the influence of agile software development as given by the participating agile IT companies. Though the survey shows very promising results at first view, there are also quite astonishing findings. It is reported that development cost, software quality and software maintainability have not improved as much as expected. With respect to development cost and software maintainability, 7%, respectively 12% of the participants reported that these have even got worse. This clearly contradicts the intention of the authors of the agile manifesto, who want to deliver high quality code that is easily maintainable.

One reason for this might lie in the fact that only few of the agile practices are used consistently throughout the whole software development process. While engineering practices like coding

Items	completely disagree	disagree	agree	completely agree
Computer Science graduates (M.Sc.) have sufficient knowledge of agile methodologies	5%	53%	33%	9%
Computer Science undergraduates (B.Sc.) have sufficient knowledge of agile methodologies	8%	60%	28%	4%

Table 2: Knowledge of graduates

Items	completely disagree	disagree	agree	completely agree
Agile development should be an integral part of the Computer Science curriculum	0%	5%	49%	46%
Agile should not be taught at university, it is better learned on the job	34%	48%	12%	7%

Table 3: Agile as part the computer science curriculum

standards, unit testing or automated builds are used by two-third or more of the agile companies, other necessary practices like continuous integration, refactoring, test-driven development are used by only half the participants or even less. A similar result is obtained with respect to the management practices: while two-third or more of the participants use iteration planning, release planning or user stories, only half or even less of the participants use daily standups, task boards or retrospectives.

The SAS shows, that there are too few software engineers with the skills for agile development. This suggests that we as teachers do not yet educate the students with the required skills. This assumption is backed by answers in Table 2. Almost 70% percent of the participating companies think that undergraduates have too little knowledge of agile; still the majority thinks this is true for graduates.

Table 3 answers the questions, whether agile development should be an integral part of the computer science curriculum. The majority of the participants recommend that agile software development should be an integral part of the computer science curriculum. As educators, we have to take the findings from the above tables seriously and try to make sure that future graduates will have sufficient knowledge of agile methodologies.

Evaluation of Effort and Learning Effect

Two major characteristics of agile software development are its focus on working software over

		far too high	too high	exactly right	too little	far too little
Lecture	Documentation	4%	25%	36%	23%	12%
	Management	2%	16%	45%	27%	11%
	Programming	4%	9%	50%	27%	10%
Project	Documentation	32%	38%	27%	3%	1%
	Management	18%	31%	40%	10%	1%
	Programming	2%	8%	40%	34%	16%

Table 4: How do you estimate the effort for the different activities?

documentation and lightweight management. Therefore, the authors wanted to know how much effort computer science student spend on programming, management and documentation in the lectures and in their student projects, and about the learning effect they got from these activities. Table 4 and Table 5 show the results of an evaluation the authors conducted from 103 students at the two Universities of Applied Sciences in Zurich and Northwestern Switzerland.

Table 4 shows that the students estimated the effort spent for the different activities in the lectures more or less appropriate. In the student project, however, the majority of the students estimated the effort spent for documentation and management too high or even far too high.

Table 5 shows that the students estimated the learning effect of management activities significantly higher in student projects than in lectures. Interesting is the result for the documentation activity. The learning effect for software project documentation was seen to be much lower in the student project than in lectures. Setting this in relation to the results from Table 4 might suggest that the wrong style of documentation was taught in the student project.

The perceived results of this evaluation support the authors' hypothesis, that too much time is spent on agile management practices and, even worse, on documentation. The strong focus on documentation might come from the still existing influence of plan-driven methodologies.

Pyramid of Agile Competences

Before developing a new agile software engineering course, it is important to analyze the needed

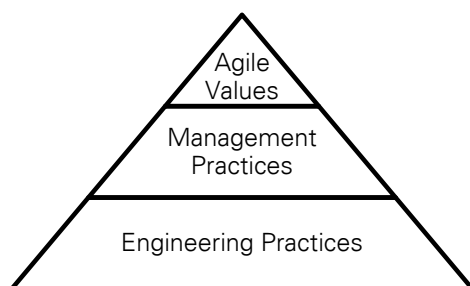


Figure 1: Pyramid of agile competences

		very high	high	low	very low
Lecture	Documentation	0%	22%	47%	31%
	Management	2%	29%	51%	18%
	Programming	18%	45%	24%	14%
Project	Documentation	8%	29%	51%	12%
	Management	10%	39%	38%	13%
	Programming	18%	42%	28%	12%

Table 5: How do you estimate the learning effect?

skills and competences for agile software development. The required competences can be divided into three major categories:

Mastering the technical skills or engineering practices, builds the foundation for being able to develop high quality software. These engineering practices are especially defined by eXtreme Programming and include best practices like unit testing, clean coding, test-driven development, collective code ownership and the like. Engineering practices are mostly competences that refer to the single individual.

On the second level come the agile management practices. They define how agile projects are organized and run. Agile management practices include iterative planning, short release cycles, small releases, strong customer involvement and highly interactive teams. Management practices are typically team aspects, which require the appropriate social competences.

On top of these competences come the agile values, which are articulated in the agile manifesto and are based on characteristics like mutual respect, openness, and courage. Figure 1 visualizes the required competences in an Agile Competence Pyramid.

The pyramid visualizes the decreasing number of required skills from bottom to top. On the other hand, it reflects the increasing difficulty to teach these skills. Engineering practices can be taught very well in the classroom through lecturers and be learned by the individuals at their own pace. Management competences are best taught through student projects in teams, as our student evaluation confirms. The most difficult competences to teach are the values on top of the pyramid, since they often require a change in the attitude of the individual.

These different competence levels have to be considered in an agile software engineering course and have guided the authors in the design of the new course.

Agile Software Engineering Course

The course was a typical 16-week semester class in the last year of the undergraduate level (B.Sc.). The students completed one Java programming

project in an agile team of six to eight members during the course of the semester. Per week there were a two hours lecture with the whole class and a two hours programming workshop with half the class. 27 students were enrolled.

The scope of the course was equivalent to four ECTS credit points (European Credit Transfer and Accumulation System, one credit point is equivalent to 30 hours of studying). The course consisted of lectures (32 hours), workshops (32 hours), and self-study including programming (56 hours).

The authors have successfully used a Scrum-XP-hybrid for many years and therefore decided to use it in this course as well. Why do Scrum and XP work well together? Scrum focuses on management practices while XP focuses mostly on engineering practices – they address different areas and complement each other.

Layout of the New Software Engineering Course

Table 6 shows the layout of the course. The course was divided into two parts of equal length and was designed with the insights from the previous chapters in mind. The two parts reflect the competence pyramid in Figure 1. Part one (weeks 1 to 7) lays the focus on building a strong foundation, i.e. the engineering practices. Part two (weeks 8 to 14) builds the second and third layer of the pyramid, i.e. the management practices and values. All practices were actively applied in a student project during part two.

For this course, the following learning target was defined using Bloom's taxonomy [19]: "After successfully attending this course, students have the necessary skills to develop software in an agile team. They can apply the most important agile engineering- and management practices and understand the importance of the agile values."

Part One: Applying Engineering Practices

- *eXtreme Programming (XP)*: In the first two lectures the students were given an introduction to XP. The XP practices and the Agile Manifesto were discussed. In the workshops, each student completed a coding assessment and was given feedback.
- *Version Control*: As a preparation for Continuous Integration, the concept of a version control system (VCS) was introduced. Subversion (SVN) was used as repository in the workshop. Some students suggested that GIT should rather be used than Subversion.
- *Project Automation*: Ant (Another neat tool) build scripts were introduced in the lecture and practiced in the workshops. Some students suggested using Maven instead of Ant build scripts.
- *Continuous Integration (CI)*: With version control and project automation in place, the con-

Week	Lecture	Workshop
1	eXtreme Programming Agile Manifesto	Installation IDE & Plug-Ins Coding Assessment 1
2	eXtreme Programming Version Control	Coding Assessment 2 Version Control Syst. (SVN)
3	eXtreme Programming Project Automation	Build Scripts (Ant)
4	Continuous Integration	CI (Jenkins Build Server)
5	Unit Testing	JUnit
6	Unit Testing / Mock Objects Clean Code/ Code Smells	JUnit EasyMock
7	Refactoring	Refactoring
8	Introduction to Test-Driven Design / Scrum	TDD, The Craftsman articles
9	Scrum	Agile Game Development (Sprint 1)
10	Scrum	Agile Game Development (Sprint 2)
11	Agile Estimating and Planning Planning Poker	Agile Game Development (Sprint 3) Agile
12	Metrics Agile Teams	Agile Game Development (Sprint 4) Metrics (EMMA)
13	User Stories Agile Principles	Agile Game Development (Sprint 5)
14	Demonstration of computer games	Agile Game Development (Sprint 6)
15/16	Preparation for examination: No lecture	Preparation for examination: No workshop

Table 6: Overview of semester plan

cept and benefits of CI were discussed. In the workshop, a CI-server Jenkins was configured.

- *Clean Code and Code Smells*: Clean code has had a marvelous effect on the quality and readability of student's code [9,10]. The students read most of the Clean Code book as part of the self-study.
- *Unit Testing and Mock Objects*: The concept of automatic unit testing was introduced. In the workshop, exercises with JUnit and EasyMock were carried out. These JUnit tests were added to the CI-server.
- *Refactoring*: Good understanding of automatic unit testing and refactoring are the basis of Test-Driven Design. A catalog of refactorings was discussed and practiced in the workshop.
- *Introduction to Test-Driven Design (TDD)*: "TDD is hard. It takes a while for a programmer to get it." [17]. TDD is especially difficult to teach in the classroom. For that reason, the students were only given an introduction to TDD. In the workshop, the students worked through some of the craftsman articles [18]. One student gave

Items	excel- lent	good	bad	very bad
The content of this course is...	12	11	0	0
This course was divided into engineering- and management practices and agile values. How would you judge this concept?	12	11	0	0
How did the agile values come across in the lectures and workshops?	1	19	1	0
In the student project, you worked in a Scrum team of 6 to 8 fellow students. How would you judge this concept?	9	11	4	0
How would you judge the workshops in part one?	1	20	1	0
How would you judge the workshops in part two?	6	14	3	0

Table 7: Course evaluation

the following feedback: “Reading the craftsman articles really helped me to understand how TDD works.”

Part Two: Applying Management Practices

- *Student project*: While the students were working individually or in small groups in part one, part two was different - the agile game was played in the classroom. In order to really understand how Scrum works, the students must be members of a “real” Scrum team. Since this is not possible in the classroom, the Scrum team was simulated in the student project. The goal of the student project was to develop a 2D computer game applying all needed engineering practices. The students worked in four Scrum teams of six to eight. Each team was free to decide what kind of computer game they wanted to develop. One student was voted ScrumMaster; the lecturer was the product owner. The teams completed six one-week sprints. Every week during the workshops, each team did the sprint planning, sprint review and retrospective coached by the lecturer. During self-study, the students developed the actual game. In the last week, all the teams could demonstrate a working game. In order to get a good start, the students were given an introduction to game development with Slick2D [20].
- *Scrum* was introduced in the lecture. Problems and questions, which had arisen in the Scrum teams were addressed in the next lecture and discussed in the plenum.
- *Pair Programming* was introduced ad hoc. The students were asked to pair with peers while developing the game.
- *Planning Poker*: Agile estimating and planning was introduced during the lecture and

Items	Yes	No
Would you recommend this course to your fellow students?	23	1
Did you enjoy this course?	20	0

Table 8: How did you like the course?

“... the development of the computer game in a Scrum team.”
“... that the material in the course was not only covered theoretically but I also had the opportunity to apply and deepen it in the workshops.”
“... the practical relevance.”
“... that the topics covered were interesting and important. I had the opportunity to practice the newly learned in the student project. That was great!”

Table 9: What did you like best about the course?

practiced in the Scrum teams [11]. User stories were estimated by playing planning poker [12].

- *Task board*: The functioning of the task board and burndown charts were discussed. For this course, an electronic task board was used.

Teaching Agile Values

Agile values are difficult to teach [13]. The approach in this course was to show the students, that these values are not just something the creators of the Agile Manifesto intended to give lip service to and then forget. They are working values. The concepts of agile values were introduced in the first part. Usage of the values was propagated in the second iteration through means like retrospectives, common code ownership or pair programming. Many discussions during the lectures and workshops tried to transport that message.

Student Feedback

In the last week of the semester, 24 students filled in an evaluation form (the items and answers are translated from German). An excerpt of the encouraging results is shown in Table 7.

In the planning phase there was some uncertainty as to whether the student project would falter due to group size and commitment of the individual member of the scrum teams. These fears were ungrounded. On the contrary, the students were exceptionally committed and delivered top quality computer games. The students were asked what they liked most about the course. In Table 9 are some statements, translated from German. The students were also asked what they disliked about the course. Nine students did not have any dislikes. Most of the students disliked the amount of work during the student project in the second

part. Many students suggested that the student project should be longer (see Table 10).

Evaluation and Suggestions

The quality of the students' work was measured twofold. On the one hand, the student project presentations, which included a demonstration of their computer games, were evaluated. On the other hand, the students had to pass a formal oral exam. The average grade was a 5.1 on a scale from 1 (very poor) to 6 (excellent). This was higher than expected. A systematic classification of the outcome quality remains to be done.

The experience from this course and input from students lead to the following suggestions:

Group dynamics are very important and therefore special attention should be paid to the way the Scrum teams are put together. The students should have access to a room, where they can meet for standups and have a wall for the task board. For this course, an electronic task board was used. Unfortunately, because of poor performance it did not meet our expectations.

Working only a couple of hours every week on the student project is not ideal. Many students suggested an intensive week instead. During this week, the students would only work on the project in the Scrum team. One semester is rather short for the material covered in this course. If the students had been familiar with engineering practices like unit testing, refactoring, build automation or clean code prior to the course, this time could have been used for test-driven development or additional iterations.

Further Work

Advanced practices like Behavior Driven Development (BDD) or Acceptance Test Driven Development (ATDD) were not covered in this course. Because of limited time, only an introduction to Test-Driven Development could be taught. Testing is a very important topic and should therefore be deepened in future courses. The same is true for requirements engineering, which was only partly covered in this course.

It is the authors' opinion that agile software development cannot be taught in isolated Software Engineering courses. A challenge will be the integration of agile development in other courses like programming, object-oriented analysis and design, algorithms and data structures, etc. Special attention needs to be paid to the fact, that agile software development does not work well together with big-design up front (BDUF) approaches. This could mean a shift from BDUF to emergent design as advocates of Scrum propose it. That said, further work is necessary on how agile development can successfully be integrated into the computer science curriculum.

