Fachhochschule Nordwestschweiz
Hochschule für Technik

Institut für Mobile und Verteilte Systeme

FOKUS REPORT

2007

7.5
9 771662 201005

Vorwort

Liebe Leserin, lieber Leser

Am 1. Januar 2006 haben sich die drei Fachhochschulen FH Aargau, FH beider Basel und FH Solothurn zur zweitgrössten Fachhochschule der Schweiz, der Fachhochschule Nordwestschweiz FHNW, zusammengeschlossen. In dieser bewegenden Zeit ist auch unser Institut für Mobile und Verteilte Systeme IMVS entstanden.

Das Institut für Mobile und Verteilte Systeme hat sich in der Zwischenzeit gut positioniert. Neben der Ausbildung und Weiterbildung sind wir vor allem auch in der angewandten Forschung aktiv.

Im Zentrum unserer Forschungstätigkeit steht die Vision „Mobile Gesellschaft". Dies bedeutet, dass wir die zwei grossen Trends Mobilität und Vernetzung verfolgen und bearbeiten. Diese beiden Trends haben die Gesellschaft und die Informationstechnologie in den letzten Jahren geprägt und wir sind sicher, dass sie auch die Zukunft prägen werden; völlig neue Anwendungen werden entstehen und es werden sich unzählige Chancen für neue innovative Geschäftsprozesse eröffnen. Denn durch mobile Vernetzung lassen sich bestehende Prozesse vereinfachen und vollkommen neue Benutzergruppen erschliessen.

Mit den Beiträgen im vorliegenden IMVS Fokus Report greifen wir Themen aus unserem Fokus auf. Die Beiträge sollen Ihnen einerseits einen Einblick in unser Schaffen geben, anderseits aber auch die gute Beziehung zur Industrie, zur Wirtschaft und zu unseren Partnern vertiefen.

Ich wünsche Ihnen viel Spass beim Lesen und hoffentlich einige neue Erkenntnisse.

Fachhochschule Nordwestschweiz
Leiter Institut für Mobile und Verteilte Systeme

Prof. Dr. Jürg Luthiger

Inhalt

Impressum

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

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

Redaktion: Prof. Dr. C. Stamm
Layout: A. Hofmann
Erscheinungsweise: jährlich
Druck: jobfactory Basel
1. Auflage: 200

ISSN: 1662-2014

Editorial

Gut dreissig Jahre dauerte es, bis die von Carl Adam Petri in den frühen 60er Jahren postulierte Idee des Computers als Kommunikationsmedium umgesetzt wurde. Auf einmal sprachen alle nur noch von Internet, World-Wide-Web, XML, Web-Services usw. Alles drehte sich um Kommunikation, Dots, Netze und Inhalte. Innert fünfzehn Jahren schwappte der Hype vom Desktop zum Laptop zum PDA und zum Smartphone über. Dabei gewann der geneigte Beobachter der Szene den Eindruck, ein grosser Teil der Informatikergemeinde hätte vergessen, dass der zugrunde liegende Computer auch zur Daten-Verarbeitung und nicht nur zur Vermittlung bzw. Sammlung derselben dient. Doch zum Glück ist heute der oft heraufbeschworene Silberstreifen am Horizont sichtbar: Die mobilen Geräte werden dank vielfältiger Software erwachsen. Mobile Telefone werden zu interaktiven Multimedia-Centern und PDAs zu Navigatoren und Routenplanern. Die amputierten Mediatoren werden endlich zu umfassenden Datenverarbeitern!

In ihrem Artikel zum zukünftigen WLAN-Standard 802.11s breiten die beiden Autoren Andreas Hofmann und Claude Rubattel das vermaschte, drahtlose Netzwerk (Mesh-WLAN) aus und legen damit eine wichtige Grundlage für zukünftige, lokalitätsbasierte Anwendungen, welche z.B. auch ohne Satellitenempfang und damit in Gebäuden zum Einsatz kommen können. Sie zeigen, wie mit einfachsten Mitteln erste eigene Mesh-WLANs aufgebaut werden können. Dominik Gruntz und Severin Olloz nehmen den Trend der Location-Based-Services auf und zeigen in ihrem Beitrag anhand einer konkreten Implementierung den Umgang mit dem J2ME-Location-API. Dabei wird einmal mehr deutlich, wie gross die Unterschiede zwischen den einzelnen Gerätetypen sind und wie viel Zeit in die gerätespezifische Anpassung mobiler Software gesteckt werden muss.

Sobald sich mobile Benutzerinnen in einem Mesh-WLAN mit ihresgleichen über TCP/IP austauschen, werden sie als gleich berechtigte Partnerinnen, also als mobile Peers, gesehen. Mobiles Peer-to-Peer (P2P) bringt neben den herkömmlichen Kommunikationsmöglichkeiten und -schwierigkeiten auch das Potential mit, hoch spannende, verteilte Anwendungen zu realisieren, zum Beispiel im Zusammenhang mit Lokalisierungsinformationen. Dabei ist aber noch nicht eindeutig klar, wie solche P2P-Anwendungen aufgebaut sein sollen. Ein Versuch, in dieser Fragenstellung etwas Klarheit zu schaffen, hat Sun Microsystems mit dem von ihr lancierten Open-Source-Projekt JXTA unternommen. Peter Gysel und zwei seiner Studenten beschreiben in ihrem Beitrag, wie sie Basisdienste auf einheitliche Art und Weise anbieten. Hinter einer Service orientierten Schnittstelle greifen sie auf verschiedene Middleware-Technologien wie JXTA, Cajo und Java Message Service zu. Cajo baut auf Java RMI auf und ermöglicht somit die vereinfachte Kommunikation zwischen mehreren virtuellen Maschinen auf verteilten Rechnern. Gerade in P2P-Anwendungen, aber nicht nur, wirft die verteilte Bearbeitung von Daten, wie zum Beispiel das Nachführen eines Terminplans oder einer Adressdatenbank, die kritischen Probleme der Datensynchronisation und -replikation auf. Kritisch deswegen, weil die Daten für jeden Einzelnen sehr wichtig sind und fehlerhafte Synchronisation zu ungewünschten Effekten bis hin zu Datenverlust führen kann. Bernhard Wyss, Titus Jakob und zwei Studenten nehmen sich dieser Problematik an und zeigen anhand des Oracle Database Lite Produktes für mobile Geräte, wie sich Synchronisationskonflikte auf mobile Benutzer auswirken können.

Die Wahl der „richtigen“ oder besser der geeigneten Programmiersprache spielt auch bei der Entwicklung von mobilen Anwendungen eine wichtige Rolle. Ob C#, C++ oder Java hängt im mobilen Umfeld weniger von den Projektvorgaben als von der technischen Unterstützung der Gerätehersteller ab. Oliver Ruf weist in seinem Beitrag darauf hin, dass es nicht immer die herkömmlichen Programmiersprachen sein müssen, sondern auch mal Python sein darf. Er zeigt auf, in welchen Anwendungsbereichen eine Software-Entwicklung in Python viel Sinn machen und eventuell zu einer Effizienzsteigerung führen kann.

Unser Institut für Mobile und Verteilte Systeme (IMVS) ist aus der Fusion der drei Teilschulen FH Aargau, FH beider Basel und FH Solothurn hervorgegangen. Der Zusammenschluss liegt noch keine zwei Jahre zurück und daher sind die damaligen Ausrichtungen der Teilschulen immer noch gut spürbar. Unter diesem Aspekt ist der Schlussartikel dieses ersten IMVS Fokus Reports einzuordnen. Martin Schindler und Christoph Stamm gehen auf die zeitaufwändige Röhreninspektion von Kanalisationssystemen ein. Sie machen die Möglichkeiten und Effizienz moderner, digitaler Bildverarbeitungsverfahren deutlich und zeigen gleichzeitig, wie fruchtbar die Zusammenarbeit zwischen Industrie und Hochschule für beide Seiten sein kann.

Prof. Dr. Christoph Stamm
Forschungsleiter IMVS

Vermaschter Datenfunk

Maschennetze stellen eine interessante Netzwerktopologie dar. Wird diese Topologie bei Funknetzen verwendet, bilden sich flächendecke, ausfallsichere und sich selbstorganisierende Funknetze. In diesem Artikel wird beschrieben, welche Schritte die weit verbreitete WLAN-Technik bis heute gemacht hat. Weiter werden die grundsätzlichen Probleme, welche beim nächsten Entwicklungsschritt der WLAN-Technik hin zu Mesh-WLANs auftreten, mit ähnlichen Problemen bei den uns vertrauten Kabelnetzen verglichen. Es wird gezeigt, wie die Probleme bei den Kabelnetzen erfolgreich gelöst wurden und welche Ideen existieren, dieselben Probleme für Funknetze zu lösen. Die Erfahrungen beim Aufbau eines Mesh-WLANs werden beschrieben und ein Ausblick auf mögliche und geeignete Anwendeungen für Mesh-WLANs gemacht.

Andreas Hofmann, Claude Rubattel | andreas.hofmann1@fhnw.ch

Immer mehr Stadtgebiete werden heute flächendeckend mit drahtlosem Internet-Zugang (WLAN) versorgt. Dabei kommt vermehrt die Maschen-Technik zum Einsatz, weil dadurch im grossen Stil auf Kabelanschlüsse ans Internet verzichtet werden kann. So rüstet zum Beispiel Nortel den Campus eines nationalen Kunstzentrums in Taiwan mit einem Maschennetz aus [NOR07], oder Packethop erstellt im Bundesstaat New Jersey ein Maschennetz für die Polizei von Lakewood [PAC07]. Die Polizei soll auf diese Weise ein erweitertes Kommunikationsmittel erhalten. In der Stadt Cambridge baut das Massachusetts Institute of Technology (MIT) zusammen mit der Stadt ein Maschennetz nach einem System, welches am MIT selbst entwickelt worden ist [ROO07].

Auch in Europa ist man auf diesen anfahrenden Zug aufgesprungen. In Berlin entsteht seit einiger Zeit ein freies Funknetz, das sogenannte Freifunk-Netz [FRE07a]. Es macht sich die beiden Umstände zu Nutzen, dass einerseits die von 802.11x benutzten Frequenzbänder konzessionslos von allen benutzt werden dürfen und anderseits dass die Firma Linksys die Firmware eines ihrer Access-Points (AP) im Quellcode offen legen musste, nachdem bekannt wurde, dass für diese Firmware Opensource-Software eingesetzt wurde. Mittels ausgetauschter Firmware werden die normalen APs in Maschenknoten verwandelt, die sich selbständig zu einem Maschennetz vermaschen. Viele Privatpersonen installieren solche modifizierten APs auf den Dächern ihrer Häuser und so breitet sich das Netz ständig weiter über Berlin aus. Es spannt sich ein eigentliches Bürgernetz, das von keiner zentralen Infrastruktur abhängig ist. Ein Internet Service Provider (ISP) sorgt schliesslich dafür, dass einzelne der APs über einen Gateway mit dem Internet verbunden sind. Somit dienen die APs auf den Dächern den Benutzern als indirekten Zugang ins Internet.