"... too much work during the second part"
"... too little time for developing the computer game"
"... agile was praised too much. Negative aspects of agile were not or too little mentioned"
"... the electronic task board"
"... too little time for the student project, because of simultaneous projects in other courses"

Table 10: What did you dislike about the course?

References

- [1] Version One. State of Agile Development Survey results. http://www.versionone.com/state_of_agile_development_survey/11/, 20.10.2012
- [2] Martin Kropp, Andreas Meier, Swiss Agile Study - Einsatz und Nutzen von Agilen Methoden in der Schweiz. www.swissagilestudy.ch, 20.1.2013.
- [3] Orit Hazzan, Yael Dubinsky: Why Software Engineering Programs Should Teach Agile Software Development. ACM SIGSOFT Software Engineering Notes 2007, Vol. 32/2.
- [4] Bernd Bruegge et al: Agile Principles in Academic Education: A Case Study. 6th International Conference on Information Technology: New Generations 2009. ITNG '09.
- [5] Vldan Devedzic and Sasa R. Milenkovic. Teaching Agile Software Development: A Case Study, IEEE transactions on Education Vol. 24. No 2. 2011.
- [6] Andreas Schroeder et al. Teaching Agile Software Development through Lab Courses. IEEE Global Engineering Education Conference 2012. EDUCON '12.
- [7] Viljan Mahnic. A Capstone Course on Agile Software Development Using Scrum. IEEE TRANSACTIONS ON EDUCATION, VOL. 55, NO. 1, 2012
- [8] Rico, D.F., Sayani, H.H. Use of Agile Methods in Software Engineering Education. Agile Conference, 2009. AGILE '09.
- [9] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 2009, ISBN 0-13-235088-2
- [10] Robert C. Martin, The Clean Coder: A Code of Conduct for Professional Programmers, Prentice Hall, 2011, ISBN 0-13-708107-3
- [11] Mike Cohn, Agile Estimating and Planning, 2006, ISBN 0-13-147941-5
- [12] Mike Cohn, User Stories Applied, For Agile Software Development, 2004, ISBN 0-321-20568-5
- [13] Agile Manifesto. <http://agilemanifesto.org/>, 20.1.2013.
- [14] Ken Schwaber, Mike Beedle. Agile Software Development with Scrum, 2001, ISBN 0-13-207489-3
- [15] Kent Beck, Extreme Programming Explained: Embrace Change. Addison-Wesley, 2004 ISBN 0-321-27865-8
- [16] Kent Beck, Test-Driven Development: By Example. Addison-Wesley, 2003, ISBN 0-321-14653-0
- [17] Henrik Kniberg, Scrum and XP from the Trenches. How we do Scrum. An agile war story, 2007, ISBN: 978-1-4303-2264-1
- [18] Robert C. Martin, The Craftsman, <http://www.objectmentor.com/resources/publishedArticles.html>, 20.1.2013.
- [19] Benjamin S. Bloom, David R. Krathwohl (1956). Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain, New York, Longmans, Green.
- [20] Slick2D - Open source 2D java game library: <http://www.slick2d.org>, 22.01.2013

Large-Scale Indoor Tracking

Im Projekt „Large-Scale Indoor Tracking in transitorischen Räumen“ schliessen sich drei Forschungsinstitute von drei verschiedenen Teilschulen der FHNW mit externen Künstlern zusammen, um gemeinsam neue Trackingverfahren für ein interaktives Kunstwerk zu entwickeln. Der daraus entstandene interaktive Soundwalk „LautLots“ wurde im September 2013 im Badischen Bahnhof in Basel uraufgeführt. Die technischen Zielsetzungen, Hintergründe und Ergebnisse dieses Soundwalks werden in diesem Beitrag beleuchtet.

Matthias Krebs, Thomas Resch, Christoph Stamm | christoph.stamm@fhnw.ch

Techniken des Verfolgens von sich bewegenden Objekten, so genannte Trackingverfahren spielen in verschiedensten Zusammenhängen in der FHNW-Forschung eine wichtige Rolle: Nutzt die Hochschule für Musik Infrarot-Tracking für die Performance elektronischer Musik, dienen im Institut für Design- und Kunstforschung (HGK) räumliches Tracking als Untersuchungswerkzeug in Stadt- und Siedlungsentwicklungsprozessen, als digitales Entwurfswerkzeug oder Ausgangsmaterial für künstlerische Projekte; das Institut für Mobile und Verteilte Systeme (HT) beschäftigt sich mit der Lokalisierung von mobilen Objekten/Personen und vor allem mit den darauf aufbauenden, kontextsensitiven Diensten in mobilen Geräten.

Im Projekt „Large-Scale Indoor Tracking in transitorischen Räumen“ schliessen sich die Forschungsinstitute der zuvor genannten Hochschulen mit externen Künstlern zusammen, um gemeinsam neue Trackingverfahren für ein interaktives Kunstwerk zu entwickeln. Inbegriff eines transitorischen Raums ist der Badische Bahnhof in Basel: als Raum auf einer Grenze (zwischen der Schweiz und Deutschland) und als Gebäude,

das man aufsucht, um es schnell zu verlassen. An einem solchen Ort können Transformationen von Kulturen und Lebenswelten wie durch ein Brennglas beobachtet werden. Im Projekt gilt es erstens diesen geschichtsträchtigen Grenzraum in Kunst und Technik neu zu erforschen und eine Vision seiner Zukunft zu gestalten und zweitens anhand des gewählten Fallbeispiels neue Mittel in Technik und Kunst zu entwickeln. Das Hauptergebnis dieses Projekts ist der Soundwalk „LautLots“: Ein akustischer Guide durch den Badischen Bahnhof, innerhalb dessen das Publikum in unmittelbare Interaktion mit Geschichte und Gegenwart des Badischen Bahnhofs tritt und diesen Grenzort neu erkunden kann. Der Soundwalk findet in der Zeit zwischen dem 8. und 13. September 2013 statt.

Soundwalk

In vorliegendem Bericht beschränken wir uns hauptsächlich auf die technischen Aspekte des Gesamtprojekts und geben Einblicke in die verwendeten Technologien und angewandten Techniken. Dennoch ist es wohl hilfreich, kurz darzustellen, was das künstlerische Ergebnis dieses

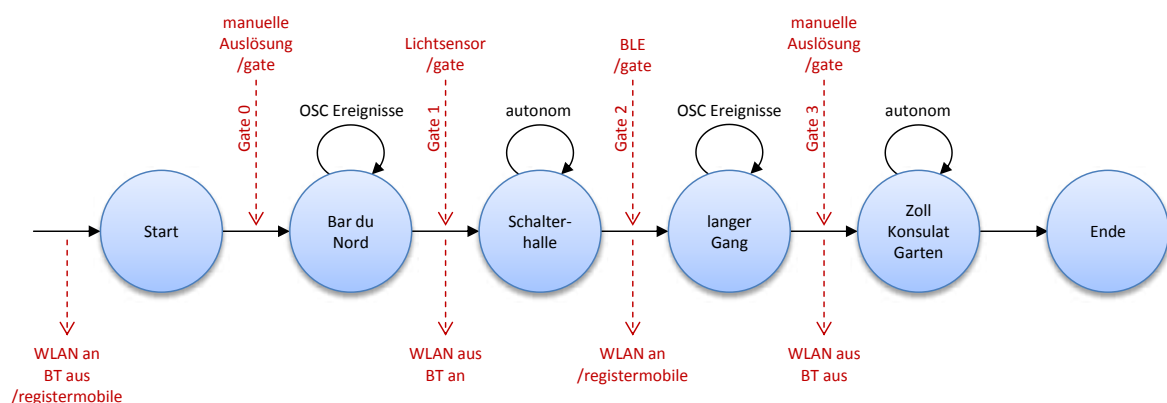


Abbildung 1: Zustandsmaschine zur Steuerung des Soundwalks „LautLots“ durch verschiedene Räumlichkeiten des Bad. Bahnhofs.



Abbildung 2: Head Mounted Device: Auf dem Bügel zwischen den Ohrmuscheln sind ein Smartphone und sechs Infrarot-LEDs angebracht. Der Kopfhörer ist ans Smartphone angeschlossen.

Projekts ist. Für weitere Einblicke sei auch auf den begleitenden Blog verwiesen [Blog].

Künstlerisches Ergebnis des Projekts ist der Soundwalk „LautLots“, der durch sehr unterschiedliche Räumlichkeiten des Bahnhofs führt (Abb. 1). Ausgerüstet mit *Head Mounted Devices* begeben sich die Teilnehmenden auf eine For-

schungsexpedition durch den Badischen Bahnhof, auf der sie Raum-, Orientierungs- und Navigationsexperimente durchführen. Das nur für die Teilnehmenden wahrnehmbare akustische Erlebnis hängt ab von deren Position im Raum sowie von deren Bewegungen und Verhalten zueinander. Der Soundwalk bringt Erkenntnisse über die vielgestaltige Topographie von Innenräumen, die Fernreisenden verschlossen bleiben. Er ist Hörspiel, interaktiver Audioguide und literarische Zimmerreise zugleich.

Head Mounted Device

Wer am Soundwalk teilnehmen will, muss den zur Verfügung gestellten Kopfhörer aufsetzen und schon kann es losgehen (Abb. 2). Nun, Kopfhörer ist ein wenig untertrieben, denn auf dem Bügel zwischen den Ohrmuscheln ist ganz schön viel Technik angebracht. Als erstes sticht das Smartphone ins Auge, welches unterschiedlichste Sensoren, die Tonwiedergabe und natürlich die Kommunikationsvorrichtung in sich vereint. Da der Kopfhörer ans Smartphone angeschlossen ist, werden offensichtlich die Stimmen und Klangwelten, welche in den Ohrmuscheln erschallen, auf dem Smartphone abgespielt oder sogar darauf produziert. Im Gegensatz zu einer Abspielliste von Musiktiteln, die in festgelegter oder zufälliger Reihenfolge gespeicherte Musikstücke abspielt, ist hier jedoch ein unsichtbarer „Regisseur“ am Werk, der die Abspielfolge auf virtuose Art lenkt. Mehr als das, der „Regisseur“ tritt in gleichzei-

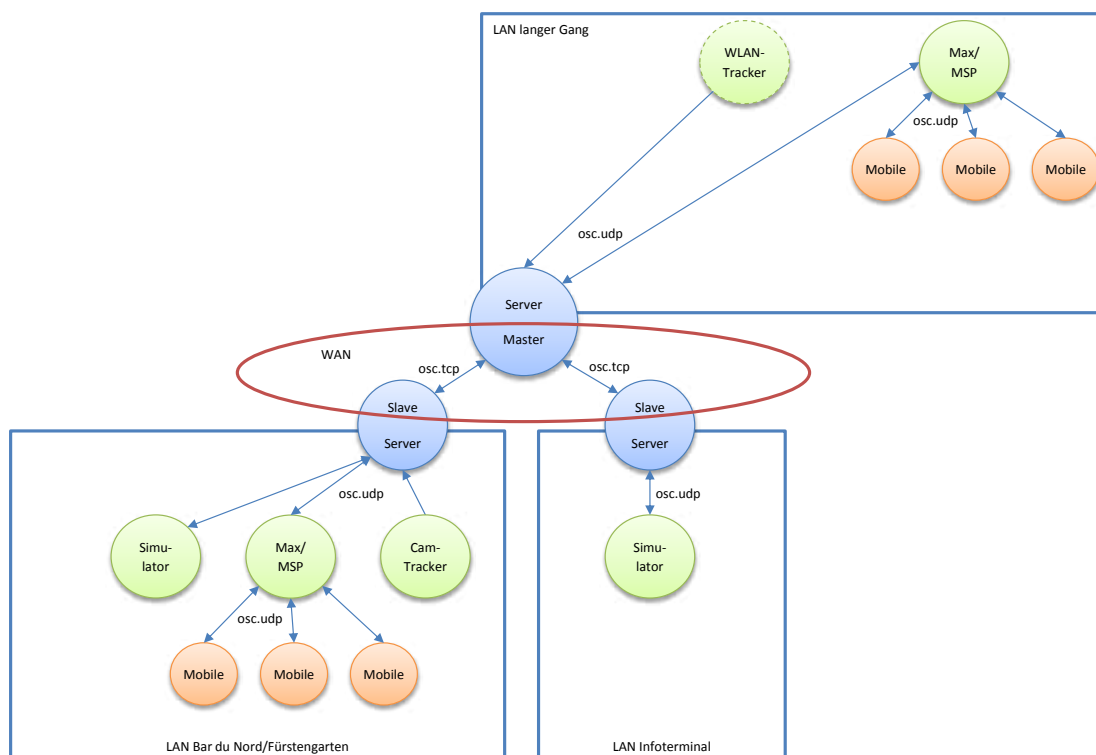


Abbildung 3: Kommunikationsstruktur. Im Zentrum befinden sich drei über das Internet miteinander verbundene Server, welche jeweils Teil eines separaten WLANs sind. Für die Kommunikation zwischen mobilen und stationären Clients und den Servern wird das leichtgewichtige und weitverbreitete OSC Protokoll verwendet.

tige und individuelle Dialoge mit den Trägern der *Head Mounted Devices* (HMD). Über eine drahtlose Netzwerkverbindung sendet der „Regisseur“ individuelle Kommandos an die Smartphones und löst Ton- und Klangwiedergabe aus. Mit Gesten wie Kopfnicken oder Kopfschütteln oder mit Bewegungen im Raum können dann die Träger der HMDs auf die Regieanweisungen reagieren und dem „Regisseur“ ein Feedback geben.

Während die Gesten auf dem Smartphone direkt mithilfe der eingebauten Sensoren und spezieller Software ermittelt und über ein Drahtlosnetzwerk an den „Regisseur“ gesendet werden, braucht es für die Bestimmung der Raumpositionen eine Lokalisierungstechnik analog zu GPS, die im Gegensatz dazu aber innerhalb von Gebäuden funktioniert.

Kommunikationskonzept

In drei Räumlichkeiten des Soundwalks sind unabhängige lokale Drahtlosnetzwerke (WLANs) in Betrieb, welche über Internetverbindungen miteinander verbunden sind (Abb. 3). In den WLANs befinden sich stationäre Server, die für die Kommunikation innerhalb des WLANs und für den Datenaustausch zwischen den verschiedenen WLANs verantwortlich sind. Infolge der geringen Anzahl Server kommt eine einfache Master/Slave-Architektur zum Einsatz, bei der sich die Slaves beim Master registrieren und aller Datenverkehr zwischen den Servern über den Master abgewickelt wird.

Der ganze Meldungsaustausch zwischen den Systemen basiert auf dem Open Sound Control (OSC) Protokoll, welches leichtgewichtig, auf verschiedensten Plattformen verfügbar und einfach handzuhaben ist. Ursprünglich ist OSC vor allem für den Austausch von Sound-Daten verwendet worden. Dabei spielen kleine Latenzzeiten eine wichtige Rolle und der Verlust reduziert eines einzelnen Datenpaketes reduziert zwar die Soundqualität, zerstört die Datenübermittlung aber nicht als Ganzes. Daher kommt bei OSC vorwiegend UDP als Trägerprotokoll zum Einsatz. Für spezielle Einsätze kann jedoch auch TCP verwendet werden, wobei nicht alle OSC-Implementierungen TCP gleich gut unterstützen. Die von uns eingesetzte OSC C-Bibliothek hatte beispielsweise noch Fehler im TCP-Teil und musste zuerst korrigiert werden.

Innerhalb der WLANs verwenden wir UDP als Trägerprotokoll für OSC. Da in einem WLAN häufig mal ein Datenpaket verloren geht, kann es schnell passieren, dass eine OSC-Meldung nicht ankommt. Für wichtige Meldungen wie zum Beispiel die Registrierung bei einem Server müssen daher spezielle Vorkehrungen getroffen werden, damit solche Meldungen mit hoher Wahrscheinlichkeit ankommen. Zwischen den Servern kommt TCP als Trägerprotokoll zum Einsatz. So wird sichergestellt, dass die Meldungen zwischen den einzelnen WLANs wirklich ausgetauscht werden und allfällige bestehende Internetverbindungen bzw. Router können unverändert weiterverwendet, da die Slaves zum Master eine TCP-Verbindung aufbauen und nur der Master-Server im Internet sichtbar und ansprechbar sein muss.

In den WLANs befinden sich verschiedene mobile und stationäre Clients, welche über OSC mit dem lokalen Server unidirektional oder bidirektional kommunizieren können. Die mobilen Clients dürfen sich zwischen den WLANs bewegen, müssen sich aber beim Eintritt in ein WLAN beim entsprechenden Server bzw. bei einem stationären Client anmelden. Dadurch können die anderen Server informiert werden, dass sie keine weiteren Meldungen an die entsprechenden mobilen Clients mehr senden sollen. Die stationären Clients übernehmen pro WLAN individuelle Aufgaben, wie beispielsweise die Lokalisierung der mobilen Clients (xTracker) oder die Regie des Soundwalks (Max/MSP). Für die stationären Clients dürfen eigenständige Computer eingesetzt werden oder sie dürfen auch als separate Prozesse auf dem gleichen Rechner wie der Server ablaufen. Auf die einzelnen Aufgaben der stationären Clients wird in weiteren Abschnitten des Artikels noch genauer eingegangen.

Kameratracking

Der Soundwalk beginnt in der Bar du Nord, dem ursprünglichen Buffet dritter Klasse des Badischen Bahnhofs. Der aufmerksamen Beobachterin fällt auf, dass an der Decke mehrere Kameras installiert worden sind, welche üblicherweise nicht vorhanden sind. Doch diese Kameras sind unauffällig und zudem beinahe blind, denn sie „sehen“ die normalen Barbesucher nicht. Es findet also keine klassische Videoüberwachung statt, bei der Videoaufnahmen auf Vorrat gespeichert oder direkt auf spezielle Vorkommnisse hin untersucht

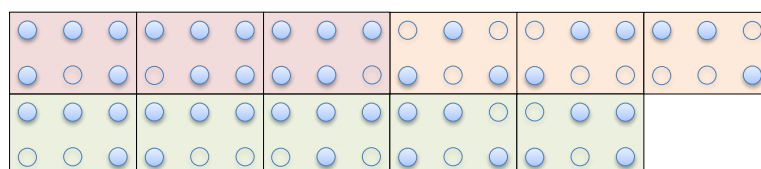


Abbildung 4: Rotationsasymmetrische Identifikationsmuster mit je 3 bis 5 Punkten angeordnet in einer regulären 2x3 Matrix. Die drei Farben grenzen die Muster mit gleicher Anzahl Punkte ab. In „LautLots“ sind die unteren fünf Muster verwendet worden.

werden. Die Kameras sind nur empfindlich für infrarotes und für uns Menschen unsichtbares Licht. Die Infrarotstrahlung wird von sechs weitwinkligen Infrarot-LEDs erzeugt, welche um das Smartphone herum auf dem HMD platziert worden sind. Die sechs rasterförmig angeordneten LEDs genügen, um mindestens elf verschiedene LED-Muster zu erzeugen, welche paarweise rotationsasymmetrisch sind und zudem eine eindeutige Ausrichtung haben (Abb. 4). Dadurch können mehrere Muster gleichzeitig verfolgt, analysiert und eindeutig einer Person zugeordnet und anhand der Musterausrichtung auch die Blickrichtung der Person ermittelt werden.

Bei den eingesetzten Kameras handelt es sich um handelsübliche, weitwinklige Überwachungskameras mit einer Auflösung von 1240x1024 Bildpunkten, welche auch in der Nacht im nahen Infrarotbereich betrieben werden können. Durch zusätzlich eingebaute Infrarotfilter werden störende Bildinhalte aus dem sichtbaren Spektrum herausgefiltert, so dass die von den HMDs ausgestrahlten Infrarotsignale ohne grosse Bildbearbeitung (Binarisierung genügt) analysiert und weiterverarbeitet werden können. Die Grösse des Raumes erfordert den Einsatz von mehreren Kameras, welche nebeneinander und mit vertikaler Ausrichtung unter der Decke angeordnet sind, so dass es zu schmalen Überlappungen an den Rändern kommt. Die Kameras verfügen über einen PoE-Anschluss, der sowohl für die Datenkommunikation als auch die Speisung verwendet wird. Die Videoströme werden über Ethernet zu einem GBit-Switch geführt und von dort zu einem Rechner geleitet, der problemlos fünf Videoströme mit 15 Bildern pro Sekunde parallel zu analysieren vermag. Dass die Prozessoren nur 15 Bilder pro Sekunde verarbeiten, liegt primär am Flaschenhals der Netzwerkverbindung zwischen Switch und Rechner. Die ermittelten Positionen und Ausrichtungen werden wie bei allen eingesetzten Clients mittels des OSC-Protokolls an einen Server gesendet, welcher die empfangenen Informationen an alle registrierten Datenkonsumenten weiterleitet. Dadurch lassen sich die Bewegungen der getrackten Personen in verschiedenen Räumen des Badischen Bahnhofs gleichzeitig verwenden bzw. visualisieren.

Der Einsatz von weitwinkligen, preiswerten Kameras führt oft dazu, dass die Bildeigenschaften am Bildrand nicht sonderlich gut sind, dass es zu stärkeren Verzerrungen kommt. Diese Verzerrungen sind vorgängig experimentell ermittelt worden und werden noch vor der Bildanalyse herausgerechnet. Dadurch wird zwar das Kamerasichtfeld etwas kleiner, aber die aufgenommenen Muster sind bis zum Rand hin gut sichtbar. Diese gute Sichtbarkeit hängt jedoch nicht nur von der Entzerrung des Kamerabilds, sondern in grösserer Masse auch von der Weitwinkligkeit (90°

der eingesetzten IR-LEDs ab. Nachdem die einzelnen Kamerabilder parallel analysiert worden sind, werden die Pixelkoordinaten der erkannten Muster mittels einer perspektivischen Transformation in Raumkoordinaten umgerechnet. Für diese Transformation ist vorgängig eine (interaktive) Ermittlung der Transformationsmatrix pro Kamera notwendig. Dabei verwenden wir vier im Bild sichtbare und bekannte Raumpunkte, welche im Kamerabild angewählt werden und daraus lässt sich die Transformationsmatrix ermitteln. Je genauer diese Transformationsmatrizen sind, desto geringer sind die Abweichungen zwischen den Raumkoordinaten, wenn zwei Kameras das gleiche bekannte Muster innerhalb eines Überlappungsbereichs detektieren.

Die eigentliche Musteridentifizierung ist ein mehrstufiger Prozess, welcher mit der Identifikation der LEDs im Bild beginnt. Aus den detektierten und vermeintlichen LEDs werden durch räumliche Nachbarschaft Clusters gebildet. Von Clustern mit relevanter Anzahl von Einzelpunkten werden dann charakteristische Merkmale bestimmt, um eine möglichst eindeutige Zuordnung zwischen entdeckten und erwarteten Clustern zur erreichen. Die Güte der einzelnen Zuordnungen nennen wir hier Relevanz. Diese Relevanz wird zusammen mit dem Schwerpunkt des Clusters und der räumlichen Ausrichtung des Musters an einen Controller übermittelt, welcher von allen Kameras die erkannten Muster zusammenträgt. Sollte das gleiche Muster von verschiedenen Kameras an ganz verschiedenen Raumpositionen entdeckt werden, so kann die Relevanz herangezogen werden, um die relevanteste Position zu ermitteln. Verschwindet ein Muster nur für kurze Zeit, das heisst, wenn ein Muster in wenigen Bildern hintereinander nicht mehr detektiert wird, so wird vorerst davon ausgegangen, dass sich das Muster in der zuletzt ermittelten Geschwindigkeit in Richtung seiner Ausrichtung bewegt. Sobald das Muster dann wieder sichtbar ist, wird die geschätzte Position durch die ermittelte ersetzt, andernfalls wird für das Muster keine Position mehr an den Controller übermittelt.

Als charakteristisches Merkmal verwenden wir die Sequenz der Innenwinkel der konvexen Hülle der Einzelpunkte eines Clusters. Da jedes von uns gewählte Muster einen eindeutigen grössten Winkel besitzt, lassen sich zwei Folgen in linearer Zeit miteinander vergleichen und der quadratische Fehler berechnen. Darüber hinaus dient der grösste Winkel auch zur Bestimmung der Ausrichtung, indem der Vektor vom Schwerpunkt des Clusters zum grössten Winkel verwendet wird.

Gates

Beim Verlassen der Bar du Nord in Richtung der Schalterhalle des Bahnhofs passieren wir ein mit einer speziellen LED-Leiste beleuchtetes Tor.

Situation	Lichtstärke [Lux]
Bar mit wenig Beleuchtung	50
Künstliche Beleuchtung, z.B. Schalterhalle	200 - 300
LED-Leiste in 30 cm Entfernung	3000
Tageslicht im Schatten	10'000
Tageslicht an der Sonne	85'000

Tabelle 1: Verschiedene Lichtsituationen mit entsprechenden Lichtstärken gemessen mit einem Samsung Galaxy S3

Diese Torbeleuchtung wird ähnlich einer Lichtschranke wahrgenommen und löst eine Reihe von Aktivitäten aus, unter anderem eine Abmeldung beim „Regisseur“ und die Deaktivierung der Drahtlosnetzwerkverbindung auf dem Smartphone. Das Smartphone verliert somit den Kontakt zum „Regisseur“ und befindet sich nun in einem autonomen Zustand. Nach wie vor hören wir die Stimme im Kopfhörer, welche uns durch die Schalterhalle zu einer unscheinbaren und verschlossenen Tür lotst.

Für die technische Umsetzung verwenden wir den Lichtsensor auf der Vorderseite des Smartphones, welcher normalerweise die Display-Helligkeit regelt. Unter Android wird ein Lichtsensor unter der Kennung *Sensor.TYPE_LIGHT* angesteuert. Der Sensor liefert einen ganzzahligen Wert zurück, welcher der Einheit Lux entspricht. Im Falle unseres Samsung Galaxy S3 liegt die Empfindlichkeit im Bereich zwischen 0 und 85'000 Lux. Sie ist stark richtungsabhängig; die besten Ergebnisse erzielt man, wenn sich die Lichtquelle fast senkrecht zum Smartphone-Display befindet. Die Tabelle 1 führt verschiedene Lichtsituationen und die gemessenen Lichtstärken auf.

Anhand dieser Beispiele zeigt sich, dass solche Messungen der Lichtstärke nicht mit dem subjektiven Empfinden übereinstimmen. Beispielsweise wird eine LED-Leiste bei direktem Blickkontakt bereits als gleissend hell empfunden, während ein Aufenthalt im Freien an einer Stelle ohne direkte Sonneneinstrahlung als deutlich weniger hell empfunden wird.

Um den Lichtsensor als eine Art Lichtschranke nutzen zu können, muss ein Lichtstärkenverlauf erzielt werden, welcher sich ausreichend von der Umgebung unterscheidet. Mit den von uns getesteten LED-Lichtquellen haben wir im besten Fall ca. 3500 Lux erreicht. Da in der recht düsteren Bar du Nord im Durchschnitt nur 50 bis 100 Lux gemessen werden, ist der Unterschied zu der am Türrahmen angebrachten LED-Lichtquelle ausreichend gross.

Beim Durchschreiten der Lichtschranke in normaler Gehgeschwindigkeit zeigt sich eine Lichtspitze, welche innerhalb einer Sekunde ihr Maximum erreicht und danach in derselben Zeit abfällt (Abb. 5). Nimmt man einen unteren Schwellwert von 1000 Lux und einen oberen Schwellwert von 3000 Lux, so ergibt sich bei einer Abtastrate von ca. 5 Hz ein Anstieg von

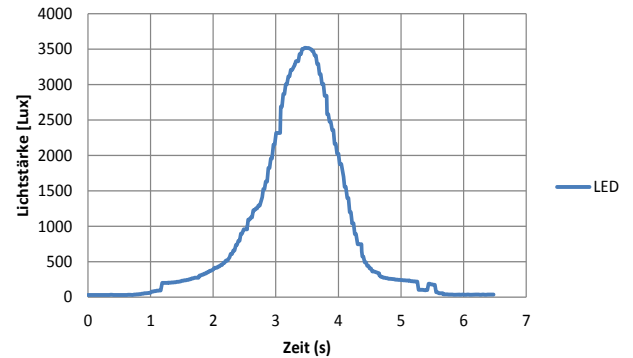


Abbildung 5: Messung der Lichtstärke beim Durchschreiten der Bar du Nord, an welcher eine LED-Lichtquelle angebracht worden ist.

ca. 400 Lux pro Messwert. Mit den Parametern für Schwellwerte und Flankensteilheit kann das Durchschreiten der LED-Leiste zuverlässig detektiert werden. Einziger Nachteil dieser Methode ist, dass im Bereich eines Fensters einfallendes Tageslicht, welches problemlos 3000 Lux oder mehr erreichen kann, eine Fehldetektion auslösen kann. Aus diesem Grund kann eine solche einfache Detektionstechnik nur in Räumen eingesetzt werden, welche gleichmässig und mit künstlichem Licht beleuchtet sind.