Einfache Funknetze

Momentan bestehen hauptsächlich zwei Möglichkeiten, um Geräte mittels WLAN miteinander kommunizieren zu lassen [IEE03]. Zum einen ist dies der so genannte Ad-Hoc-Modus (auch Peer-To-Peer-Modus genannt), in dem meist zwei Geräte spontan ohne weitere Konfiguration eine gleichberechtigte Ende-zu-Ende Verbindung errichten. Dadurch können genau die beiden partizipierenden Geräte miteinander kommunizieren. Man nennt diesen Modus auch Independent Basic Service Set (IBSS).

Die zweite Möglichkeit besteht darin, dass ein spezielles Basisgerät – der Access-Point (AP) – eine Funkzelle mit einem bestimmten Radius spannt und mehrere Geräte mit WLAN-Anschluss (WLAN-Clients) diese Funkzelle nutzen, um untereinander in Kontakt zu treten (siehe Abb. 1). Dabei können zwei kommunizierende Clients weiter auseinander liegen als im Ad-Hoc-Modus, da der AP als Zwischenstation und Vermittler fungiert. Dieser Modus wird Infrastrukturmodus, manchmal auch Basic Service Set (BSS), genannt. Clients melden sich beim AP an, der diese verwaltet und koordiniert. Häufig sind APs gleichzeitig auch Router und können so Verbindungen in andere Netze, zum Beispiel ins Internet, herstellen.

Erweiterte Funknetze

Die Reichweite von WLAN-Funkzellen ist stark von der räumlichen Umgebung abhängig und beträgt zwischen 30 und 100 Metern. Das ist für viele Wohnungen ausreichend. Soll ein grösseres Firmengebäude oder gar ein ganzes Gelände damit abgedeckt werden, so ist diese Reichweite aber meistens zu gering. Es bestehen mehrere Möglichkeiten, eine grössere Abdeckung durch das WLAN zu erreichen. Allen Lösungen ist jedoch gemein, dass sie nicht mehr nur aus einem einzigen AP und einigen WLAN-Clients bestehen, sondern immer mehrere APs umfassen.

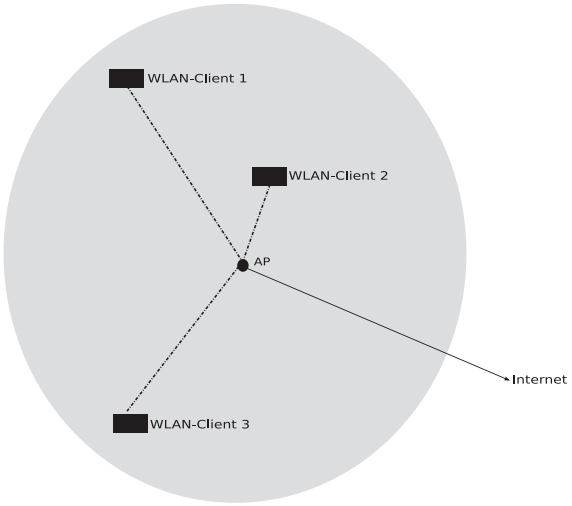


Abb. 1: Der Access Point (AP) spannt eine Funkzelle auf an der mehrere WLAN-Clients teilnehmen

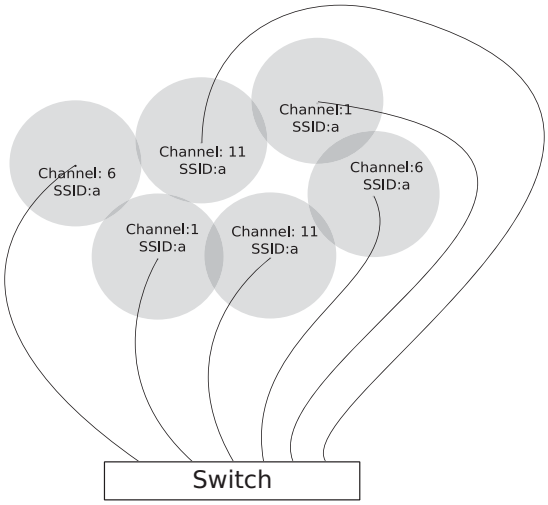


Abb. 3: Flächendeckendes WLAN mit kabelgebundenem Distribution System

Geht es lediglich darum, an jedem Ort via WLAN Zugriff auf das Internet zu realisieren, können mehrere APs mit Routing-Funktion und Internetanbindung so aufgestellt werden, dass alle relevanten Bereiche des Geländes mindestens durch eine Funkzelle abgedeckt sind. Der Nachteil dieser Lösung ist, dass zu jedem AP eine Internetanbindung geführt werden muss und dass die Gesamtheit aller WLAN-Zellen kein zusammenhängendes Netz bildet, in welchem alle eingebuchten WLAN-Clients untereinander erreichbar sind (siehe Abb. 2). Dieser Nachteil kann unter Verwendung eines gemeinsamen, kabelbasierten Backbones (Distribution Systems) wettgemacht werden (siehe Abb. 3). Bei einer solch modifiziernten Konfiguration spricht man nicht mehr vom Basic Service Set, sondern vom Extended Service Set (ESS). Es handelt sich beim ESS um eine Kopplung zweier oder mehrerer BSS. Das Distribution System wird meist über ein lokal vorhandenes Netzwerk, z.B. ein Ethernet, kabelgebunden realisiert. In einem derartigen ESS ist sogar ein ziemlich nahtloses Handover möglich, was bedeutet, dass sich WLAN-Clients im ganzen WLAN-Netz frei bewegen können, ohne dass die Verbindung beim Einbuchen an den nächsten AP unterbrochen wird. Ein weiterer Vorteil ist, dass infolge des Backbones alle WLAN-Clients untereinander erreichbar sind.

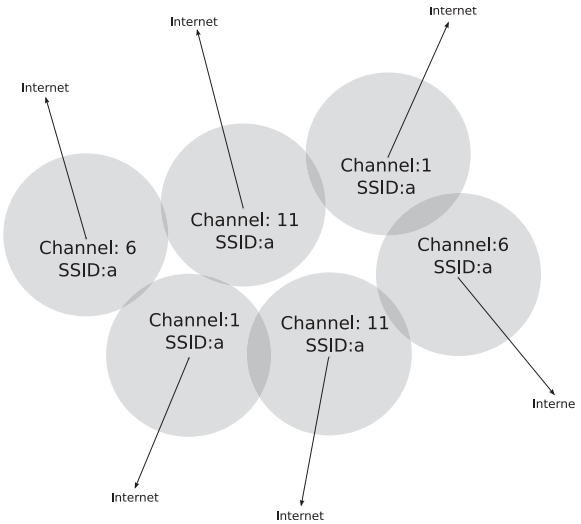


Abb. 2: Grossflächige WLAN-Abdeckung mit separatem Internet-Anschluss pro Funkzelle

Kabelloses Backbone

Ausgehend von der zuletzt beschriebenen Netzwerktopologie liegt es auf der Hand, das kabelgebundene durch ein kabelloses Backbone (Wireless Distribution System, WDS) zu ersetzen. APs, die heute schon ein solches WDS unterstützen, können meist entweder im Bridge-Modus oder im Repeater-Modus operieren. In beiden Fällen ist jedoch mit einem nicht unwesentlichen Konfigurationsaufwand zu rechnen.

Der Bridge-Modus dient dazu, z.B. zwei Ethernet-Switches kabellos miteinander zu verbinden (Point-to-Point); an jedem Switch ist ein AP angeschlossen, der via Funk die Verbindung zwischen den Switches herstellt (siehe Abb. 4). In diesem Betriebsmodus kann sich kein WLAN-Client bei den zwei ausgezeichneten APs einbuchen; die APs dienen exklusiv als Bridge.

Im Repeater-Modus kommunizieren alle APs und Clients auf dem gleichen Kanal. Jeder AP sendet alles was er empfängt an seinen benachbarten AP weiter (siehe Abb. 5). So kann die Reichweite des Funknetzes zwar vergrössert werden, doch reduziert sich die Übertragungsrate pro eingesetztem Repeater, da der gemeinsame Funkkanal infolge der Weiterleitung länger belegt bleibt.

Maschennetze

Ein Maschennetz ist grundsätzlich eine Netzwerktopologie, bei welcher jeder Knoten mit einem

Abb. 4: Zwei Access-Points im Bridge-Modus verbinden zwei Ethernet Switches

oder mehreren anderen Knoten verbunden ist. Bei einer vollständig vermaschten Topologie ist jeder Knoten mit jedem anderen verbunden. Das Ziel der Maschen-Funknetze ist es, Datennetze zu bilden, die sich durch einfaches Hinzufügen eines sogenannten Maschenknotens (eines speziellen AP) von alleine bilden und ohne Kabelverbindungen auskommen. Man spricht von mobilen Ad-Hoc-Netzwerken (MANET), wenn die Knoten auch noch mobil sein können. MANETs sind eine Unterkategorie von Maschennetzen. Maschennetze gelten als die ausfallsicherste Netzwerktopologie, die allerdings komplexe Routing-Algorithmen notwendig macht. Der Übergang in andere Netze, z.B. das Internet, soll nur an wenigen Maschenknoten realisiert werden müssen (theoretisch an genau einem) und von allen Teilnehmern als Gateway benutzt werden können. Es wird somit eine logische Kombination der beiden in Abb. 2 und Abb. 3 gezeigten Topologien angestrebt. Die APs sollen sich selbständig via Funk verbinden, damit ein durchgehendes Funknetz existiert, in welchem alle WLAN-Clients untereinander erreichbar sind.

Das Problem der Schleifen

Auf den ersten Blick scheint die funkbasierte Vermaschung der APs ein kleiner Schritt zu sein. Doch bereits bei genauerer Betrachtung erkennt man Probleme, wie sie bei kabelgebundenen LANs ebenfalls auftreten, wenn redundante Wegstrecken zur Erhöhung der Ausfallsicherheit geführt werden. Es geht bei den Problemen unter anderem um die Mehrfachübermittlung von Paketen, die ein Netz mit unnötigem Datenverkehr fluten und so das Funktionieren des Netzes gefährden.