Beim Übergang von der Schalterhalle in den langen Gang wird ebenfalls ein Gate benötigt. An dieser Stelle muss die Drahtlosnetzwerkverbindung wieder aktiviert werden, damit sich das Smartphone erneut beim „Regisseur“ anmelden kann. Aus baulichen Gründen kommt hier keine Lichtschranke, sondern eine Art Funkbarke zum Einsatz. Ein Sender vom Typ *Stick'n'Find*, welcher den Bluetooth-Low-Energy-Standard verwendet, ist so konfiguriert, dass er im Abstand von 4 Sekunden ein Broadcast-Signal sendet, welches von einem Smartphone mit passender Bluetooth-Schnittstelle empfangen werden kann. Das Smartphone empfängt die Kennung des Senders in Form einer MAC-Adresse und die Stärke des Signals als RSSI-Wert (dBm). Befindet sich das Smartphone in einem Abstand von zwei bis drei Metern zum Sender, so wird eine Signalstärke oberhalb des konfigurierten Schwellwerts von -90 dBm erreicht. Dadurch wird die Drahtlosnetzwerkverbindung reaktiviert, was eine Registrierung im Netzwerk des langen Gangs ermöglicht.

BLE-Tracking

Der zweite von uns getrackte Raum des Soundwalks ist ein 120 Meter langer Gang parallel zu den Geleisen, welcher dem Publikum normalerweise nicht offen steht. Neben zwei WLAN-Zugriffspunkten kann der aufmerksame Beobachter mehrere blau leuchtende LEDs an den Seitenwänden des Gangs entdecken. Jede dieser blauen LEDs gehört zu einem *Bluetooth Low Energy Tag* (BLE-Tag). Diese batteriebetriebenen Tags senden in regelmässigen Abständen ein Advertising-Signal ab, welches von einem handelsüblichen Smartphone mit Bluetooth 4.0 Unterstützung

empfangen werden kann. Das Smartphone empfängt die eindeutigen MAC-Adressen der Tags, welche sich in der Nähe befinden, sofern die elektromagnetischen Signale nicht durch Wände oder andere Abschirmungen zu stark gedämpft sind.

Seit Ende 2009 ist der Industriestandard Bluetooth um die Ergänzung Low Energy (BLE) erweitert worden. BLE verwendet dieselben Frequenzen im 2.4-GHz-Band wie das herkömmliche Bluetooth-Protokoll, ist aber dazu nicht kompatibel. Entsprechende Geräte müssen diesen Standard also explizit unterstützen. Momentan sind vor allem BLE-Chips von Texas Instruments und Nordic Semiconductors erhältlich.

BLE in der Version 4.0 definiert für verschiedene Anwendungsgebiete unterschiedliche Profile. Für unseren Anwendungsfall der Lokalisierung bietet sich vor allem das Proximity-Profil an, mit welchem abgefragt werden kann, welche BLE-Tags sich in welcher Distanz zum Smartphone befinden. Dieses Profil wird beispielsweise in der zuvor erwähnten *Stick'n'Find*-Lösung verwendet.

Die Firma *Stick'n'Find* bietet kleine runde BLE-Tags an, mit deren Hilfe Gegenstände lokalisiert werden können [SnF]. Die Tags werden mit einer Knopfzelle betrieben und enthalten den BLE-Chip 51822 von Nordic Semiconductor. Dieser unterstützt das Proximity-Profil, verwendet aber eine proprietäre Firmware. *Stick'n'Find* bietet ein SDK für iOS an, nicht aber für Android. Die Spezifikationen sind ebenfalls nicht offengelegt, weshalb wir die Standardkonfiguration der BLE-Tags getestet haben. Sind die Tags aktiviert, senden sie alle 4 Sekunden ein Broadcast-Signal, sofern sie nicht mit einem Gerät verbunden sind. Dieses Intervall

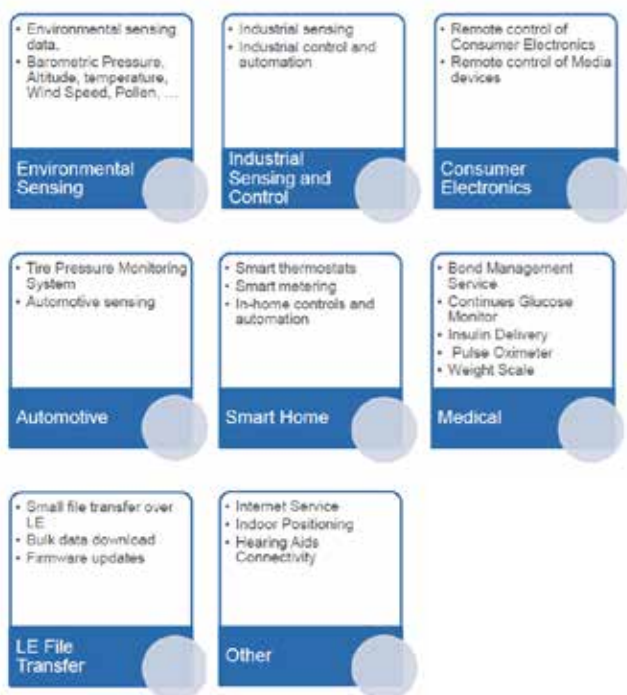


Abbildung 6: Zukünftige Einsatzgebiete von Bluetooth Low Energie gemäss der Bluetooth Special Interest Group [BSIG].

kann mit Hilfe des iOS-SDK umprogrammiert werden, wobei eine dauerhafte Umstellung des Intervalls bei unseren Tests nicht funktionierte.

Als Alternative zu *Stick'n'Find* haben wir BLE-Module der Firma Wavetek getestet, welche den gleichen BLE-Chip beinhalten [Wav]. Die Tags sind für uns so vorprogrammiert worden, dass sie bei minimaler Sendeleistung arbeiten und dafür zehn Mal pro Sekunde ein Advertising-Signal senden. Der Preis dafür ist eine relativ kurze Batterielaufzeit von weniger als einer Woche. Ein Advertising-Signal überträgt hauptsächlich die MAC-Adresse des BLE-Geräts. Das empfangende Smartphone ermittelt zusätzlich noch die Signalstärke in Form eines RSSI-Werts (dBm). Diese beiden Parameter können dann für eine Lokalisierung verwendet werden.

Die Unterstützung Seitens der Smartphones ist jedoch eher mangelhaft. Aktuelle Android-Geräte besitzen in der Regel zwar ein Bluetooth-Modul, welches BLE unterstützt, BLE selbst wird jedoch in Android erst ab Version 4.3 offiziell unterstützt. Die meisten Geräte laufen jedoch noch mit Android 4.1 oder 4.2. Manche Hersteller, beispielsweise Samsung, bieten deshalb eigene SDKs an, um BLE-Geräte ansteuern zu können. Diese SDKs sind leider in der Regel hersteller- oder gar modellspezifisch.

Lokalisierung

In der Bluetooth Special Interest Group sind bereits Profile für Bluetooth 4.1 definiert worden (Abb. 6). Eines dieser Profile soll in Zukunft auch Indoor-Lokalisierung ermöglichen. Verschiedene Hersteller, unter anderem auch Microsoft (Ex-Nokia) sind bereits an der Entwicklung entsprechender Lösungsansätze [CPK13, WYZ13, ZLS13].

Für eine Lokalisierung basierend auf BLE-Tags bieten sich vor allem zwei Techniken an: Fingerprinting und Triangulierung.

Beim Fingerprinting-Ansatz müssen die Positionen der Sender (BLE-Tags oder WLAN-Zugriffspunkte) nicht bekannt sein. Stattdessen werden in einem Vorbereitungsschritt an verschiedenen bekannten Testpunkten des Raumes Signalmessungen durchgeführt und alle vorhandenen Signale pro Position zu einem positionsspezifischen also charakteristischen „Fingerabdruck“ vereint und zusammen mit den Positionskordinaten in einer zentralen Datenbank abgespeichert. Sobald diese Datenbank aufgebaut ist, kann sie für die eigentliche Lokalisierung herbeigezogen werden. Zur Bestimmung der Position misst das Smartphone im Raum alle Signale und erstellt daraus wieder einen „Fingerabdruck“. Diesen sendet es an einen Server, welcher Zugriff auf die zentrale Datenbank hat, und der Server durchsucht die Datenbank nach möglichst ähnlichen „Fingerabdrücken“. Aus den gespeicherten Raumpositionen der

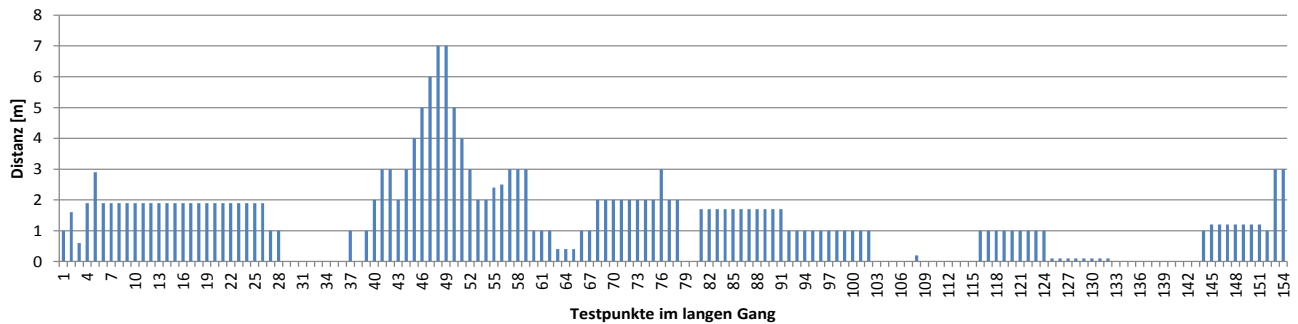


Abbildung 7: Lokalisierungsfehler beim Triangulierungsverfahren mit 18 BLE-Tags gemessen bei mehr als 150 Testpunkten in einem 120 Meter langen Gang.

ähnlichsten „Fingerabdrücke“ lässt sich dann die Raumposition des Smartphones abschätzen.

Dieser Ansatz funktioniert für statische Lokalisierung (die Bestimmung von Personen oder Objekten, welche sich kaum bewegen) recht gut. In einer ruhigen Umgebung stabilisieren sich die empfangenen Signale nach wenigen Sekunden und es kann eine Lokalisierungsgenauigkeit von 2.5 bis 5 Metern erzielt werden. Der Vorteil dieser Methode besteht darin, dass alle möglichen Signalquellen für die Bestimmung der „Fingerabdrücke“ verwendet werden können, egal ob die Senderpositionen bekannt sind oder nicht. Ein gewichtiger Nachteil ist jedoch der vorgängige Aufbau der Datenbank und die spätere Aktualisierung dieser, auch dann, wenn die Aktualisierung automatisch von den Benutzern der Datenbank erledigt werden kann.

Beim zweiten Ansatz, der Triangulierung, müssen die Positionen der Signalquellen (BLE-Tags) und mit Vorteil auch die Signalstärken der „sichtbaren“ Tags bekannt sein. Mit Sichtbarkeit meinen wir hier, dass das Smartphone ein Tag genau dann „sieht“, wenn es von ihm ein Advertising-Signal empfängt. Da das Advertising-Signal die eindeutige MAC-Adresse des Tags beinhaltet, kann mithilfe eines Mappings die Position des Tags eruiert werden. Für den Fall, dass mindestens drei Tags sichtbar sind, können die bekannten Positionen der Tags verwendet werden, um die Position des Smartphones abzuschätzen. Zur eigentlichen Triangulierung kommt es dann, wenn aus den empfangenen Signalstärken von drei verschiedenen Sendern auf die Distanzen zu den Sendern geschlossen wird und diese Distanzen zusammen mit den Positionen der Sender verwendet werden, um die Position des Empfängers genau zu berechnen.

Dieser Ansatz funktioniert auch für eine dynamische Lokalisierung recht gut, vorausgesetzt, dass die Advertising-Signale beim Empfänger in kurzen Abständen (ca. 0.1 s) eintreffen. Erste Tests zeigen, dass sich damit durchschnittliche Lokalisierungsgenauigkeiten von 1.5 bis 3 Metern erzielen lassen (Abb. 7). Der Nachteil dieses Ansatzes ist, dass die Positionen der Signalquellen

bekannt sein müssen (oder die Signalquellen müssen nicht nur ihre eindeutige MAC-Adresse, sondern auch gleich noch ihre Position übermitteln) und dass ein relatives enges Netz an Tags vorhanden sein sollte, wenn eine hohe Lokalisierungsgenauigkeit angestrebt wird. Da sich BLE-Tags aber sehr einfach anbringen lassen und der Stromverbrauch nicht sehr gross ist, kann der Aufbau einer entsprechenden Tag-Infrastruktur durchaus für grosse Räume in Erwägung gezogen werden. Zukünftige Forschungsprojekte könnten hier aufzeigen, mit welchem finanziellen Aufwand welche Genauigkeit erzielt werden kann.

Android App

Die Steuerung unseres *Head Mounted Devices* (HMD) wird von einem Smartphone übernommen. Die Anforderungen an Hard- und Software sind dabei im Vergleich zu gewöhnlichen Smartphone-Apps aussergewöhnlich hoch: Es müssen in Echtzeit verschiedenste Sensoren ausgewertet, drahtlose Verbindungen wie WLAN und Bluetooth genutzt und auch Audio-Processing durchgeführt werden. Dazu kommt, dass die App nicht durch Stromsparmechanismen des Betriebssystems beeinträchtigt werden darf.

Bei der Wahl der Smartphone Plattform haben wir uns für Android entschieden, da Android den App-Entwicklern die grösstmögliche Flexibilität im Umgang mit dem Betriebssystem bietet. Geschlossenerer Plattformen wie Apple iOS unterliegen starken Einschränkungen durch den Hersteller, welche nicht oder nur mit grossem Aufwand umgangen werden können.