Ein Ethernet-Netzwerk, wie in Abb. 6 dargestellt, würde sehr rasch mit Datenverkehr geflutet, da Pakete, die von LAN-Client 1 nach LAN-Client 2 gesendet werden, auf mehreren Pfaden ans Ziel gelangen können und bald im Kreis rotieren wür-

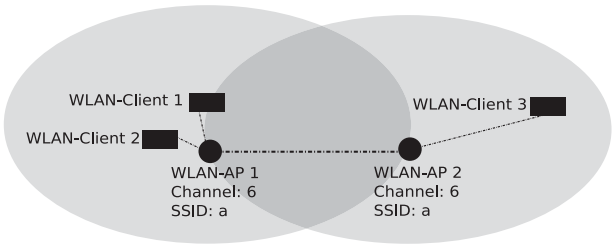


Abb. 5: WLAN-Zelle mit einem AP als Repeater, der die Reichweite erweitert

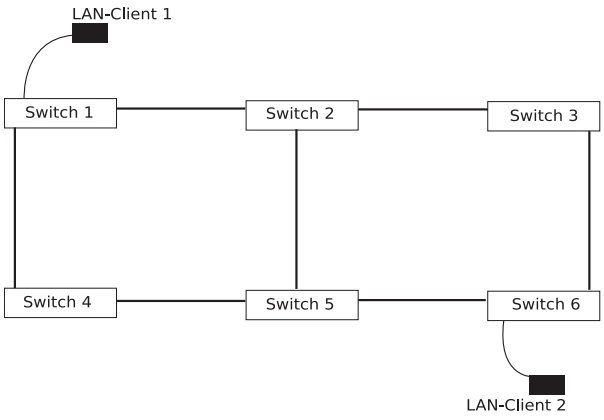


Abb. 6: Schleifen lassen ein Kabelnetz aus dem Tritt kommen

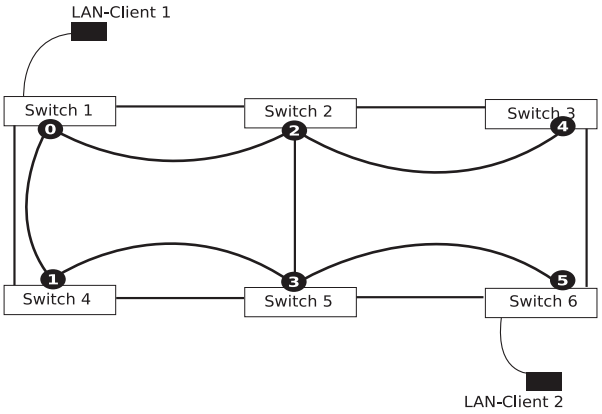


Abb. 7: Logische Topologie ohne Schleifen durch den Spannbaum

den. Die physikalische Topologie ermöglicht diese Situation und ohne getroffene Gegenmassnahmen wäre das Netz auch tatsächlich sofort überlastet. In kabelgebundenen LANs, in welchen aber genau solch redundante Verbindungen wie z.B. zwischen Switch 2 und Switch 5 oder zwischen Switch 3 und Switch 6 erwünscht sind, kommt ein Netzwerkprotokoll zum Einsatz, welches von allen Switches beherrscht werden muss. Es legt eine logische Netztopologie über die physikalische Topologie, so dass zu jedem Ziel genau ein Pfad führt. Das hierfür verwendete Protokoll bedient sich dem Spannbaum aus der Graphentheorie (siehe Abb. 7) und nennt sich denn auch Spanning Tree Protocol (STP). Es ist in der IEEE-Norm 802.1D standardisiert [IEE04]. Datenpakete werden dann nur noch auf den Pfaden des Spannbaumes transportiert. Fällt eine Verbindung in diesem Baum aus, so spannt das Protokoll automatisch einen neuen Spannbaum.

Falls keine geeigneten Protokolle verwendet werden, tritt das gleiche Überflutungsproblem auch in einem vermaschten WLAN-Netz auf, welches über ein kabelloses Backbone verfügt. Bei Funknetzen, zumal sie sich erst noch spontan bilden können und die Knoten mobil sind, eignen sich aber nicht dieselben Protokolle. Für vermaschte Funknetze eignet sich vor allem das Optimized Link State Routing.

Optimized Link State Routing

Beim Optimized Link State Routing Protokoll (OLSR) handelt es sich um eine optimierte Variante des bekannten Link State Routing Protokolls, welches die kürzesten Wege in einem kantengewichteten Graphen bestimmt [CLA03]. Die Gewichtung der Kanten wird als Distanz bezeichnet und wird mittels der Antwortzeit eines Nachbarknotens auf eine Kontrollmeldung berechnet. OLSR benötigt keine zentrale Verwaltungsstelle. Die Topologie bildet sich, indem die Knoten periodisch Kontrollmeldungen senden, um gegenseitig Nachbarschaftsinformationen auszutauschen. Jeder Nachbar muss umgehend auf diese Meldungen antworten. Mit diesen Nachbarschaftsinformationen erstellt sich jeder Knoten seine eigene Routing-Tabelle und kann mit einem geeigneten Algorithmus den kürzesten Weg zu jedem Knoten berechnen.

Die Optimierung von OLSR gegenüber dem Link State Routing liegt darin, dass jeder Knoten aus den direkten Nachbarn nur eine beliebige Untermenge auswählt, und diese in sein sogenanntes Multi Point Relay Set (MPR) stellt (siehe Abb. 8). Wenn ein Knoten eine Meldung versendet, so wird diese nur von jenen Nachbarknoten weitergeleitet, die sich im MPR-Set des sendenden Knoten befinden. Die Idee der MPRs ist also, die Broadcast-Meldungen beim Fluten des Netzes zu reduzieren.

802.11s der neue Standard für Maschen-WLANs

Bis zum heutigen Zeitpunkt ist noch kein Standard verabschiedet worden, der die Funktionsweise von Maschen-Funknetzen beschreibt. Doch mit IEEE 802.11s ist eine Teilspezifikation des Industriestandards 802.11 am Entstehen, der es gestatten soll, dass Geräte unterschiedlicher Hersteller Maschennetze mittels WLAN spannen können [IEE07]. Im Juni 2005 sind 15 Vorschläge für diesen Standard eingereicht worden, von denen zwei den grössten Zuspruch hatten. Zum einen handelt es sich um die SSEMesh-Gruppe, in der hauptsächlich die Firmen Cisco und Intel dabei sind. Zum anderen gibt es die Wi-Mesh-Allianz mit den Firmen Philips, Swisscom Innovation und Nortel. Die Vorschläge dieser beiden Gruppen sind im Januar 2006 vereint worden und das Resultat dieser Verschmelzung dient momentan als Ausgangslage zur Erarbeitung des 802.11s Standards. Es wird erwartet, dass dieser im Jahr 2009 verabschiedet wird.

Heute gibt es erste Hersteller, die Maschen-Access-Points mit einem Pre-Standard 802.11s anbieten und deren Geräte noch als proprietär zu bezeichnen sind. Zu nennen sind unter anderem die Hersteller Motorola, Cisco, Firetide, Nortel, Packethop, Belair und auch Meraki. Viele der erhältlichen Mesh-WLAN-APs können auch als herkömmliche WLAN-Clients eingesetzt werden. Selbst das Handover wischen den verschiedenen Zellen beherrschen einige Geräte.

Eigene Erfahrungen mit Mesh-WLAN

Wir haben an zwei verschiedenen Standorten separate Mesh-WLANs auf der Basis des Berliner Freifunkprojektes aufgebaut [FRE07b]. Als geeigneter AP wird in diesem Projekt der WRT54G-Router der Firma Linksys empfohlen, unter an-

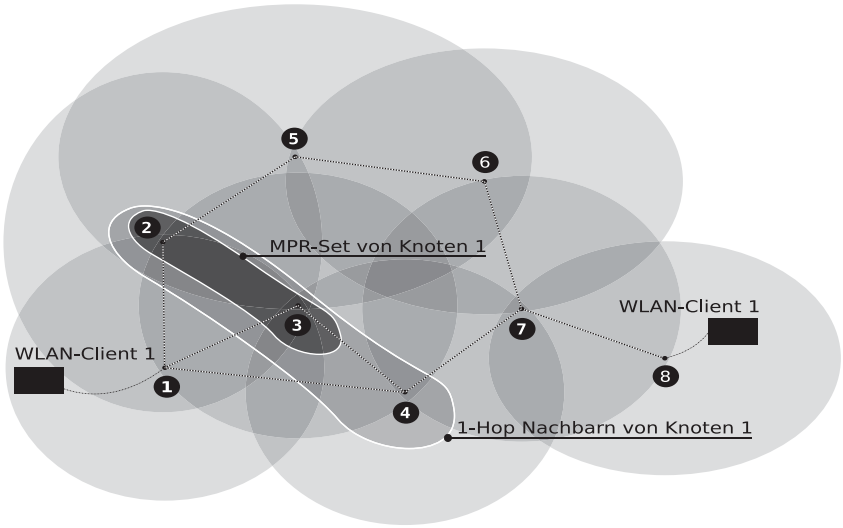


Abb. 8: OLSR hat eine logische Netztopologie gespannt

derem weil er die Anpassung der Sendeleistung zulässt und weil der Quellcode für die Original-Firmware von Linksys veröffentlicht worden ist. Dadurch sind eine Vielzahl von erweiterten Firmwares entstanden (BatBox [WRT07b], Wifibox [SFN07a], Sveasoft [SVE07] und OpenWRT [WRT07a]), die der WRT54G-Familie zusätzliche Funktionen bescheren, welche sonst nur bei viel teureren Routern vorhanden sind [FRE07a]. Die Routers dieser Familie unterstützen Internetverbindungen mit mehreren Clients über Ethernet 802.3, WLAN 802.11b sowie WLAN 802.11g

In unseren beiden Netzwerken verwenden wir APs vom Typ WRT54GL (L steht für Linux, Version 4 mit BCM5352E Chipset und 200-MHz-CPU). Dies weil die Weiterentwicklung vom WRT54G keine erweiterte Firmware mehr zulässt. Diese Weiterentwicklung hat dazu geführt hat, dass ab Version 5.0 der Router anstelle von Linux nur noch mit dem weniger ressourcenintensiven Betriebssystem VxWorks der Firma Wind River Systems läuft. Sie lässt momentan nur die originale Firmware von Linksys zu.

Ein Ziel von Maschen-Funknetzen kann es sein, Datennetze auf Bedarf hin zu bilden oder anzupassen. Soll sich also das Maschennetz durch einfaches Hinzufügen eines APs von alleine neu organisieren, so müssen drei Voraussetzungen erfüllt sein. Erstens müssen die APs im Ad-Hoc-Modus betrieben werden, so dass jeder AP direkt mit jedem anderen AP kommunizieren kann. Zweitens müssen alle APs dieselbe ESSID verwenden. Die ESSID (Extended Service Set Identifier) erlaubt die eindeutige Identifikation des WLANs und wird benötigt, damit die Clients in einem erweiterten Funknetz den richtigen AP finden. Drittens müssen die APs die erweiterten Routing-Funktionen anbieten. Üblicherweise bauen die Hersteller diese aber nicht in ihre Geräte ein, so dass man mit einem Austausch der Firmware selber dafür sorgen muss.

In unseren beiden Netzwerken haben wir uns für die Firmware OpenWRT entschieden. Diese wird bei Lösungen empfohlen, die auf Optimized Link State Routing Protokoll (OLSR) basieren. OpenWRT ist eine Art Mini-Linux, das die Möglichkeit bietet, neue Pakete zu installieren [FRE07a].

Firmware-Upgrade und Betrieb

Bevor wir unsere APs (alle vom Typ WRT54GL) konfigurieren und in Betrieb nehmen können, tauschen wir bei jedem einzelnen AP die Firmware aus. Die Firmware wird unter [FRE07b] angeboten und kann ganz einfach auf den APs installiert werden. Alles was man tun muss ist, die Firmwaredatei bei sich lokal auf dem Computer bereit zu haben und die IP-Adresse des AP zu kennen, um so via Webbrowser auf die AP-Konfigurationswebseite zu gelangen. Beim WRT54GL findet man den Menupunkt „Administration“ und weiter den Punkt „Firmware Upgrade“. Wir mussten dann die sich auf unserem lokalen Computer befindende Firmware Binärdatei auswählen und den Upgrade Prozess starten. Alles was man jetzt noch tun muss, ist acht zu geben, dass der Vorgang nicht durch einen Stromausfall unterbrochen wird. Eine genaue Anleitung befindet sich in [FRE07d].

Die anschliessende Konfiguration der APs verläuft standardmässig: Kanäle, Netzwerkmodi, Gerätenamen und Service Set Identifier (SSID) wählen. Für die Beschränkung des Zugriffs kann zudem der übliche Sicherheitsmechanismus aktiviert werden. Danach sind die APs betriebsbereit und es reicht, einen von ihnen über ein Kabel ans Internet anzuschliessen. Die andern müssen nur noch strategisch platziert werden, wobei darauf geachtet werden muss, dass sie in Verbindung

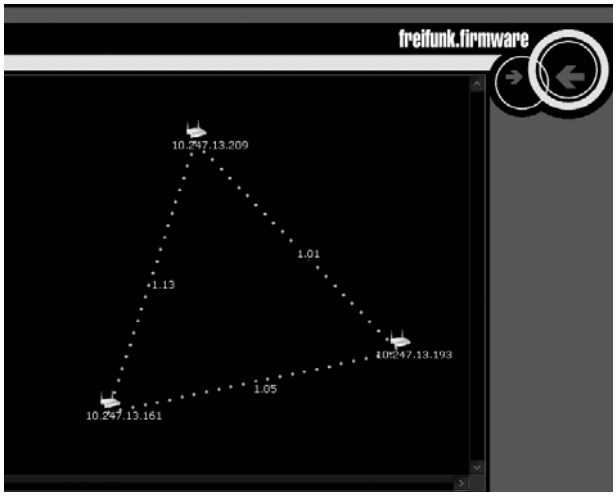


Abb. 9: Vermaschung von 3 APs

mit ihren Nachbarn bleiben. Ab diesem Zeitpunkt können mobile WLAN-fähige Geräte (Laptop, Pocket PC, usw.) eine drahtlose Netzwerkverbindung mit einem der verfügbaren AP aufbauen und eine Netzwerkadresse beziehen.

Das vermaschte Netzwerk lässt sich nun im Browser eines Rechners mit Internet-Zugang visualisieren. Dazu gibt man als Web-Adresse die IP-Nummer des zuvor ans Internet angeschlossenen APs ein. Daraufhin öffnet sich eine Webseite „freifunk.firmware“ und mit der Auswahl „OLSR-Viz“ erscheint eine Graphik des vermaschten WLAN-Netzwerks (siehe Abb. 9).

Ausblick auf mögliche Anwendungen

Mobile drahtlose Maschennetze ermöglichen interessante Anwendungen. So ist es zum Beispiel möglich, gerade an dem Ort, wo die nötige Infrastruktur fehlt, ein spontanes Funknetz zu errichten. Denkbare Szenarien sind Katastrophenfälle, wo in einem entlegenen Gebiet viele Hilfskräfte auf einem eng begrenzten Gebiet erscheinen und ein spontan gebildetes Datennetz von Nutzen ist, um die notwendigen Handlungen untereinander abzusprechen und Informationen auszutauschen.

Ein anderes schönes Beispiel, in welchem der einfache und kostengünstige Aufbau der Kommunikationsinfrastruktur im Vordergrund steht, zeigt das 100-Dollar-Laptop-Projekt, welches von der gemeinnützigen Organisation „One Laptop Per Child“ unter Vorsitz des MIT Professors Nicholas Negroponte gegründet wurde. Der dabei entwickelte Laptop verfügt ebenfalls über Maschen-WLAN. Er soll vor allem in Entwicklungsländern zum Einsatz kommen, wo keine vorhandenen Netzinfrastrukturen vorliegen, ein Austausch unter den Benutzern aber genauso gefragt ist und der Anschluss eines einzigen Laptops ans Internet allen den Zugang zum World Wide Web erlaubt.

Alleine durch die Tatsache, dass in einem Maschennetz die Abstände zwischen den Maschenknoten klein sind und sich mehrere Funkzellen überlappen müssen, wird unter der Voraussetzung der bekannten Position der festen APs eine recht genaue Lokalisierung der Teilnehmer auch ohne Zugriff auf Satelliten möglich. Genau diese Lokalisierung gestattet den mobilen Nutzern neuartige, ortsbezogene Anwendungen einzusetzen. Eine Fahrplananfrage für die nächstgelegene Haltestelle des öffentlichen Verkehrs könnte dann wesentlich komfortabler als heute geschehen, indem zum Beispiel nur noch der Zielort eingegeben werden muss. Das zu verwendende Transportmittel, allfällige Umsteigestationen und die Dauer bis zur nächstmöglichen Abfahrt können sofort angezeigt werden.

Ganz allgemein gilt: Die Ortsbezogenheit kombiniert mit anderen innovativen Ideen wird eine Vielzahl von innovativen Anwendungen ermöglichen.

Referenzen

[CLA03] T. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR), 2003

[FRE07a] J. Neumann, M. Behling, Was ist freifunk?, 2007

[FRE07b] Freifunk, http://freifunk.net, 2007-03-14

[FRE07c] Freifunk, http://freifunk.net/wiki/WikiStartSeite, 2007-03-14

[FRE07d] Freifunk, Linksys WRT54G Installation besserer Firmware, http://wiki.freifunk.net/Linksys_WRT54G_Installation_besserer_Firmware, 2007-03-14

[IEE03] IEEE 802.11 working group, ANSI/IEEE Std 802.11, 1999 Edition (R2003), 2003

[IEE04] IEEE 802.1 working group, IEEE Standard for Local and metropolitan area networks Media Access Control, 2004

[IEE07] IEEE 802.11s working group, Status of Project IEEE 802.11s, 2007-03-14

[NOR07] J. Lu, Taiwan‘s national center for traditional arts deploys nortel wireless mesh, 2006

[PAC07] PacketHop, Lakewood Police deploys New Jersey‘s First 4.9 GHz Broadband Mo-bile-mesh, 2007

[ROO07] D. Aguayo, J. Bicket, S. Biswas, D. S. J. De Couto, R. Morris, MIT Roofnet Imple-mentation, 2003

[SFN07a] WIFI-BOX - WRT54G(s) GPL Firmware, http://sourceforge.net/projects/wifi-box, 2007-03-14

[SVE07] http://www.sveasoft.com, 2007-03-19

[WRT07a] What is OpenWRT?, http://openwrt.org, 2007-03-14

[WRT07b] Linux on the WRT54G, http://www.batbox.org/wrt54g-linux.html, 2007-03-14

Erfahrungen mit dem Location API (JSR 179)

Location Based Services (LBS) sind Dienste, welche vom Standort des Benutzers abhängen. Eine typische Anwendung solcher Dienste ist ein Touristenführer welcher auf Sehenswürdigkeiten in der Nähe hinweisen kann. Mit dem J2ME Location API (JSR 179) ist für die Java Plattform ein API definiert worden, welches J2ME-Anwendungen erlaubt, auf Lokalisierungsinformation zuzugreifen. Erste Geräte unterstützen das JSR 179. In diesem Artikel wird auf die Implementierung des JSR 179 in einem konkreten Gerät (Nokia N71) eingegangen und es wird ein Simulator vorgestellt, mit dem das Location API getestet werden kann.

Dominik Gruntz und Severin Olloz | dominik.gruntz@fhnw.ch

Unter Lokalisierung versteht man die Bestimmung der eigenen Position in einem Koordinatensystem. Im Zusammenhang mit mobilen Geräten wird häufig ein ellipsoides, geografisches Koordinatensystem verwendet, mit welchem die Position in Längengrad (östlich oder westlich von Greenwich), Breitengrad (nördlich oder südlich des Äquators) und der Höhe über Meer angegeben wird. Die Koordinaten beziehen sich dabei auf ein zugrundeliegendes Referenzsystem. Ein weit verbreitetes Referenzsystem ist das World Geodetic System 1984 (WGS84).

Die Position des mobilen Gerätes kann mit Hilfe von Winkeln und Entfernungen zu Referenzpunkten bestimmt werden. Eine typische Methode ist das satellitenbasierte GPS (Global Position System). Um die Position mit GPS zu bestimmen, müssen die Abstände zu drei Satelliten und eine Referenzzeit von einem vierten Satelliten bekannt sein. Die Position wird bei GPS anhand der Laufzeit eines Funksignals bestimmt (Time of Arrival, TOA). Mit GPS kann die Position auf ca. 15 Meter genau bestimmt werden. Mit Hilfe eines Korrektursignals, das von einer Basisstation ausgesendet wird, kann die Genauigkeit auf eins bis fünf Meter verbessert werden (Differential-GPS). Ein Nachteil des GPS-Verfahrens ist, dass es ohne Satellitenempfang (z. B. in Gebäuden) nicht funktioniert.

Eine zweite bekannte Technologie ist die Positionsbestimmung auf der Basis von Zelleninformationen der GSM- und UMTS-Funknetze (Cell of Origin, COO). Das mobile Gerät kann feststellen, mit welcher Basisstation es verbunden ist. In städtischen Gebieten haben die Zellen einen Durchmesser von ca. 300 Metern, in ländlichen Gebieten jedoch über 20 Kilometer. Je grösser der Durchmesser der Zelle, umso ungenauer ist diese Art der Positionsbestimmung. Der Vorteil dieser Methode ist jedoch der geringe Stromverbrauch und die Tatsache, dass es auch in Häusern funktioniert.

In der Schiffs- und Luftfahrt wird die Position auf Grund von Eingangswinkeln (Angel of Arrival, AOA) von Funksignalen gemessen. Dazu sind jedoch mehrere Antennen oder eine drehende Antenne notwendig.

Für die Lokalisierung in Gebäuden werden Methoden verwendet, welche die Signalstärke von Sendern messen, deren Standort und Sendestärke bekannt ist. Da die Stärke des Signals mit zunehmender Entfernung vom Sender abnimmt, kann daraus die Distanz abgeschätzt werden. Als Funksignale werden entweder WLAN oder Bluetooth verwendet. Ein Problem bei dieser Methode ist, dass die Signalstärke und somit auch die Distanzabschätzung von Hindernissen beeinflusst werden.