Eine erste Herausforderung ist die Entwicklung einer App, deren Funktionen laufend im Hintergrund operieren und auch dann noch funktionieren, wenn beispielsweise das User Interface der App durch Druck auf den Home-Button in den Hintergrund versetzt oder das Display ausgeschaltet wird. Im Normalfall würde dadurch die App einfach durch das Android-System pausiert.

Apps, welche sich im Hintergrund befinden, können durch die Speicherverwaltung von Android jederzeit aus dem Speicher entfernt werden. Wir lösen dieses Problem durch Verwendung ei-

Kopfnicken	Schwellwert
Ableitung der linearen Beschleunigung	0.3 m/s ³
Gyroskop	1.1 rad/s
Maximale Zeit zwischen linearer Beschleunigung und Gyroskop	45 ms
Maximale Zeit zwischen Teilgesten	500 ms
Minimale Anzahl Teilgesten	2 - 3

Tabelle 2: Empirisch ermittelte Schwellwerte zur Detektion von Kopfnicken mithilfe eines Gyroskops und Beschleunigungssensors.

nes Hintergrund-Services, welcher zusammen mit der App gestartet wird und praktisch die gesamte Funktionalität der App enthält. Android erlaubt es, Services zu definieren, welche weiter im Hintergrund operieren können, auch wenn andere Apps im Vordergrund sind. Auch die Umgehung von Stromsparmechanismen ist möglich. Android bietet dazu sogenannte *Wake Locks*. Durch Akquirierung eines solchen Wake Locks kann ein Service weiter ausgeführt werden, auch wenn beispielsweise das Display abgeschaltet wird. Ohne ein Wake Lock greifen in diesem Fall die Stromsparmechanismen des Systems, und die CPU wird in den Standby-Modus versetzt.

Ein letztes Problem des Android-Systems im Zusammenhang mit der Speicherverwaltung kann nur mittels eines Workaround umgangen werden: Das explizite Beenden einer App ist nicht vorgesehen. Wenn das User Interface einer App, egal auf welche Weise, geschlossen wird, wird die

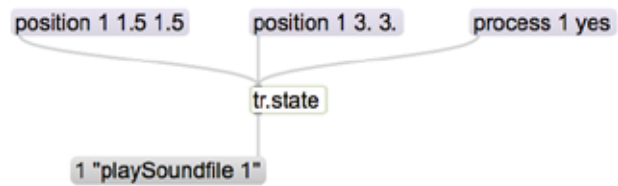


Abbildung 9: Das Backend Objekt „tr.state“ für Max/MSP mit eingehenden und ausgehenden Meldungen für Client 1.

App nicht wirklich beendet. Android entscheidet selbst, wann der Speicher freigeräumt wird. Entwickler können darauf keinen Einfluss nehmen. Weil unser Hintergrund-Service auch dann weiterlaufen muss, wenn das User Interface geschlossen wird, ist eine Beendigung unserer App nicht einfach. Ohne eine vorgängige Terminierung wäre jedoch ein Neustart unmöglich. Wir umgehen dieses Problem, indem wir im User Interface eine spezielle „Exit“-Funktion anbieten, die vor dem Schliessen den Service beendet. Android kann anschliessend den Speicher bei Bedarf freigeben. Beim nächsten Aufrufen der App wird der Service einfach neu gestartet, falls er sich noch im Speicher befindet oder neu instanziiert.

Für das Audio-Processing verwenden wir *Pure Data* [PD]. PD ist eine spezielle Programmiersprache zur Verarbeitung und Erzeugung von Audio-Daten, kann aber auch zur Auswertung von Sensoren und Steuerung von Abläufen genutzt

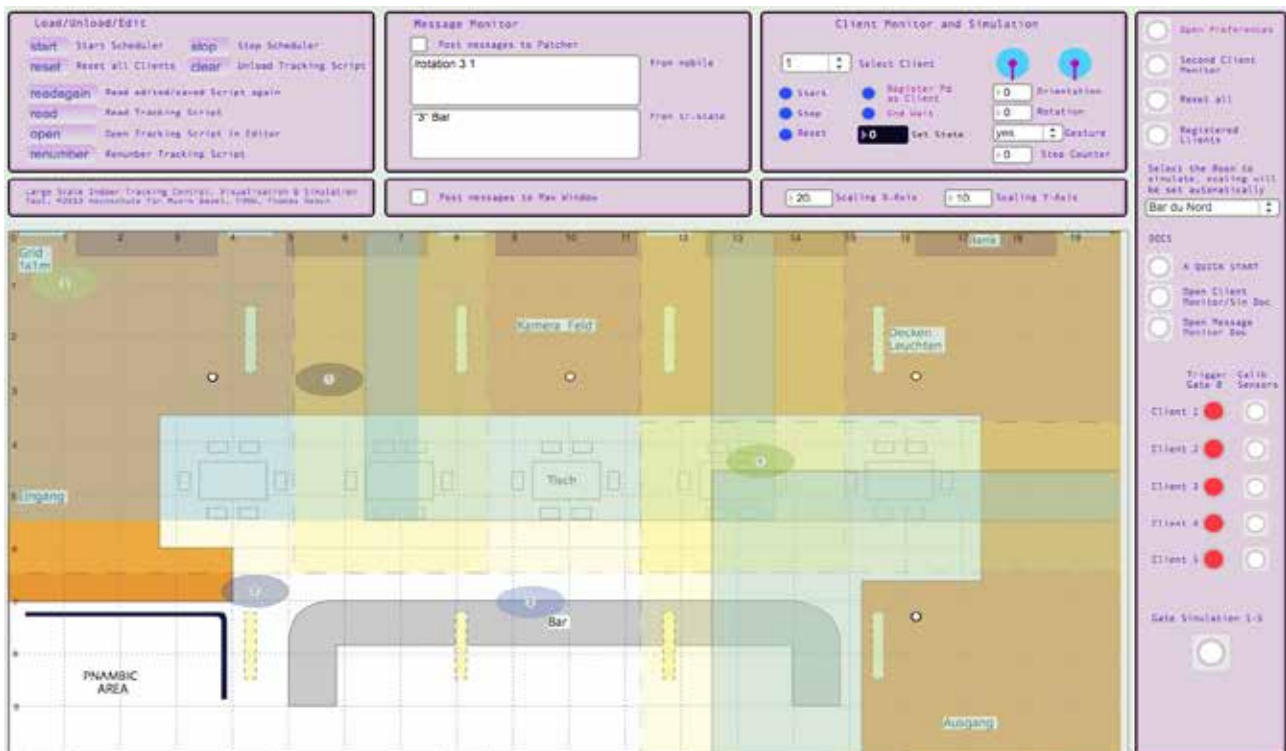


Abbildung 8: Grafische Benutzeroberfläche des „Regisseurs“ in Max/MSP programmiert. Sie ermöglicht die Visualisierung von Position, Orientierung und Gesten als auch die Simulation dieser Ereignisse.

```

1: newClient 1;
2: newClient 2;
3: nextState 1 STATE 0 position 1. 1. 1. 1. message sender "playSoundfile 1";
4: nextState 2 STATE 1 !position 1. 1. 1. 1. message sender "playSoundfile 2";
5: nextState 3 STATE 2 yes message sender "playSoundfile 3";
6: nextState 1000 maxTime 60. message sender "The End";

```

Listing 1: Einfaches Skript zur Programmierung einer Zustandsmaschine

werden. PD kann mit Hilfe der Open-Source-Bibliothek *libpd* genutzt werden. Es handelt sich um eine C-Bibliothek, welche auch unter Android lauffähig ist. In unserer App wird *libpd* für die Ausgabe von vorgefertigten Audiodateien sowie für die Steuerung von Teilen des Audiowalks eingesetzt. Dabei werden auf dem Smartphone entsprechende PD-Skripte, sogenannte Patches, ausgeführt. Diese verarbeiten Ereignisse von Sensoren des Smartphones oder Anweisungen, die vom „Regisseur“ über das Netzwerk erhalten werden.

Die Android App generiert auch selber Meldungen, die an den „Regisseur“ gesendet und von ihm ausgewertet werden müssen. Dazu gehören u.a. die Gestenerkennung für Kopfnicken und Kopfschütteln und die Schritterkennung, die immer dann einspringt, wenn sich das Smartphone im ungetrackten Bereich befindet.

Das Detektieren von Kopfnicken und Kopfschütteln erfolgt mit Hilfe des eingebauten Gyroskops und Beschleunigungssensors. Zunächst wird die Ableitung der linearen Beschleunigung gebildet. Sofern auf der für Kopfnicken/Kopfschütteln relevanten Achse ein bestimmter Schwellwert sowohl beim Gyroskop als auch bei der Ableitung der linearen Beschleunigung überschritten wird und beide Überschreitungen innerhalb eines bestimmten Zeitbereiches stattfinden, wird dies als Teilbewegung der Geste gezählt. Folgen mehrere solcher Teilbewegungen innerhalb einer bestimmten Zeit aufeinander, wird dies als vollständiges Kopfnicken/Kopfschütteln interpretiert. Die Schwellwerte in Tabelle 2 sind im Laufe des Projekts empirisch ermittelte Werte.

Die Schritterkennung basiert auf Messungen der Beschleunigung entlang der z-Achse. Wird ein bestimmter Schwellwert überschritten, wird dies als Schritt gewertet und die Detektion für die nächsten 250 ms blockiert. Dabei muss sauber darauf geachtet werden, dass andere Bewegungen wie beispielsweise Kopfnicken nicht als Schritt gezählt werden.

Max/MSP

Unsere Reise durch den Bahnhof wird am Ende des langen Gangs fortgesetzt. Dabei passieren wir noch weitere Tore und weitere Räume und kommen schliesslich wieder in der Bar du Nord an.

Auf einer grossen Leinwand sehen wir die aktuellen Positionen aller Teilnehmerinnen und deren zurückgelegte Wege durch den Bahnhof. Der „Regisseur“ – eine in Max/MSP programmierte Software – weiss also zu jedem Zeitpunkt, wo wir uns befinden. Er kann uns mit positions- und orientierungsabhängigen Informationen versorgen und unseren weiteren Weg beeinflussen. Die in Abbildung 8 dargestellte grafische Benutzungsoberfläche des „Regisseurs“ erlaubt zudem sowohl die Visualisierung von Position, Orientierung und Gesten als auch die Simulation dieser Ereignisse.

Um den Künstlern möglichst schnell eine Software zur Verfügung stellen zu können, die es ihnen erlaubt, den Ablauf des Soundwalks selbstständig zu planen, ohne auf die ständige Hilfe der Programmierer angewiesen zu sein, wird als Basis zur Entwicklung des „Regisseurs“ die visuelle Programmierumgebung Max/MSP benutzt, welche aus dem bereits erwähnten *Pure Data* entstanden ist. Max/MSP erlaubt sehr schnelles Prototyping, insbesondere die GUI-Entwicklung ist mittels Drag-and-Drop sehr effizient. Kontrollstrukturen werden allerdings ab einer gewissen Komplexität sehr schlecht lesbar und schwierig zu warten, daher ist das Backend als Objekt für Max/MSP in C programmiert. Das Backend besteht im Wesentlichen aus einer programmierbaren Zustandsmaschine, die in Echtzeit auf die eingehenden Events reagiert, den Zustand der mobilen Clients wechselt und Meldungen an die Smartphones verschickt (Abb. 9). Ein exemplarisches Skript zur Programmierung der Zustandsmaschine ist in Listing 1 abgebildet und nachfolgend beschrieben.

Im Skript werden zunächst die mobilen Clients mit den Namen „1“ und „2“ generiert. Beide starten automatisch in Zustand 0. In Zeile 3 wird ein Zustandsübergang von 0 nach 1 beschrieben, welcher durch Einnehmen der angegebenen Raumposition ausgelöst wird. Mit dem Zustandsübergang wird auch die Aktion „playSoundfile 1“ an das entsprechende Smartphone des Clients gesendet. Das Schlüsselwort sender drückt dabei aus, dass die Nachricht an den gleichen Client gesendet wird, welcher auch den Zustandsübergang vollzieht. In Zeile 4 wird ein Zustandsübergang von 1 nach 2 beschrieben, welcher durch Verlassen der ange-

geben Raumposition ausgelöst wird. Der in Zeile 5 beschriebene Zustandsübergang von 2 nach 3 wird durch den Erhalt der Nachricht *yes* ausgelöst. Schliesslich in Zeile 6 wird unabhängig vom aktuellen Zustand in Zustand 1000 gewechselt, falls sich der mobile Client länger als 60 Sekunden im gleichen Zustand befindet.

Zusätzlich zu *position*, *yes* und *no*, *maxTime* und *STATE* existieren zahlreiche weitere Bedingungen für Zustandsänderungen, u.a. *orientation* und *rotation* oder auch der Name eines Clients, was individuelle Abläufe für unterschiedliche Clients erlaubt. Bis zu 20 dieser Bedingungen können logisch miteinander AND-verknüpft werden. Ebenso können bis zu 20 unterschiedliche Meldungen pro Zustandsübergang verschickt werden.

Ergebnisse und Ausblick

Das Kunstprojekt „LautLots“ fand sowohl bei den Teilnehmerinnen und Teilnehmern als auch den Medien (Radio SRF2, Basellandschaftliche Zeitung, Programm-Zeitung) erfreulich grossen Anklang. Die Technik dahinter interessierte die hauptsächlich kunstaffinen Teilnehmerinnen und Kulturberichtstatter nur bedingt, obwohl ein Grossteil der gesamten Projektzeit in die Technik und die technische Umsetzung der künstlerischen Inszenierung floss.

Das gesamte Projekt war von Anfang an eine enge Verwebung von Kunst und Technik und es war frühzeitig klar, dass der pädagogische Anspruch der Hochschule für Technik, die verwendete Technik offen darzulegen und so das Interesse daran zu wecken, dem künstlerischen Anspruch der Hochschule für Musik widersprach, die Magie der akustisch generierten virtuellen Welt auf-

rechtzuerhalten. Hier prallten gänzlich verschiedene Kulturen aufeinander, welche sich auch bei der generellen Herangehensweise und Umsetzung immer wieder offenkundig zeigten.