Eine Methode für die hochgenaue Ortung (ca. 30 Zentimeter) basiert auf dem Ultra Wide Band (UWB). Als Technologie wird dabei entweder die Differenz der Laufzeit von Signalen, welche von verschiedenen Referenzstationen gleichzeitig gesendet werden (Time Difference of Arrival, TDOA), oder eine Kombination der Methoden TOA und AOA verwendet.

Das J2ME Location API definiert eine einheitliche und einfache Schnittstelle, über die Java-Anwendungen auf Lokalisierungsinformation zugreifen können. Das API abstrahiert dabei von den unterschiedlichen Lokalisierungsmethoden.

Java Location API (JSR 179)

Das J2ME Location API [JSR179] ist ein optionales Paket, welches im Rahmen des Java Community Prozesses (JCP) als Java Specification Request (JSR) 179 definiert worden ist. Dieses Paket kann mit unterschiedlichen J2ME-Profilen verwendet werden (z. B. Mobile Information Device Profile (MIDP) 1.0 oder MIDP 2.0). Als minimale Plattform wird die Connected Limited Device Configuration (CLDC) 1.1 verlangt, da das Location API mit Gleitpunktzahlen arbeitet. Das Location API ist

```
Criteria crit = new Criteria();
      crit.setHorizontalAccuracy(100);        // 100m
      crit.setVerticalAccuracy(100);          // 100m
      crit.setPreferredResponseTime(Criteria.NO _ REQUIREMENT);
      crit.setPreferredPowerConsumption(Criteria.POWER _ USAGE _ HIGH);
      crit.setCostAllowed(false);
      crit.setSpeedAndCourseRequired(true);
      crit.setAltitudeRequired(true);
      crit.setAddressInfoRequired(true);

      LocationProvider provider = LocationProvider.getInstance(crit);
      if (provider != null) {
           Location loc = provider.getLocation(60); // timeout von 60 Sek
           Coordinates c = loc.getQualifiedCoordinates();
           if (c != null) {
                 longitude = c.getLongitude();
                 latitude = c.getLatitude();
                 altitude = c.getAltitude();
           }
      }
```

Listing 1: Zugriff auf Lokalisierungsinformation

auch ein bedingt zwingender Teil der „Mobilen Service Architektur“ (MSA) [JSR248]. Die MSA legt fest, welche JSRs von einem MSA kompatiblen Gerät unterstützt werden müssen. Als Teil der MSA-Spezifikation muss das Location API von einem MSA-Gerät unterstützt werden, welches einen GPS-Empfänger enthält oder welches mit anderen Methoden Lokalisierungs-information über ein API anbietet. Eine Liste von Geräten, welche das JSR 179 unterstützen, ist unter [SDN07] oder [POL07] verfügbar. Es handelt sich etwa um 30 Geräte.

Das Java Location API besteht aus zwei Schnittstellen und elf Klassen (davon zwei Exception Klassen). Im folgenden Java Programmauszug (Listing 1) wird die aktuelle Position des Gerätes ermittelt. Mit der Methode LocationProvider.getInstance wird eine Lokalisierungsmethode angefordert. Als Parameter können Kriterien definiert werden, die von der Lokalisierungsmethode erfüllt werden müssen. Die Kriterien spezifizieren z. B. die Genauigkeit der Lokalisierungsmethode, eine maximale Antwortzeit oder eine Angabe über den erlaubten Stromverbrauch. Falls kein Provider existiert, der die verlangten Kriterien erfüllen kann, dann wird null zurückgegeben.

Auf der von der Methode getLocation zurückgegebenen Location Instanz kann die Position im WGS84 System abgefragt werden. Über weitere Methoden können Geschwindigkeit und Richtung (sofern diese Information verfügbar ist) sowie Informationen über die Lokalisierungsmethode abgefragt werden. Folgende Bits, welche die Lokalisierungsmethode beschreiben, können gesetzt sein:

1. MTE_TIMEDIFFERENCE
   TDOA Methode
   (für zelluläre oder terrestrische RF Systeme)
2. MTE_ANGLEOFARRIVAL
   AOA Methode
   (für zelluläre oder terrestrische RF Systeme)
3. MTE_TIMEOFARRIVAL
   TOA Methode
   (für zelluläre oder terrestrische RF Systeme)
4. MTE_CELLID
   COO Methode für zelluläre Systeme
5. MTE_SATELLITE
   Satellitenbasierte Lokalisierung (z. B. GPS)
6. MTE_SHORTRANGE
   Nahbereichslokalisierung
   (z. B. mit Bluetooth, UWB)

Das Location API wird von den Geräteherstellern implementiert. Diese entscheiden damit, welche Lokalisierungsmethoden sie bereitstellen. Das Location API erlaubt jedoch nicht abzufragen, welche Methoden unterstützt werden (und welche Kriterien diese noch erfüllen können).

Implementierung von Nokia

Wir haben die Implementierung des JSR 179 des mobilen Nokia-Telefons N71 genauer untersucht. Informationen zur Implementierung des Location API für das N71 und andere S60-Geräte findet man in den Implementation Notes [Nok07].

Beim Aufruf LocationProvider.getInstance wird auf dem Nokia N71 eine Instanz der Klasse com.nokia.mid.impl.symbian.location.LocationProviderImpl zurückgegeben. Unsere Untersuchungen haben gezeigt, dass diese Klasse zumindest die zwei Lokalisierungsmethoden SATELLITE (GPS) und CELLID (COO) unterstützt.

Lokalisierung mit GPS

Das Mobiltelefon N71 enthält keinen integrierten GPS-Empfänger. Daher wird bei der Lokalisierung mit GPS auf einen externen GPS-Empfänger über Blootooth zugegriffen. Als Protokoll zwischen dem mobilen Gerät und dem GPS-Empfänger wird ein Protokoll der National Marine Electronics Associ-

ation (NMEA) verwendet. Dieses NMEA-Protokoll findet bei verschiedenen Positionierungssystemen Verwendung. Grosse Verbreitung hat die im Jahr 1983 verabschiedete Version 0183 [NMEA]. Dieser Standard verwendet eine RS-422-Schnittstelle, um mit der Aussenwelt mit einer Geschwindigkeit von 4800 Baud zu kommunizieren. Bei heutigen mobilen Geräten werden die GPS-Empfänger nicht mehr über ein serielles Kabel angeschlossen, sondern über eine Bluetooth-Funkverbindung, die bei einer maximalen Bandbreite von 723,2 kBit/s ohne Probleme mit den anfallenden NMEA-Daten fertig wird. Die Datenübertragung läuft in kleinen Datensätzen ab, wobei verschiedene Datensätze existieren. Ein Beispieldatensatz könnte wie folgt aussehen: $GPRMC,162614,A,5230.5900,N,01322.3900,E,10.0,90.0,131006,1.2,E,A*13<CR><LF>

Ein Datensatz beginnt mit einem Dollarzeichen ($) gefolgt von einer Satzkennung und endet mit einem Stern (*) gefolgt von einer Checksumme und einem Carriage Return (CR) und einem Line Feed (LF). Ein Datensatz kann maximal 80 Zeichen enthalten. Im oben gezeigten Beispiel definiert GPRMC eine Recommended Minimum Specific (RMC) Nachricht eines GPS-Empfängers. Zwischen der Satzkennung und dem Stern stehen die eigentlichen Daten durch Kommata getrennt.

Das Nokia N71 unterstützt diese Methode, wenn die an den Location-Provider gestellten Kriterien folgende Bedingungen erfüllen:

1. Horizontal Accuracy ≥ 10m (oder NO_REQUIREMENT)
2. Vertical Accuracy ≥ 30m (oder NO_REQUIREMENT)
3. Preferred Response Time ≥ 1000ms (oder NO_REQUIREMENT)
4. Power Usage ≥ MEDIUM (oder NO_REQUIREMENT)
5. Cost Allowed egal (true oder false)
6. Speed and Course required egal (true oder false)
7. Altitude required egal (true oder false)
8. Address Info required nein (false)

Wenn die Position mit dieser Methode bestimmt wird, so kann auf dem vom Location-Provider zurückgegebenen Location-Objekt mit der Methode getExtraInfo(„application/X-jsr179-location-nmea“) auf die NMEA Datensätze zugegriffen werden, aus der die Position bestimmt worden ist.

Lokaliserung über die Zelleninformation des Operators (COO)

Bei COO bezieht das Mobiltelefon N71 die Lokalisierungsinformation vom Telecom Provider. Dazu wird gemäss [Loy07] ein Mobile Originated Location Request (MO-LR) basierend auf dem 3GPP Location Services (LCS) Protokoll [3GPP] über den Signalisierungskanal (control-plane) an den Operator geschickt.

Das Nokia N71 unterstützt diese Methode, wenn die an den Location-Provider gestellten Kriterien folgende Bedingungen erfüllen:

1. Horizontal Accuracy ≥ 200m (oder NO_REQUIREMENT)
2. Vertical Accuracy ≥ 1m (oder NO_REQUIREMENT)
3. Preferred Response Time ≥ 12000ms (oder NO_REQUIREMENT)
4. Power Usage = LOW
5. Cost Allowed ja (true)
6. Speed and Course required nein (false)
7. Altitude required nein (false)
8. Address Info required nein (false)

Leider unterstützt keiner der Telecom Anbieter, die wir austesten konnten, dieses Protokoll bzw. stellt keiner die Lokalisierungsdaten zur Verfügung. Daher bricht die Lokalisierung bei uns nach einem Timeout mit einer Exception ab.

GPS Simulator

Um LBS-Anwendungen testen zu können, muss eine Umgebung vorhanden sein, mit der die Lokalisierungsinformation bereitgestellt werden kann. Anwendungen, die das Location API verwenden, können zum Beispiel mit dem Sun Java Wireless Toolkit (WTK) for CLDC [WTK07] getestet werden. Dieses Toolkit erlaubt die Konfiguration des Location Providers (horizontale und vertikale Genauigkeit, etc.). Die Position (Längen-, Breitengrad und Höhe) wird über die entsprechenden Felder im External Events Fenster eingegeben (siehe Abb. 1). Es ist auch möglich, eine Wegmarkendatei (XML) zu definieren und diese dann abzuspielen. Ein anderer Simulator, der die Eingabe der Position über Karten erlaubt, ist in [Pars05] vorgeschlagen worden. Beide Simulatoren können jedoch nur mit J2ME-Emulatoren verwendet werden.

Um eine LBS-Anwendung auf einem echten Gerät zu testen kann ein GPS-Simulator helfen, welcher NMEA-Datensätze im Sekundentakt über Bluetooth verschickt (analog zu einem externen GPS-Empfänger). Die Firma Skylab Mobilesystems vertreibt einen solchen GPS-Simulator [Skylab].

Um die Implementierung des Location API auf dem Nokia N71 einfach testen zu können, haben wir einen eigenen GPS-Simulator entwickelt, bei dem die generierten NMEA Datensätze genau kontrolliert werden können [GPS-SIM]. Wir haben uns beim Design der Benutzerschnittstelle (siehe Abb. 2) am Simulator des Wireless Toolkits von Sun orientiert. Neben der Eingabe der Standortdaten

Abb. 1: Location Simulator im Java Wireless Toolkit 2.5

Abb. 2: GPS Simulator

kann auch eine Wegmarkendatei aus dem WTK Simulator abgespielt werden.

Bluetooth

Unser GPS-Simulator verschickt die NMEA-Datensätze direkt über die Bluetooth-Schnittstelle. Aus dem Java-Programm greifen wir über das API JSR 82 [JSR082] auf die Bluetooth-Schnittstelle zu. Für die Implementierung dieser Schnittstelle muss auf Bibliotheken von Drittherstellern ausgewichen werden. Unter Windows hat sich die unter der GP-Lizenz stehende Implementierung Blue Cove [BlueCove] als zuverlässig erwiesen. Für eine Implementierung unter GNU/Linux muss schon einiges mehr an Suchaufwand betrieben werden. Schlussendlich zeigte sich die freie Version der Avetana GmbH [Avetana] als ausgereift. Der Simulator läuft damit unter Windows und unter Linux. Beim Start müssen lediglich unterschiedliche Bibliotheken auf dem Klassenpfad liegen.

Jedes Bluetooth-Gerät wird durch eine eindeutige Bluetooth-Adresse identifiziert. Zudem beschreibt eine 24-Bit-Geräteklasse die Art des Bluetooth-Gerätes sowie die von diesem Gerät angebotenen Dienste. Ein GPS-Sender gibt üblicherweise an, dass er einen Lokalisierungs-Dienst unterstützt. Unter Linux kann die Geräteklasse einfach definiert werden. Glücklicherweise verlässt sich das Nokia N71 nicht auf diese Angaben, und so kann ein GPS-Sender auch mit einem PC simuliert werden, der einer anderen Geräteklasse angehört. Es ist lediglich nötig, dass ein serieller Dienst angeboten wird, über den das GPS-Gerät die Lokalisierungsdaten bereitstellt. Dieser Diensttyp wird durch den 128-Bit-Schlüssel 00001101-0000-1000-8000-00805f9b34fb eindeutig definiert. Daraus ergibt sich folgende URL mit der der Dienst gestartet wird: btspp://localhost:00001101000010 00800000805f9b34fb;name=GPSSimulator

Auf der Seite des PCs müssen die Optionen so eingestellt sein, dass das Bluetooth-Gerät auch gefunden wird. Beim Zugriff auf das Location API zeigt das Nokia N71 alle Bluetooth-Geräte an, welche einen seriellen Service unterstützen. Das Gerät merkt sich ein einmal ausgewähltes Gerät, d.h. um die GPS-Quelle zu wechseln, muss die Bluetooth-Verbindung zum alten Service auf dem Nokia N71 gelöscht werden.

Funktionalität

Im folgenden Abschnitt beschreiben wir die Möglichkeiten, die der Simulator momentan bietet, und wie das Nokia N71 auf unterschiedliche Einstellungen reagiert.

Im obersten Feld der Benutzerschnittstelle kann definiert werden, welche NMEA-Satztypen verschickt werden sollen und ob die einzelnen NMEA-Sätze Informationen enthalten sollen. Für das Nokia N71 gilt, dass die Position nicht mehr bestimmt werden kann, falls einer der drei Datensätze GPGGA, GPGSA und GPRMC fehlt (die Methode getLocation wirft dann eine LocationException). Falls jedoch diese drei Datensätze geliefert werden, dann genügt es, wenn die Sätze GSA und GGA oder GSA und RMC Daten enthalten. Falls nur der Satz GSA Daten enthält, dann wirft die Methode getLocation auf dem N71 (entgegen der API Spezifikation) eine IllegalArgumentException. Interessant ist, dass das N71 die Position nicht bestimmen kann, falls ein GPS-Sender nur den RMC (Recommended Minimum Sentence C) Datensatz, also nur den empfohlenen minimalen Datensatz sendet.

In den Bereichen Location und Course kann die Position (in Grad), die Höhe (in Metern), die Geschwindigkeit (in Knoten) und die Richtung (in Grad) angegeben werden. Geschwindigkeit und Richtung sind im Location-Objekt jedoch nur

dann verfügbar, wenn beim Zugriff auf den Location Provider das Kriterium Speed and Course required auf true gesetzt wird. Die Höhe hingegen ist im Location-Objekt immer verfügbar, unabhängig von der Einstellung des Kriteriums Altitude required.

Im Bereich Misc können Angaben zur Genauigkeit der Messung gemacht werden. Eingestellt werden können die Anzahl der sichtbaren Satelliten, der Status der Messung (A = Daten OK, V = Empfängerwarnung), sowie die Qualität der Messung (0 = ungültig, 1 = GPS, 2 = DGPS, 6 = geschätzt). Das Nokia N71 ignoriert jedoch diese Qualitätsangaben.

Über das Feld HDOP und VDOP kann die sich aus der Satellitenkonstellation ergebende numerische Kondition (Dilution of Precision, DOP) definiert werden. Tiefe Werte (unter 4) stehen für optimale Konstellationen. Aus den Werten HDOP und VDOP kann die horizontale und vertikale Genauig-keit der Standortbestimmung (Estimated Position Error, EPE) wie folgt berechnet werden:

EPE horizontal (68%) = HDOP * URE

EPE vertikal (68%) = VDOP * URE

Der Faktor URE (User Range Error) fasst die Fehler der GPS-Messung zusammen (Einfluss der Erdatmosphäre, Uhrenfehler im Satelliten, Empfängergeräusche, etc). Dieser Wert kann je nach Ausrüstung zwischen 3 bis 6 betragen (ein Wert unter 10 ist zu bevorzugen, das Nokia N71 verwendet den Wert 8). Es gilt, dass 68% aller Positionsbestimmungen innerhalb eines Kreises mit Radius EPE um die echte Position liegen. EPE horizontal und EPE vertikal können über die Methoden getHorizontalAccuracy bzw. getVerticalAccuracy abgefragt werden.

Zusammenfassung

In diesem Artikel haben wir erste Erfahrungen mit einem Gerät beschrieben, welches das Location API unterstützt. Beim API haben wir eine Funktion vermisst, mit der man die unterstützen Lokalisierungsmethoden abfragen könnte. Ebenfalls vermissen wir die Möglichkeit, eine gegebene Implementierung durch eigene Lokalisierungsmethoden zu ergänzen.

Um jene Positionierungsmethode des Nokia N71, die auf einem externen GPS-Gerät basiert, besser testen zu können, haben wir einen GPS-Simulator entwickelt. Das Design des Simulators erlaubt es, einfach Erweiterungen (wie zusätzliche NMEA-Datensätze) vorzu-nehmen.

Abgesehen vom Testen von JSR 179 Implementierungen kann der GPS-Simulator auch verwendet werden, um Anwendungen, die auf Lokalisierungsdaten zugreifen, in-house zu demonstrieren. Mit unserem GPS-Simulator kann unter anderem die Beispielapplikation CityGuide aus dem WTK direkt auf einem mobilen Java-Gerät ausgeführt werden (das Projekt enthält eine Wegpunkte-Datei). Wir haben den Simulator auch bei der Entwicklung der MIDP-Version des HikeTrackers [HIKE] verwendet, um die Applikation auf verschiedenen Endgeräten zu testen.

Um JSR-179 basierte Applikationen auch auf Geräten ausführen zu können, welche das Location API nicht unterstützen, haben wir eine eigene JSR 179 Implementierung entwickelt, welche über Bluetooth auf einen GPS-Empfänger zugreift. Leider ist es aus Sicherheitsgründen nicht möglich, eigene Klassen im Paket javax.microedition.location abzulegen. Die Klassen der eigenen Implementierung haben damit eigene Namen und der Quellcode des Midlets, das auf dieses API zugreift, muss modifiziert werden.

Referenzen

[Pars05] D. Parsons, Implementing a map based simulator for the location API for J2ME, Res. Lett. Inf. Math. Sci., 2005, Vol. 7, pp 157-170, http://iims.massey.ac.nz/research/letters

[Loy07] Kimmo Löytänä, private communications, 2007.

[Nok07] Location API for J2ME (JSR-179): Implementation Notes, Version 1.2, Feb 2007, http://www.forum.nokia.com/document/Java_ME_Developers_Library/GUID-690419B7-DF05-4C10-A76A-6D75654A63F8.pdf

[JSR179] Location API for J2ME, Version 1.0.1, Java Community Process, March 2006, http://jcp.org/en/jsr/detail?id=179

[JSR248] Mobile Service Architecture, Java Community Process, Final Release Dec 2006, http://jcp.org/en/jsr/detail?id=248

[JSR082] Java Bluetooth API, Final Release Jun 2006, http://www.jcp.org/en/jsr/detail?id=82

[SDN07] The JavaME Device Table, (Filter auf Location API setzen), Sun Developer Network, http://developers.sun.com/techtopics/mobility/device/device

[POL07] Devices supporting the Location API, J2ME Polish, http://www.j2mepolish.org/devices/devices-locationapi.html

[NMEA] NMEA 0183 Standard, http://www.nmea.org/pub/0183/

[BlueCove] Freie JSR-82 Implementierung für die Windows Plattform http://sourceforge.net/projects/bluecove

[Avetana] Freie JSR-82 Implementierung, http://www.avetana-gmbh.de/avetana-gmbh/produkte/faq.xml

[WTK07] Sun Java Wireless Toolkit for CLDC, Version 2.5, Januar 2007, http://java.sun.com/products/sjwtoolkit/

[Skylab] Skylab GPS Simulator, http://www.skylab-mobilesystems.com/en/products/gps_sim.html

[3GPP] 3GPP TS 23.271, "Technical Specification Group Services and System Aspects; Functional Stage 2 Description of Location Services (LCS)", Version 7.0.0, März 2005.

[GPS-SIM] GPS Simulator Project Home, http://sourceforge.net/projects/gps-simulator

[HIKE] HikeTracker GPS Project Home, http://sourceforge.net/projects/hiketracker

Kapselung verschiedener Middleware-Technologien

Eine flexible und zugleich benutzerfreundliche, vernetzte Zusammenarbeit ist in der Software-Entwicklung unabdingbar geworden. Zu diesem Zweck haben wir eine schlanke, dienstanbietende Schnittstelle entwickelt, die die drei Middleware-Technologien Cajo, Java Message Service und JXTA kapselt. Als Ergebnis steht eine technologieunabhängige Schnittstelle zur Verfügung, welche den Datenaustausch für verschiedene Anwendungen über eine situationsgerechte Middleware ermöglicht. Drei solche Anwendungen, die einen Grossteil der erforderlichen Kommunikationsmöglichkeiten für gemeinsame Software-Entwicklung abdecken, haben wir ausgearbeitet: Reine, textbasierte Kommunikation (Chat), Austausch von Dateien (File Sharing) und Verteilung von Java-Objekten (Messaging).