Aus technischer Sicht sind etliche Probleme mehr oder weniger elegant gelöst worden, andere konnten nur teilweise gelöst oder angepackt werden. Exemplarisch zu erwähnen ist die Auseinandersetzung mit Bluetooth Low Energie, welche nicht von Anfang an geplant war, sondern erst im späteren Projektverlauf als Alternative zu WLAN-basierter Lokalisierung Aufmerksamkeit erhielt. Da erste Tests mit BLE vielversprechend verliefen, besteht das Bedürfnis in diesem Bereich weitere Untersuchungen machen zu können und in einer Machbarkeitsstudie das Potential für Indoor-Lokalisierung auf BLE-Basis aufzuzeigen.

Referenzen

- [Blog] LautLots. Ein akustischer Guide durch den Badischen Bahnhof. <http://www.lautlots.ch>
- [BSIG] Bluetooth Special Interest Group. <http://www.bluetooth.org>
- [CPK13] Liang Chen, Ling Pei, Heidi Kuusniemi, Yuwei Chen, Tuomo Kröger, Ruizhi Chen: Bayesian Fusion for Indoor Positioning Using Bluetooth Fingerprints. *Wireless Personal Communications (WPC)* 70(4):1735-1745 (2013)
- [PD] Pure Data. <http://puredata.info/>
- [SnF] Stick-N-Find. <https://www.sticknfind.com/>
- [Wav] WaveTek. <http://www.wavetek.com.hk/>
- [WYZ13] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, Laurie G. Cuthbert: Bluetooth positioning using RSSI and triangulation methods. *CCNC 2013*:837-842
- [ZLS13] Li Zhang, Xiao Liu, Jie Song, Cathal Gurrin, Zhiliang Zhu: A Comprehensive Study of Bluetooth Fingerprinting-Based Algorithms for Localization. *AINA Workshops 2013*:300-305

Parsing Graphs: Applying Parser Combinators to Graph Traversals¹

Connected data such as social networks or business process interactions are frequently modeled as graphs, and increasingly often, stored in graph databases. In contrast to relational databases where SQL is the proven query language, there is no established counterpart for graph databases. One way to explore and extract data from a graph database is to specify the structure of paths (partial traversals) through the graph. We show how such traversals can be expressed by combining graph navigation primitives with familiar grammar constructions such as sequencing, choice and repetition – essentially applying the idea of parser combinators to graph traversals. The result is trails, a Scala combinator library that provides an implementation for the neo4j graph database and for the generic graph API blueprints.

Daniel Kröni, Raphael Schweizer | daniel.kroeni@fhnw.ch

Enter the tangled world of Carol where everything centers on friendship and pets. The graph in Figure 1 shows four people, their relationships and their pets.

Carol has a dog and now she is thinking about a cute name. Of course the name should be unique, at least within her circle of friends. She asks herself: “What names did my friends and their friends etc. give their pets?” Here is how she would state that same question after having read this paper:

```
val friends = V("Carol") ~ (out-
  ("loves") | out("likes")).+
val petNames =
  friends ~> out("pet") ^^
  get[String]("name")
```

This code defines a traversal through the given graph by specifying the “grammar” of the paths to be followed. It consists of graph navigation steps (V , out) and combinators (\sim , $\sim>$, $|$, $+$). Applying *petNames* yields four results:

```
(Carol -likes-> Dave -pet-> Fluffy, "Fluffy")
(Carol -loves-> Bob -pet-> Murphy, "Murphy")
(Carol -loves-> Bob -likes-> Carol -likes->
  Dave -pet-> Fluffy, "Fluffy")
(Carol -loves-> Bob -loves-> Alice -loves->
  Bob -pet-> Murphy, "Murphy")
```

Each result contains the full path of the traversal plus a designated value, in our case, the name property of the pet. In this paper we describe how this works:

- We develop the datatype of a graph traverser.
- We describe a set of functions that allows us to combine graph navigation primitives into expressive traversal descriptions.

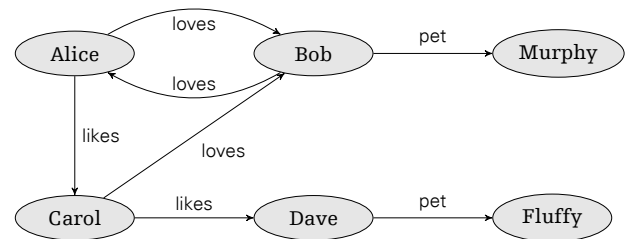


Figure 1 : Carol's World, a small example graph

- We extend the library by cycle detection, labeling functionality and subqueries to allow even more elegant traversals.

Graph Traversal Combinators

Step-by-step we develop a graph traversal library. First we shape the type of a traverser. In a first approximation, a traverser Tr_5 is a function that takes a graph as the input and returns a path as the result. *Path* is a list of graph elements that alternate between nodes and edges.

```
type Tr5 = Graph => Path
```

There may be more than one path that fits the specification of a traverser – or none at all. We account for this by letting the result be a *Stream* of paths, as proposed by Wadler [3]. *Stream* allows us to lazily yield result paths on demand rather than to eagerly compute all results.

```
type Tr4 = Graph => Stream[Path]
```

A traverser may start with an empty path, but usually it describes an extension of the preceding path. To model this scenario we extend the type of traverser and add the preceding path as a further input parameter:

```
type Tr3 = Graph => Path => Stream[Path]
```

In the end, a traverser might return an arbitrary value besides the path, for example, the value of a property:

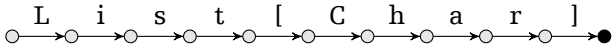
```
type Tr2[+A] = Graph => Path
=> Stream[(Path, A)]
```

¹ Die Originalversion dieses Artikels erschien in den Proceedings of the 4th Workshop on Scala, ECOOP '13, Montpelier, Juli 2013

There is no reason to restrict Tr_2 to *Graph* and *Path*. We will see that a traverser may read from any environment E , it may transform some state S and it may yield potentially multiple results:

```
type Tr1[-E, S, +A] = E => S
=> Stream[(S, A)]
```

In this sense, it is a very general structure. In fact, Hutton's and Meijer's monadic parsers [1] are using this type for parsing strings. This is because a *List[Char]* is a graph:



When parsing text, the state is the string being consumed. In contrast, when traversing a graph, we build up the path, which is the sequence of visited nodes and edges. Depending on the head of a path there are different steps we can next take. On an edge, for example, it makes no sense to ask for outgoing edges which are only available on nodes.

To accommodate this fact, we differentiate between the input state type I and the output state type O . This allows us to statically express whether a traverser expects a node or an edge and thereby rejecting meaningless patterns during compilation.

```
type Tr[-E, -I, +O, +A] = E
=> I => Stream[(O, A)]
```

The above is the type of traversers we will use in the following discussion. It is worth noting that it combines three well known monads:

- Nondeterminism: The multiple results of a traverser, represented as a *Stream*.
- Indexed-State: The state which is threaded through – potentially changing its type from I to O .
- Reader: The read-only environment E which is passed to each traverser.

Given the type Tr , we can explore primitive graph navigation traversers as well as grammar-like combinators.

Traverser Primitives

The two most basic traversers are *success* and *fail*. *success* creates a traverser that always succeeds with the given value a , leaving the state untouched. *fail* is a traverser that contains no results – a dead end, which allows no further traversal:

```
def success[E, S, A](a: A): Tr[E, S, S, A] =
  _ => s => Stream((s, a))
def fail[E, S, A]: Tr[E, S, S, A] =
  _ => _ => Stream()
```

In a similar fashion, we define the three traversers *getEnv* to read the environment, *getState* to read the state, and *setState* to write the state. Below are their signatures:

```
def getEnv[E, S]: Tr[E, S, S, E]
def getState[E, S]: Tr[E, S, S, S]
def setState[E, I, O](o: O):
  Tr[E, I, O, Unit]
```

Until now, the presented traversers have not been specific to graph traversal but were primitive building blocks for more specific traversers. We will now focus on the graph-specific navigation traversers for directed graphs which consist of nodes and edges, each with associated key-value pairs. The environment and its corresponding graph-element types are fixed to an implementation-dependent graph API. The accompanying state carries the type of the head of its path² as a phantom type:

```
import org.neo4j.{graphdb => neo4j}
type GraphAPI =
  neo4j.GraphDatabaseService
type Elem = neo4j.PropertyContainer
type Node <: Elem = neo4j.Node
type Edge <: Elem = neo4j.Relationship
case class State[+Head <: Elem]
  (path: List[Elem])
```

To navigate the graph we propose a few primitives whose names are borrowed from *Gremlin* [9]. Navigation primitives extend their input path by appending the elements they yield. The following traversers need to be implemented for each specific graph database:

Function	Description
$V, V(id)$	all nodes, node identified by id
$E, E(id)$	all edges, edge identified by id
$outE, outE(t)$	all out-edges, out-edges with tag t ⁽³⁾
$inE, inE(t)$	all in-edges, in-edges with tag t ⁽³⁾
$outV, inV$	start node, end node of an edge

As an example, we will look at *outE*'s function signature:

```
def outE(tagName: String):
  Tr[GraphAPI, State[Node], State[Edge],
  Edge]
```

which has the following expanded return type:

```
GraphAPI => State[Node]
=> Stream[(State[Edge], Edge)]
```

This function takes a graph and a path that ends in a *Node* and from there it steps onto all outgoing *edges* with the given *tagName*. This leads to paths which end in an *Edge*. Together, this edge is then returned with the extended path.

In order to access properties on nodes and edges, the function *get* must be implemented as well:

```
def get[A](key: String)(e: Elem): A
```

These primitives will be combined into powerful traversal definitions.

² In principle, it is sufficient to track only the current position in the graph, however, we are often interested in the trace.
³ neo4j and blueprints support tagged edges, in contrast to untagged nodes.

Traverser Combinators

We now want to combine these primitive traversers into complex path expressions, which results again in traversers. This property is a key to their compositional nature.

The following table shows the name of those combinators as well as the sugar we provide to concisely express traversals:

Function	Sugar	Description
seq	a ~ b	First a then b
choice	a b	Follow both branches
opt	a.?	Repeat 0..1
many	a.*	Repeat 0..n
many1	a.+	Repeat 1..n

flatMap is used to sequentially combine any two traversers. It passes through the same environment to both traversers, threads the state through the first traverser into the second one and returns the final states together with the results:

```
def flatMap[E,I,M,O,A,B](tr: =>
  Tr[E,I,M,A])(f: A => Tr[E,M,O,B]):
  Tr[E,I,O,B] =
  e => i => tr(e)(i).flatMap {
    case (m,a) => f(a)(e)(m)
  }
```

Note that the inner *flatMap* is called on *Stream*, and how the different input and output state types *I*, *M* and *O* line up – from $[_I,M,_]$ and $[_M,O,_]$ to $[_I,O,_]$. To allow recursive definitions, all combinators take their traverser arguments by-name.

Tr together with *flatMap* and *success* becomes a structure that is slightly more general than monadic, due to the state types [4]. Luckily, Scala's for-comprehension does not worry about this.

Now *map* and *filter*, using *flatMap*, *success* and *fail*, can be implemented as follows:

```
def map[E,I,O,A,B](tr: => Tr[E,I,O,A])
  (f: A => B): Tr[E,I,O,B] =
  flatMap(tr)(a => success(f(a)))

def filter[E,I,O,A](tr: => Tr[E,I,O,A])
  (f: A => Boolean): Tr[E,I,O,A] =
  flatMap(tr)(a => if(f(a))
    success(a) else fail)
```

There is another, less powerful but often sufficient way to sequentially combine two traversers. *seq* does not use the result of the first traverser to obtain the subsequent traverser as in *flatMap* but simply returns both values in a fancy-looking tuple named ~:

```
case class ~[+A,+B](a: A, b: B)

def seq[E,I,M,O,A,B](fst: => Tr[E,I,M,A],
  snd: => Tr[E,M,O,B]): Tr[E,I,O,A~B] =
  for(a <- fst; b <- snd)
  yield new ~(a,b)
```

The related functions *~*, *~>* and *<~* return the whole tuple, the right-hand-side and the left-hand

side. These functions as well as the infix sugar for *map* ^^ are courtesy of Scala's parser combinators [2, 727-755]. They allow the writing of good-looking sequential compositions of traversers such as *out* which first navigates from a node to an outgoing edge and from there to the target node:

```
def out(tagName: String)
  : Tr[GraphAPI, State[Node], State[Node],
  Node] = outE(tagName) ~> inV()
```

In addition to the above sequencing function, a means is needed to express branching: *choice*. Since we are interested in all matching result paths this combinator follows *both* arguments using the same state and concatenates (#:::) their results. This is different to typical combinator parsers which for reason of speed often try the second alternative only if the first one fails:

```
def choice[E,I,O,A](
  either: => Tr[E,I,O,A],
  or: => Tr[E,I,O,A]): Tr[E,I,O,A] =
  e => i => either(e)(i) #::: or(e)(i)
```

Now we have all the ingredients to implement *opt*, *many* and *many1*. Note that they restrict their argument traverser to start and end on the same state type *S*. The implementations are straight forward:

```
def opt[E,S,A](tr: => Tr[E,S,S,A])
  : Tr[E,S,S,Option[A]] =
  choice(success(None),
  map(tr)(Some[A](_)))

def many[E,S,A](tr: => Tr[E,S,S,A])
  : Tr[E,S,S,Stream[A]] =
  choice(success(Stream()), many1(tr))

def many1[E,S,A](tr: => Tr[E,S,S,A])
  : Tr[E,S,S,Stream[A]] =
  for(a <- tr; as <- many(tr))
  yield a #:: as
```

This concludes the basic functionality of our graph traversal combinators. Improvements to this minimalistic design are discussed in the further sections. We make our traversers cycle-aware and add the ability to label values, which can then be referred to in queries. Finally we show how to implement subqueries.

Cycle Detection

Consider the following traverser:

```
V("Alice") ~> (out("loves")
  | out("likes")).+
```

Since there is no inherent ordering of the edges, a possible sequence of result paths could look like this:

```
Alice -loves-> Bob
Alice -loves-> Bob -loves-> Alice
```

Alice -loves-> Bob -loves-> Alice -loves-> Bob

...

The given implementation would never stop generating longer and longer expansions of the cycle and never yield the following path: *Alice -likes-> Carol*.

Note that the queried graph does not need to contain cycles: e.g. *(out("pet") ~ in("pet")).+* is problematic by itself.