Franco Ehrat, Peter Gysel, Joel Muller | peter.gysel@fhnw.ch

In einer Zeit, in der Mobilität und Kommunikation gross geschrieben werden, müssen auch in der Software-Erstellung Kommunikationsbedürfnisse beachtet werden. Entwickler müssen unabhängig vom aktuellen Standort kollaborativ neue Produkte erstellen und testen können. In Projektgruppen soll neben der reinen Entwicklung auch der Ergebnis- und Erfahrungsaustausch auf einem schnellen und einfachen Weg stattfinden können.

Die äusseren Bedingungen, unter welchen kommuniziert wird, können sich stark verändern. Wenn die Kommunikationspartner sich innerhalb des gleichen Netzes befinden, so haben sie quasi unbeschränkte Bandbreite zur Verfügung. Befinden sie sich jedoch in getrennten lokalen Netzen, dann stehen oft zwei Firewalls zwischen ihnen und es wird möglicherweise zweimal eine Network Address Translation (NAT) durchgeführt. In einem dritten Szenario befindet sich der eine Partner unterwegs und ist somit nur über mobile Kommunikation erreichbar. Benötigt wird ein Kommunikationswerkzeug, mit dem sowohl Entwickler untereinander, als auch Entwickler mit Kunden auf einfache Art und Weise über die situationsgerechte Middleware Daten austauschen können. Dieses Bedürfnis war die Motivation für die Software-Firma Giniality AG in Basel, um mit der Fachhochschule Nordwestschweiz ein solches Kommunikations-Framework mit besonderer Berücksichtigung der Peer-to-Peer (P2P) Kommunikation zu entwickeln.

Ziele

Mit einem Prototyp wollen wir demonstrieren, wie ganz verschiedene Kommunikationstechnologien gekapselt und über eine einzige, zentrale Schnittstelle, das Service Providing Interface (SPI), angesprochen werden können (Abb. 1). Das SPI soll die nötigen Funktionalitäten für die folgenden drei Anwendungen beinhalten: textbasierte Kommunikation (Chat), Austausch von Dateien (File Sharing, Server-Seite und Client-Seite) und Verteilen von Java-Objekten (Messaging). Unabhängig von der verwendeten Middleware-Technologie werden die Kommunikationsmöglichkeiten zentral im SPI definiert und erst in den darunter liegenden Technologie-Adaptern konkret implementiert. Die Technologie-Adapter verwenden einerseits ihre eigenen Technologie-Frameworks und erweitern bzw. implementieren anderseits das SPI.

Für die folgenden drei Middleware-Technologien stellen wir exemplarisch Technologie-Adapter zur Verfügung: Virtual-Virtual Machine (Cajo, beruhend auf Java Remote Method Invocation, RMI), Message Oriented Middleware (Java Message Service, JMS) und Peer-to-Peer (JXTA).

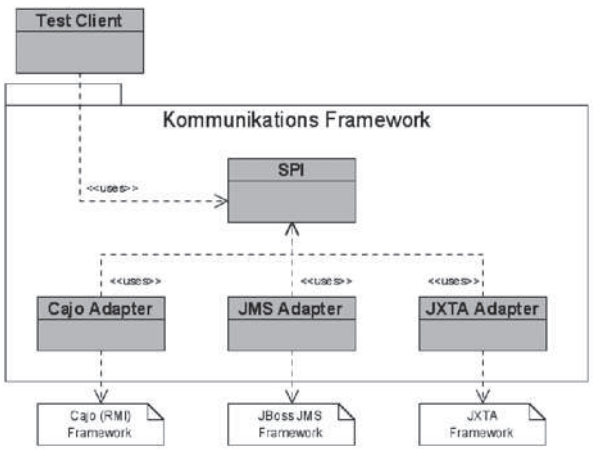


Abb. 1: Kommunikations-Framework: Verschiedene Middleware-Technologien werden über eine zentrale, technologieun-

Mit dem zuvor skizzierten Entwicklungsmuster soll dem Framework ein hoher Grad an Erweiterbarkeit gegeben werden. Weitere Adapter mit neuen Kommunikationstechnologien können separat entwickelt und hinzugefügt werden. Dabei muss das SPI nicht geändert werden, da es die Technologien und die dazugehörigen Adapter nicht kennt. Dies ist von grossem Nutzen für Applikationsentwickler, da sie lediglich wenige Klassen bzw. Interfaces kennen müssen. Zudem können über ein schlankes Application Programming Interface (API) auf die zur Verfügung gestellten Funktionalitäten zugreifen.

Die Tauglichkeit des Frameworks überprüfen wir mit einem Test-Client. Dieser Test-Client bietet die Möglichkeit, die verschiedenen Anwendungen durchzuführen und graphisch darzustellen. Auf die dabei verwendete Architektur des SPIs, die drei benutzten Middleware-Technologien und den Test-Client selber gehen wir weiter unten noch genauer ein.

Skype, MSN und ECF

Bestens bekannt sind Kommunikationsanwendungen wie Skype oder MSN. Sie bieten viele angenehme Funktionen, beruhen aber auf der klassischen Client-Server-Architektur des Internets. Das heisst, die Kommunikation nutzt vorwiegend das Hypertext Transfer Protocol (HTTP) und erfolgt über einen Server eines Dritten. Zudem müssen meistens beide Kommunikationspartner zur selben Zeit online sein.

Das sich in Entwicklung befindende Eclipse Communication Framework (ECF) [ECF06] kommt unseren Bedürfnissen näher. Dieses bietet ebenfalls eine Reihe von Kommunikationskanälen an, die direkt in Eclipse eingebunden werden können. So ermöglicht es beispielsweise der IRC-Provider (Internet Relay Chat) von ECF, Chat-Räume von Open-Source-Projekten direkt in der Eclipse- Umgebung zu betreten. Zudem können in ECF eigenständige Anwendungen auf der Rich-Client-Platform (RCP) entwickelt werden. Die Einschränkung für das ECF besteht darin, dass nur aus Eclipse heraus kommuniziert werden kann.

Cajo

Cajo [Caj06] ermöglicht eine Kommunikation zwischen mehreren virtuellen Maschinen und vereinfacht den Gebrauch von Java RMI. Es ist eine kleine (< 40 KByte), freie Java-Klassen-Bibliothek. Sie kann auf mobilen Geräten mittels der Java Micro Edition (JME) integriert werden.

Beim klassischen Java Remote Method Invocation (RMI) müssen die vom entfernten Objekt angebotenen Methoden zu Kompilationszeit in einem Interface definiert werden. Soll eine neue Methode angeboten werden, so muss das Interface neu kompiliert und verteilt werden. Interessant an Cajo ist, dass es keine zur Kompilationszeit explizit definierten Interfaces mehr benötigt. Objekte unbekannter Klassen können dynamisch zu Laufzeit mittels Java Reflection instanziert werden. Cajo stellt deshalb keine neuen Bedingungen an die Struktur von Anwendungen. Bestehende Software-Produkte können unverändert übernommen werden. Damit werden einfache P2P-Kommunikationsszenarien ermöglicht. Es sei jedoch darauf hingewiesen, dass das Auffinden der IP-Adresse von Peers vielleicht noch verbessert werden kann.

Java Message Service

Das API der Java Message Services (JMS) [JMS07] ermöglicht den Zugriff von Java-Programmen auf Nachrichtensysteme von Unternehmen. Es stellt eine Message Oriented Middleware (MOM) dar und bietet zwei verschiedene Kommunikationsarten: publish/subscribe oder Punkt-zu-Punkt-Messaging. Bei der ersten Variante wird die Kommunikation mit so genannten Topics gehandhabt. Man abonniert ein bestimmtes Topic und erhält alles, was von den andern an dieses Topic gesendet wird. Die zweite Variante arbeitet mit Queues. Eine Queue wird nur von zwei Punkten im System gleichzeitig benutzt. Ein User sendet eine Meldung an die Queue und der andere empfängt sie.

Die JMS-Schnittstelle benutzt die klassische Client-Server-Architektur und funktioniert asynchron. Dies ist zugleich Vorteil und Einschränkung dieser Technologie. Wenn eine Nachricht an ein Topic oder eine Queue gesendet wird, dann wird sie dort gespeichert und weitergesendet. Ist ein Benutzer online, so erhält er die Nachrichten sofort. Andernfalls wird sie ihm zugestellt, sobald er wieder online ist.

Das JMS-API wurde im Dokument [JMS02] standardisiert. Es gibt dazu sowohl offene als auch kommerzielle Implementierungen. In unserem Prototyp haben wir uns für das Open-Source-Produkt JBoss Application Server [JBoss06] entschieden, da wir die ganze J2EE-Umgebung aus einer Hand wollten.

JXTA

Der Name JXTA hat seinen Ursprung im englischen Wort juxtapose, was mit „nebeneinander stellen“ übersetzt werden kann. Die P2P-Architektur und die herkömmliche Server-Client-Architektur sollen nebeneinander existieren. JXTA dient der Standardisierung von P2P-Anwendungen durch offene Protokolle [JXTA07]. Dabei werden drei ehrgeizige Ziele verfolgt: Die Interoperabilität über verschiedene P2P-Systeme hinweg, die Plattformunabhängigkeit und die universelle Verbreitung, d.h. jedes Gerät kann auch Server sein.

Einige Möglichkeiten die JXTA anbietet, sind das Finden von Peers und Ressourcen, auch über Firewalls hinweg, der Austausch von Dateien mit jedermann, die Bildung eigener Gruppen von Peers

über verschiedene Netze hinweg und die sichere Kommunikation zwischen Peers über öffentliche Netze.

Service Providing Interface

Das Service Providing Interface (SPI) ist das zentrale Element unseres Kommunikations-Frameworks. Die Herausforderung besteht darin, das Interface einerseits so schlank wie möglich zu definieren und andererseits alle nötigen Anwendungsfälle („use cases") für die drei vordefinierten Anwendungen abzudecken. Zudem soll es auch möglich sein, mit allen Technologie-Adaptern das Interface, so wie es definiert ist, zu implementieren.

In Abb. 2 erklären wir die Architektur unseres SPIs am Beispiel der Anwendung „File Sharing" für den Technologie-Adapter Cajo. Das Interface FileServerModel, bietet folgende Anwendungsfälle an: Auflistung, welche Dateien auf dem Server freigegeben sind (getAllSharedFiles), Dateien freigeben (setSharedFiles) und den Server löschen (deleteServer).

Das Interface FileSharingModel erlaubt es, auf einem entfernten Server abzufragen, welche Dateien freigegeben sind (getRemoteFileList), eine Datei herunter zu laden (downloadFileFromServer) oder einen entfernten Benutzer für einen „Push" anzufragen (requestFileTransfer). Auf diese Anfrage sind zwei Reaktionen möglich: Herunterladen der betreffenden Datei (downloadFile) oder eine Ablehnung (notAcceptFileTransfer).