In general the application of *many* and *many1* may cause problems. Clearly this behavior is undesirable. Cycles should be detected and handled appropriately. Our implementation adheres to the following definition: If, within an application⁴ of *many* or *many1*, the repeated traverser yields the same snippet a second time, then it is a cycle. Consistent with our definition this path, *Carol -loves-> Bob -loves-> Alice -likes-> Carol -loves-> Bob -pet-> Murphy*, is discarded from the result mentioned in the introduction due to the repeated *-loves-> Bob* snippet.

Detecting cycles requires the snippets to be tracked, therefore we extend the state:

```
case class State[+Head <: Elem]
  (path: List[Elem],
   cycles: Set[List[Elem]])
```

For the sake of simplicity our implementation follows cycles only once, which might be returned as part of the result as well.

Labels

As a further extension, we allow the values that are emitted by a traverser to be labeled. This requires additional state of type *Map[String, List[Any]]* which maps a label to a list of values. Why use a list of values and not just a single value? The answer is that labeling inside a repetition might produce more than one value, or perhaps none at all.

For example an application of labels is looking for unhappy lovers – people who love another person but that person does not return this love:

```
val unhappyLovers = for {
  beloved <- V.as("lvr") ~
  out("loves") ~> out("loves")
  lover <- label("lvr") if
  !lover.contains(beloved)
} yield lover
```

Executing this query on the introductory graph yields the single node *Carol*.

Subqueries

The last extension we implement are subqueries. Essentially subqueries are traversers whose values are preserved while their state changes are discarded. Thus subqueries allow to “match” pat-

terns without having the matched paths polluting the result. Here is the definition of *sub* which runs its argument *tr* as a subquery and in turn yields the stream of *tr*'s results:

```
def sub[E,I,O,A](tr: Tr[E,I,O,A])
  : Tr[E,I,I,Stream[A]] =
  e => i => Stream((i, tr(e)
    (i).map(_._2)))
```

Using *sub* we can search for beloved pet owners:

```
val belovedPetOwners = for {
  petOwner <- V
  pets <- sub(out("pet"))
  if pets.nonEmpty
  lover <- in("loves")
} yield (petOwner, lover)
```

This yields *(Carol -loves-> Bob, (Bob, Carol))* and *(Alice -loves-> Bob, (Bob, Alice))*. Note that the pets do not show up in the result.

Conclusion and Related Work

We have developed a simple combinator library to concisely express graph traversals. It is currently being evaluated and extended in the context of a business intelligence project [10]. Ongoing work can be observed on the *trails* webpage [6].

To access information stored in graph databases we have found low-level APIs, imperative, embedded graph traversal languages such as the Gremlin family [9] and declarative approaches e.g. Cypher [11] or SPARQL [12]. While others stress expressiveness or good computational complexity [5] *trails* focuses on simplicity – in terms of an educative value, implementation and application.

References

- [1] G. Hutton & E. Meijer. Monadic parser combinators, 1996.
- [2] M. Odersky, L. Spoon, and B. Venners. Programming in Scala: A Comprehensive Step-by-Step Guide. Artima Inc., 2nd edition, 2010.
- [3] P. Wadler. How to replace failure by a list of successes. In Conference on Functional Programming Languages and Computer Architecture, pages 113--128, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
- [4] P. Wadler. Monads and composable continuations. LISP and Symbolic Computation, 7(1):39--55, jan 1994.
- [5] P. T. Wood. Query languages for graph databases. SIGMOD Record, 41(1):50--60, 2012.
- [6] <http://www.github.com/danielkroeni/trails>, 2013.
- [7] <http://www.neo4j.org>, 2013.
- [8] <http://blueprints.tinkerpop.com/>, 2013.
- [9] <http://gremlin.tinkerpop.com/>, 2013.
- [10] <http://www.fhnw.ch/technik/imvs/forschung/projekte/babefisch/babelfish>, 2013.
- [11] <http://docs.neo4j.org/chunked/stable/cypher-query-lang.html>, 2013.
- [12] <http://www.w3.org/TR/rdf-sparql-query>, 2013.

⁴ Top-level applications only, not mutual recursive calls.

A space- and time-efficient Implementation of the Merkle Tree Traversal Algorithm

We have developed an algorithm for the Merkle tree traversal problem which combines the efficient space-time trade-off from the fractal Merkle-trees [4] and the space efficiency from the improved log space-time Merkle-trees traversal [8]. We further programmed a low storage space and a low time overhead version of the algorithm in Java and measured its performance with respect to two different implementations.

Markus Knecht, Carlo U. Nicola | carlo.nicola@fhnw.ch

Merkle's binary hash-trees are one of the most interesting cryptographical building blocks currently available, because their security is independent from any number theoretic conjectures [6]. The security is based solely on two well defined mathematical properties of hash functions: (i) Pre-image resistance: that is, given a hash value v , it is difficult to find a message m such that $v = \text{hash}(m)$; and (ii) Collision resistance: that is, given two messages $m_1 \neq m_2$, the probability that $\text{hash}(m_1) = \text{hash}(m_2)$ can be made as small as one wishes by a suitable choice of n , the number of bits of v . It is interesting to note that the best quantum algorithm to date for searching n -bit-random records in a data base (an analogous problem to hash collision) achieves only a speed up of $O(\sqrt{n})$ to the classical one $O(n)$, where n is the number of records [1].

A Merkle tree is a complete binary tree with an n -bit value associated with each node. Each internal node value is the result of a hash of the node values of its children (n is the number of bits returned by the hash function). Merkle trees are designed so that a leaf value can be verified with respect to a publicly known root value given the authentication path connecting

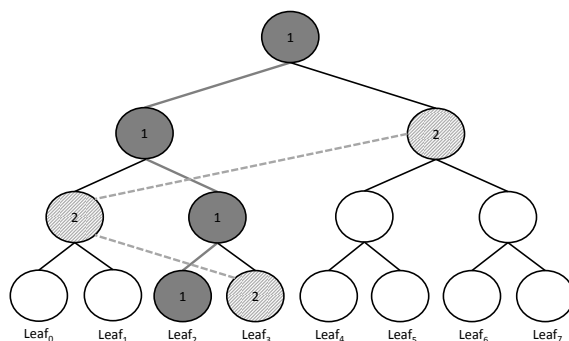


Figure 1: The nodes marked with a 1 are lying on the path from Leaf_2 to the root. The nodes marked with a 2 are the nodes of the authentication path for Leaf_2 ; all of them are siblings of a node marked with a 1.

the leaf to the root. The authentication path consists of one node value at each level l , where $l = 0, \dots, H - 1$, and H is the height of the Merkle tree ($H \leq 20$ in most practical cases). The chosen nodes are siblings of the nodes on the path connecting the leaf to the root (see Fig. 1).

The Merkle tree traversal problem answers the question of how to calculate efficiently¹ the authentication path for all leaves one after another starting with the first Leaf_0 up to the last Leaf_{2^H-1} , if there is only a limited amount of storage available (i.e. in Smartcards).

The generation of the public key (the root of the Merkle tree) requires the computation of all nodes in the tree. This means a grand total of 2^H leaves evaluations and of $2^H - 1$ hash computations. In this process the value of the public key is kept, the rest is discarded. The root value (the actual public key) is then stored into a trusted database accessible to the verifier.

The leaves of a Merkle tree are used either as a onetime token to access resources or as building block for a digital signature scheme. In the first case the tokens can be as simple as a hash of a pseudo random number generated by the provider of the Merkle tree. In the latter, more complex schemes are used in the literature (see for example [3] for a review).

The nodes in a Merkle tree are calculated with an algorithm called *TreeHash*. The algorithm takes as input a stack of nodes, a leaf calculation function and a hash-function and it outputs an updated stack, whose top node is the newly calculated node. Each node on the stack has a height i that defines on what level of the Merkle tree this node lies: $i = 0$ for the leaves and $i = H$ for the root. The *TreeHash* algorithm works in small steps. On each step the algorithm looks at its stack and if the top two elements have the same height it pops them and pushes the hash value of their concatenation

¹ Two children of the same node of a binary tree

back onto the top of stack which now represents the parent node of the two popped ones. Its height is one level higher than the height of its children. If the top two nodes do not have the same height, the algorithm calculates the next leaf and pushes it onto the stack, this node has a height of zero. To calculate all nodes in a tree of height H , the *Tree-Hash* needs $2^{H+1} - 1$ steps, where one step is either a leaf or a hash calculation.

Overview

Two solutions to the Merkle tree traversal problem exist. The first is built on the classical tree traversal algorithm but with many small improvements [8]. The second one is the fractal traversal algorithm [4]. The fractal traversal algorithm trades efficiently space against time by adapting the parameter h (the height of a subtree, see Fig. 2), however the minimal space it uses for any given H (if h is chosen for space optimality) is more than what the improved classical algorithm needs. The improved classical algorithm cannot as effectively trade space for performance. However, for small H it can still achieve a better time and space trade-off than the fractal traversal algorithm. But beyond a certain value of H (depending on the targeted time performance) the fractal traversal algorithm uses less space.

The idea of the fractal tree's traversal algorithm [4] is to store only a limited set of subtrees within the whole Merkle tree (see Fig. 2). They form a stacked series of L subtrees $\{Subtree_i\}_{i=0..L-1}$. Each subtree consists of an *Exist* tree $\{Exist_i\}$ and a *Desired* tree $\{Desired_i\}$, except for $Subtree_L$, which has no *Desired* tree. The *Exist* trees contain the authentication path for the current leaf. When the authentication path for the next leaf is no longer contained in some *Exist* trees, these are replaced by the *Desired* tree of the same subtree. The *Desired* trees are built incrementally after

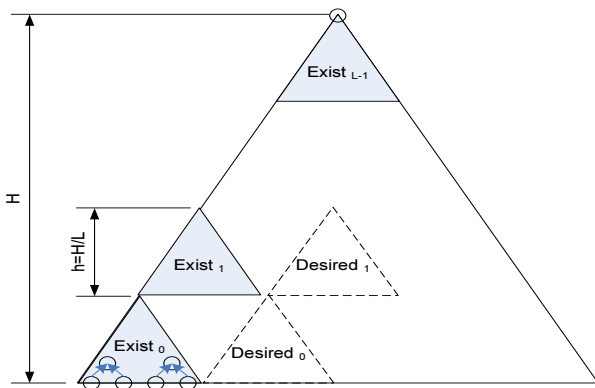


Figure 2: Fractal Merkle tree structure and notation (Figure courtesy of [2]). A hash tree T of height H is divided into L levels, each of height h . The leaves of the hash tree are indexed $\{0, 1, \dots, 2^h - 1\}$ from left to right. The height of a node is defined as the height of the maximal subtree for which it is the root and ranges from 0 (for the leaves) to H (for the root). An h -subtree is "at level i " when the altitude of its root is $h(i + 1)$ for some $i \in \{0, 1, \dots, L - 1\}$

each output of the authentication path algorithm, thus amortizing the operations needed to evaluate the subtree. During the initialization the values of the leftmost *Exist_i* trees are kept in addition to the root value.

We developed an algorithm for the Merkle tree traversal problem which combines the efficient space-time trade-off from [4] with the space efficiency from [8]. This was done by applying all the improvements discussed in [8] to the fractal tree's traversal algorithm [4].

The enhancements are as follows:

- Left nodes² have the nice property, that when they first appear in an authentication path, their children were already on an earlier authentication path. For right nodes³ this property does not hold. We can use this fact to calculate left nodes as soon as they are needed in the authentication path without the need to store them in the subtrees. So we can save half of the space needed for the subtrees, but one additional leaf calculation has to be carried out every two rounds.
- In most practical applications, the calculation of a leaf is more expensive than the calculation of an inner node. This can be used to design a variant of the *TreeHash* algorithm, which has a worst case time performance that is nearer to its average case for most practical applications. The modified *TreeHash* (see Algorithm 1) calculates in an update step, one leaf and as many inner nodes as possible before needing a new leaf, instead of processing just one leaf or one inner node as in the normal case.
- In the fractal Merkle tree one *TreeHash* instance per subtree exists for calculating the nodes of the *Desired* trees and each of them gets two updates per round (one round corresponds to the calculation of one authentication path). Therefore all of them have nodes on their stacks which need space of the order of $O(H^2/h)$. We can distribute the updates in another way, so that most *TreeHash* instances are empty or already finished. This reduces the space needed by the stacks of the *TreeHash* instances to $O(H - h)$.

It is easy enough to adapt point one and two for the approach discussed in [4], but point three needs some changes in the way the nodes in a subtree are calculated. All these improvements lead to an algorithm with a worst case storage of $[(H/h)(2^h - 1) + 2H - 2h]$ hash values. The worst case time bound for the leaf computation per round amounts to $(L - 1)(2^h - 1)/2^h + 1$.

² The authors of [7] proved that the bounds of space $O(tH/\log t)$ and time $O(H/\log t)$ for the output of the authentication path of the current leaf are optimal (t is a freely choosable parameter).

³ The left child of its parent node.

```

INPUT : stack<node>; leaf; processi
OUTPUT: updated stack

node := leaf
if (node.index mod 2) = 1 then
  continue := processi(node)
else
  continue := 1
end if
while (continue != 0) &
  (node.height = stack.top.height) do
  node := hash(stack.pop || node)
  continue := processi(node)
end while
if (continue != 0) then
  stack.push(node)
end if

```

Listing 1: Generic version of TreeHash that accepts different types of Process_i. Process_i manages nodes in dependence of the traversal algorithm used and returns true if more nodes are needed and false if the current iteration is finished. || is the concatenation operator.

We further implemented the algorithm in Java with focus on a low space and time overhead and we measured its performance.

Computation of the Desired Tree

The main difference between our algorithm and the one described in [4], is in the way we compute the nodes in the *Desired* tree. In our algorithm we use the improved *TreeHash* from [8] which needs 2^h steps to calculate a node on level h (instead of $2^{h+1} - 1$ as in [4]). We call the *TreeHash* instances which calculate the nodes on the bottom level⁴ of a *Desired* tree lower *TreeHash*. Another improved *TreeHash* instance, called higher *TreeHash*, calculates all non-bottom⁵ level nodes, by using the bottom level nodes as leaves in 2^h updates. In our case we do an update on the higher *TreeHash* all $2^{\text{BottomLevel}}$ rounds. The updates on the lower *TreeHash* are distributed with the algorithm described in [5]. The bottom level nodes which are produced by the lower *TreeHash* are ready when the higher *TreeHash* needs it as a leaf of the *Desired* tree (which happens every $2^{\text{BottomLevel}}$ rounds) [9]. In Figure 3 we show how the different nodes of the *Desired* and *Exist* trees are managed.

The space and time gains of the new algorithm

It can be shown that a subtree⁶ needs $2^h - 1$ hash values storage space when the authentication path is taken into account [9]. All subtrees together with the authentication path thus need $L(2^h - 1)$ for the subtrees (where $L = H/h$) and H hash values for the authentication path. The lower *TreeHashes* need to store at most one node per level up to the bottom level of the second highest subtree which sums up to $H - 2h$ hash values [9]. Taking all to-

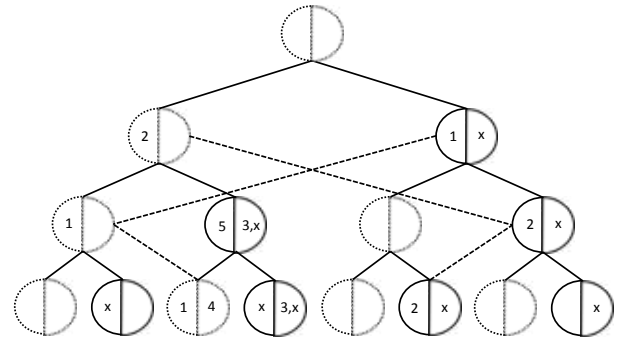


Figure 3: The left half of each circle represents the *Exist* tree and the right half the *Desired* tree. The dotted nodes are left nodes or the root and thus are not really stored in the subtree, but they may be stored in the authentication path or on the higher *TreeHash*. Nodes marked with an *x* are already discarded (in case of *Exist* tree) or not yet computed (in case of *Desired* tree). The nodes marked with a 1 are lying on the current authentication path; nodes marked with a 2 are lying on the upcoming authentication path. Nodes marked with a 3 are the nodes which are computed next by the higher *TreeHash*. The node marked with a 4 is a left node on the higher *TreeHash*. The node marked with a 5 is needed for calculating a left node in the upcoming authentication path.

gether our algorithm needs $L(2^h - 1) + 2H - 2h$ hash values storage space.

For the time analysis we look at the number of leaves' calculations per round, which often are way more expensive than the hash calculation. The improved *TreeHash* makes one leaf calculation per update and we make at most $(L - 1)$ lower *TreeHash* updates per round. The higher *TreeHash* never calculates leaves. So in the worst case both *TreeHash* need $(L - 1)$ leaves' calculations per round. We need an additional leaf calculation every two rounds to compute the left nodes as shown in [8]. If we sum this up, we need L leaves' calculations per round in the worst case. In the average case we need less, since the first $2^{\text{BottomLevel}}$ updates in each *Desired* tree do not compute any leaves. This because it would produce a left node which never is needed for the computation of a right node in the *Desired* tree. There are 2^h nodes on the bottom level of a subtree from which one node is not computed. In the average case the computation of the *Desired* trees is reduced by the factor $(2^h - 1)/2^h$. This leads to a total of $\frac{1}{2} + (L - 1)(2^h - 1)/2^h$ leaves' computations per round on the average (the $\frac{1}{2}$ term enter the equation because the left node computation needs a leaf only each two rounds). This average case analysis does not take into account that later some subtrees have no *Desired* trees and thus will no longer need an update on their *TreeHash*. Table 1 summarizes our results and Table 2 does the same for the log space- and fractal-algorithm.

When $h = 2$ our algorithm can compete with the log space-time [8] one, in the case of optimized storage space. Since our algorithm has the same improved space-time trade-off as the fractal one [4], it can handle more efficiently space vs. time

⁴ The right child of its parent node.

⁵ Lowest level for which a *Desired* tree stores nodes.

⁶ We mean with subtree the set of *Exist*, *Desired* and the higher *TreeHash* currently processed.

	$h = 1,$ $L = H$	$h = 2,$ $L = H/2$	$h = \log(H),$ $L = H/\log(H)$
Worst case space bound	$3H - 2$	$3.5H - 4$	$H(H - 1)/\log(H) + 2H - 2 \log(H)$
Average case time bound	$H/2$	$(3H - 2)/8$	$(H - 1)/\log(H) + \frac{1}{2}$
Worst case time bound	H	$H/2$	$H/\log(H) - 1$

Table 1: Space-time tradeoff of our Merkle tree traversal algorithm as function of h (number of levels) and H (height of the tree).

trade-offs in all other cases, where an optimized storage space is not required. When we choose the same space-time trade-off parameter as in the fractal algorithm [4] (column $h = \log(H)$ in Table 1), our algorithm needs less storage space.

Results

We compared the performance of our algorithm with both the algorithm from [8] (referred as Log from now on) and the algorithm from [4] (referred as Fractal from now on). We chose as performance parameters the leaves computations and the number of stored hash values. This choice is reasonable because the former is the most expensive operation if the Merkle tree is used for signing, and the latter is a good indicator of the storage space needed. Operations like computing a non-leaf node or generating a pseudo random value, have nearly no impact on the runtime in the range of H values of practical interest. A leaf computation is exactly the same in each of the three algorithms and therefore only dependent on the underlying hardware for its performance.

To be able to present cogently the results, each data point represents an aggregation over eight rounds. In the case of storage measurements one point represents the maximal amount of stored hash values at any time during these eight rounds. In the case of the leaf computation one point represents the number of leaves' computations done during the eight rounds.

For the log- and fractal-algorithms the parameters are chosen so that the algorithms use minimal storage space. The log algorithm is a good choice if a small memory footprint is needed. The fractal algorithm on the other hand, is a good choice if a better space time trade-off is needed. Our algorithm can be used in both cases but with different parameter settings. We have measured it once with the parameter chosen for a similar space time trade-off as the fractal tree (same number of levels) and once with a parameter setting for minimal storage space requirements. The results of these measurements for trees with 512 leaves ($H = 9$) are shown in the Figure 4 for a similar space-time trade-off as the fractal tree and in Figure 5 for minimal storage space.

	Log space-time [8]	Fractal [4]
Worst case space bound	$3.5H - 4$	$2\frac{1}{2} H^2/\log(H)$
Average case time bound	$H/2 - \frac{1}{2}$	$H/\log(H) - 1$
Worst case time bound	$H/2$	$2 H/\log(H) - 2$

Table 2: Space-time tradeoff of log space-time algorithm [8] and fractal algorithm [4] optimized for storage space.

We see that in a setting where a good space time trade-off is needed, our algorithm uses less space than the Fractal algorithm and this with just a small constant amount of additional leaves calculations. It uses more space as the log algorithm but it can reduce the number of leaves' calculations. For the case where a small memory footprint is needed, our algorithm uses most of the time less memory as the log algorithm, but computes more leaves.

The plots show in addition a weak point of our algorithm compared with the log algorithm: the leaves' calculations have larger deviations. The fractal algorithm has for similar parameter even greater deviations, but they are not visible in the Figure 5, because they cancel each other out over the eight rounds. If we measure the first 128 rounds with no aggregation we see, that the deviations of our algorithm decrease markedly (see Fig. 6).

References

- [1] Lov K. Grover. A fast quantum mechanical algorithm for database search. Proc. of the 28th Ann. Symp. on the Theory of Computing, 212 – 219, 1996.
- [2] Dalit Naor, Amir Shenhav, Avishai Wool. One-time signatures revisited: Practical fast signatures using fractal Merkle tree traversal. IEEE 24th Convention of Electrical and Electronics Engineers in Israel, 255 – 259, 2006.
- [3] J. Buchman, E. Dahmen, Michael Szydlo. Hash-based Digital Signatures Schemes. Post-Quantum Cryptography, 35 – 93, 2009.
- [4] Markus Jakobson, Frank T. Leighton, Silvio Micali, Michael Szydlo (2003). Fractal Merkle Tree representation and Traversal. Topics in Cryptology - CT-RSA, 314 – 326, 2003
- [5] Michael Szydlo, Merkle tree traversal in log space and time. In C. Cachin and J. Camenisch (Eds.): Eurocrypt 2004, LNCS 3027, pp. 541–554, 2004.
- [6] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, CRYPTO, volume 435 of LNCS, 218–238, 1989.
- [7] Piotr Berman, Marek Karpinski, Yakov Nekrich. Optimal trade-off for Merkle tree traversal. Theoretical Computer Science, volume 372, 26 – 36. 2007.
- [8] J. Buchman, E. Dahmen, M. Schneider. Merkle tree traversal revisited. PQCrypto '08 Proceedings of the 2nd International Workshop on Post-Quantum Cryptography Pages 63 – 78, 2008.
- [9] Markus Knecht. A space- and time-efficient Implementation of the Merkle Tree Traversal Algorithm. Master Thesis MSE FHNW, 2014.

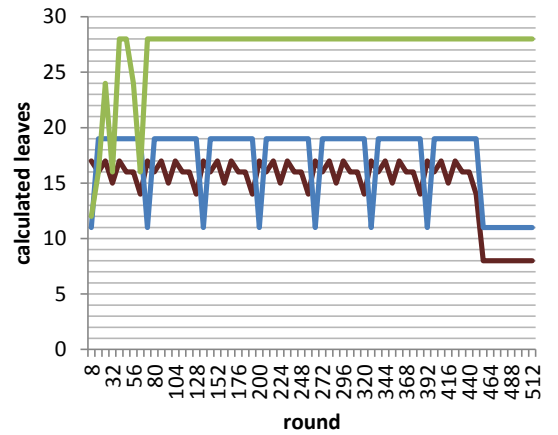
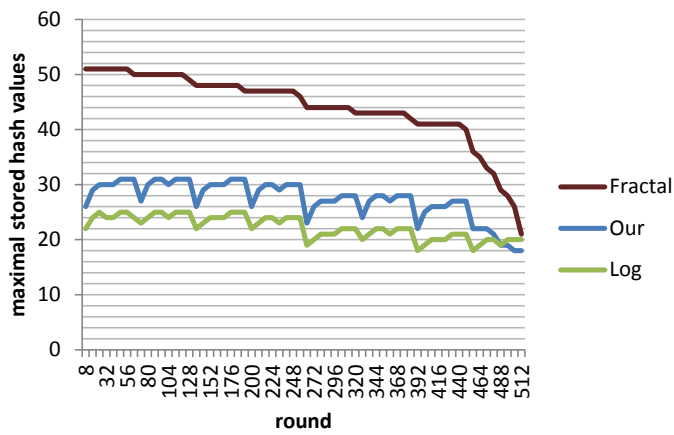


Figure 4: Left: Number of stored hash values as function of rounds for similar space-time trade-off.

Right: Number of calculated leaves as function of rounds for similar space-time trade-off.

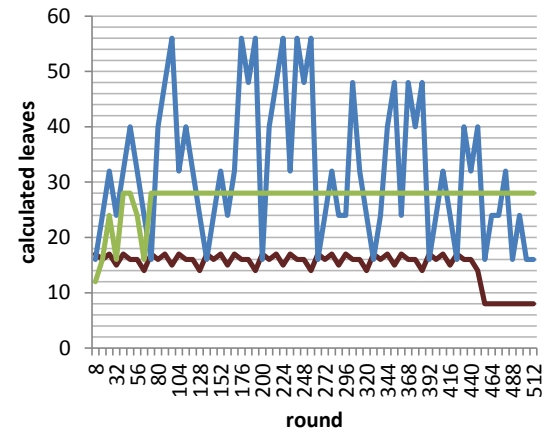
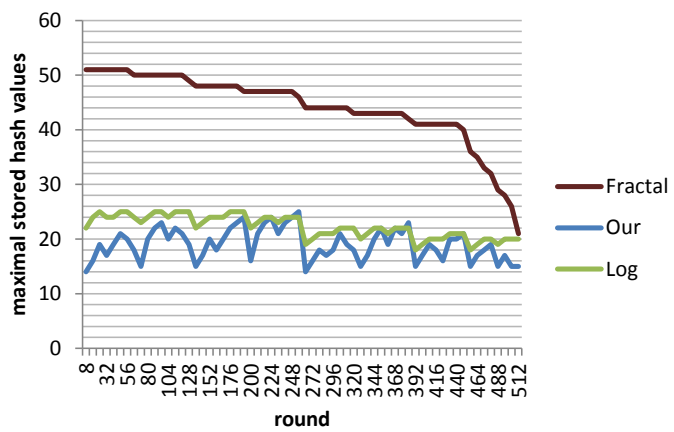


Figure 5: Left: Number of stored hash values as function of rounds for minimal space.

Right: Number of calculated leaves as function of rounds for minimal space

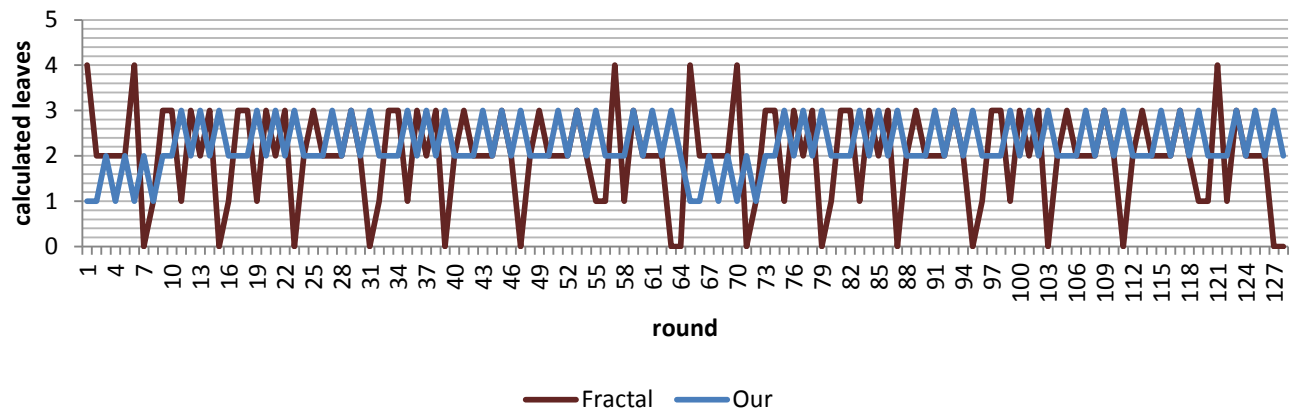


Figure 6: Number of calculated leaves as function of rounds for similar space-time trade-off (first 128 rounds in detail)



Fachhochschule Nordwestschweiz
Institut für Mobile und Verteilte Systeme
Bahnhofstrasse 6
CH-5210 Brugg-Windisch

www.fhnw.ch/technik/imvs
Tel. +41 56 202 99 33