Das SPI besteht aus dem Singleton ModelFactory, sowie für jede Anwendung aus einem Interface und einer abstrakten Klasse (z.B. FileServerModel und AbstractFileServerModel). Die Anwendung File-Sharing beinhaltet eigentlich zwei Fälle: Die Serverseite (FileServerModel) und die Clientseite (FileSharingModel). Deshalb sind in Abb. 2 zwei Interfaces und zwei abstrakte Klassen dargestellt.

Der Test-Client wählt über die Methoden der ModelFactory die Anwendung aus, z.B. „File Sharing" über die Methode createFileSharingModel. Die gewünschte Middleware-Technologie wird beim Aufruf als String mitgegeben. In einem Property-File wird nachgeschaut, welche Implementierungsklasse (z.B. BCCajoFileSharinModel) entsprechend dem mitgegebenen Technologie-String über „ClassLoading" geladen werden soll.

In der gleichen Abbildung ist ersichtlich, dass die technologiespezifischen Klassen (z.B. BCCajoFileSharinModel) der Anwendung (in unserem Fall der Test-Client) nicht bekannt sind. Dies wird durch Anwendung des Observer Patterns [Gam95] erreicht. Eine abstrakte Controller-Klasse im Test-Client (z.B. AbstractFileSharingController) implementiert das Interface Observer. Nachdem das Model eines Technologie-Adapters (z.B. BCCajoFileSharingModel) erzeugt worden ist, registriert sich der Controller beim abstrakten Model (z.B. AbstractFileSharingModel). Der Test-Client benützt also nur die Klassen des SPIs und nicht diejenigen der Technologie-Adapter.

Der Test-Client

Um die Tauglichkeit des SPIs zu verifizieren, haben wir einen einfachen Test-Client entwickelt. Am Beispiel „File-Sharing" soll gezeigt werden, dass das SPI die für diese Anwendung benötigten use cases abdeckt.

Beim Starten hat der Benutzer die Möglichkeit, die Anwendung und die Middleware-Technologie zu wählen. Für die Anwendung „File Sharing" erscheint das in Abb. 3 dargestellte Fenster.

Der obere Teil („File Server editing") steht für die vom Benutzer selbst verwalteten Datei-Server. Die Anwendung ermöglicht es, einen neuen Server anzulegen, die freizugebenden Dateien zu verwalten oder einen bestehenden Datei-Server zu löschen. Die erstellten Server werden links in einer Liste angezeigt.

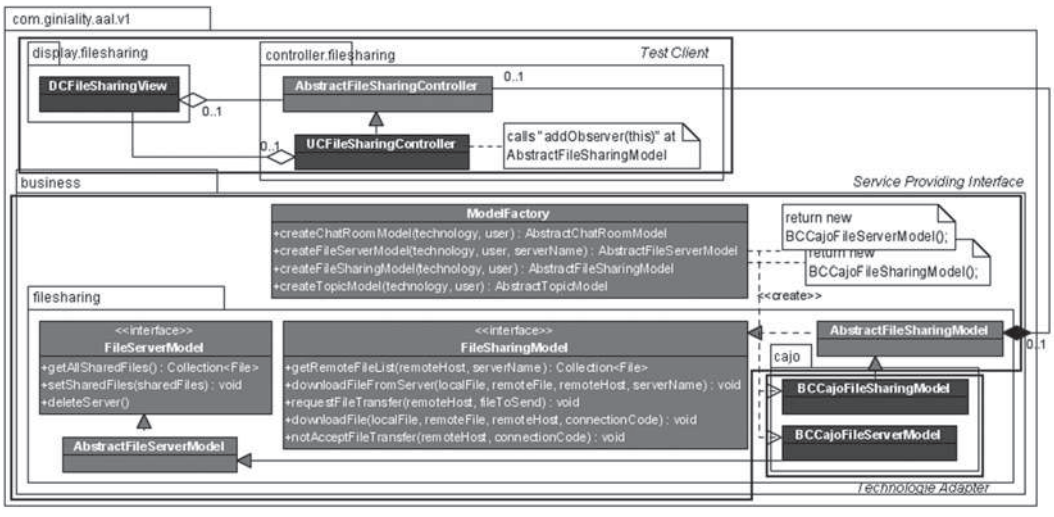


Abb. 2: Das Framework mit den drei Teilen Test-Client, Service Providing Interface (SPI) und den Technologie-Adaptern. Das SPI ist mit den Methoden für die Anwendung File-Sharing dargestellt.

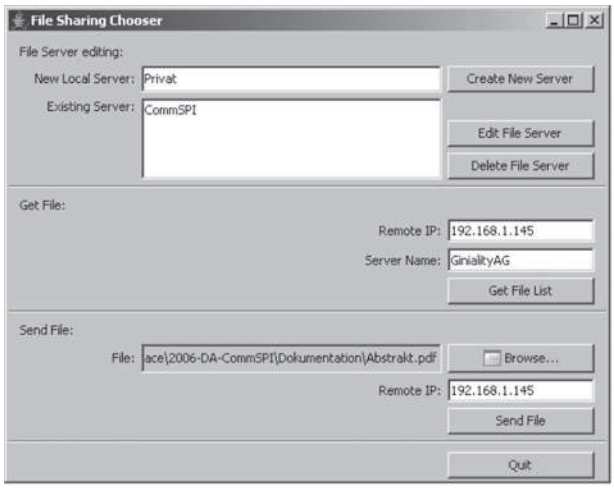


Abb. 3: Dialog-Fenster für die Anwendung „File Sharing“

Im mittleren Teil („Get File“) kommt die bewusst gewählte P2P-Architektur des Frameworks zum Ausdruck. Um von einem entfernten Benutzer freigegebene Dateien zu laden, muss man dessen IP-Adresse und den Namen des Servers kennen. Sind diese Informationen vorhanden, so kann man sich die Liste der freigegebenen Dateien anzeigen lassen und dann direkt einzelne Dateien herunter laden.

Im unteren Teil („Send File“) wird die Möglichkeit angeboten, einem entfernten Benutzer eine ausgewählte Datei zu senden („Push“). Dies geschieht jedoch nicht ohne Sicherheitsvorkehrungen: Die Datei wird nicht ohne Erlaubnis des Gegenübers gesendet und auf dessen Festplatte gelegt. Drückt der Sender den Knopf „Send File“, so wird zuerst eine Anfrage an den Empfänger gesendet, ob er die betreffende Datei von diesem Sender akzeptieren möchte oder nicht. Akzeptiert der Empfänger, so kann er zuerst den Speicherort festlegen und anschliessend wird die Datei übermittelt. Nach Abschluss der Datenübertragung wird der Empfänger informiert. Damit zwischen Anfrage und Datentransfer nichts manipuliert werden kann, erhält die Anfrage eine eindeutige Identität, die der Bestätigung und dem eigentlichen Transfer beigelegt wird. Stimmt diese Identität nicht überall überein, so wird die Übermittlung abgebrochen.

Alle hier beschriebenen use cases finden in den Interfaces FileServerModel und FileSharingModel des SPIs (siehe Abb. 2) eine entsprechende Methode.

Schlussfolgerungen und Ausblick

Mit unserem Test-Client zeigen wir, dass es möglich ist, ganz verschiedene Middleware-Technologien zu kapseln und vor dem Anwendungsentwickler zu verbergen.

Bei der Implementierung der Technologie-Adapter hat sich gezeigt, dass es schwierig ist, technologie-unabhängige Schnittstellen zu definieren. Cajo und JXTA sind synchrone Technologien, während JMS grundsätzlich asynchron funktioniert. Bei einem synchronen Aufruf kommt das Ergebnis der entfernten Aktion direkt als Rückgabewert der Methode zurück. Bei JMS muss auf das Ergebnis der entfernten Aktion gewartet werden. Dies stellt hohe Anforderungen an die Definition der Interfaces.

Als mögliche Weiterführung der Arbeiten soll geprüft werden, ob es sinnvoll und möglich ist, dass die Anwendung selber die unter gegebenen Umständen geeignetste Middleware-Technologie ermittelt. Ferner müssen für die P2P-Architektur Mechanismen entwickelt werden, die es dem Benutzer erlauben, auf einfache Art die IP-Adresse des gewünschten Kommunikationspartners zu ermitteln. Schliesslich wäre es wünschenswert, beim JMS-Adapter eine schlankere Lösung als JBoss, z.B. ActiveMQ [AMQ06], zu verwenden

Verdankung

Die Autoren bedanken sich bei den Herren Neudeck und Sauer der Firma Giniality für die ausgezeichnete fachliche Beratung während des Projektes.

Referenzen

[AMQ06] ActiveMQ, http://activemq.apache.org/

[Caj06] The cajo Project, https://cajo.dev.java.net

[ECF06] Eclipse Communication Framework, http://www.eclipse.org/ecf/

[Gam95] Erich Gamma et al., „Design Patterns: Elements of Reusable Object-Oriented Software“, Addison-Wesley, 1995.

[Gra02] J. Gradecki, „Mastering JXTA - Building Java Peer-to-Peer Applications“, Wiley Publishing, 2002.

[JBoss06] JBoss Application Server, http://labs.jboss.com/portal/jbossas

[JMS02] Java Message Service Specification, JMS API Version 1.1, April 2002, http://java.sun.com/products/jms/docs.html

[JMS07] Java message Service, http://java.sun.com/products/jms/index.jsp

[JXTA07] JXTA, http://www.jxta.org/

Replikation und Synchronisation in Oracle Database Lite

Durch die fortdauernde Verbesserung der Rechner- und Speicherleistung mobiler Geräte steigen die Ansprüche an die zu verarbeitenden Informationen. Auch mit mobilen Geräten sollte der Zugriff auf Unternehmensdaten jederzeit, überall, effizient und kostengünstig möglich sein. In diesem Umfeld kommt dem Replikations- und Synchronisationsmodell eine grosse Bedeutung zu. In dieser Arbeit wird das Oracle Database Lite Produkt bezüglich seinem Replikations- und Synchronisationsverfahren beschrieben. In Testszenarien werden Synchronisationskonflikte provoziert und das Verhalten aus Sicht der Mobile Clients beschrieben.

Matthias Hausherr, Titus Jakob, Olivier Rode, Bernhard Wyss | bernhard.wyss@fhnw.ch

Durch die fortdauernde Verbesserung der Rechner- und Speicherleistung mobiler Geräte steigen die Ansprüche an die zu verarbeitenden Informationen. Anwendungen, die bisher auf Desktop-Geräten mit ständiger Verbindung zu den Datenbank-Servern genutzt wurden, werden vermehrt auf mobilen Geräten eingesetzt. Der Zugriff auf Unternehmensdaten sollte, ungeachtet der Natur mobiler Geräte wie eingeschränkte Ressourcen, häufiger und längerer Disconnected Modus, Kommunikationskosten usw. jederzeit, überall, effizient und kostengünstig möglich sein. Durch Replikation wohldefinierter Ausschnitte des zentralen Datenbestandes auf die mobilen Geräte kann dieser Forderung gerecht werden. Trotzdem sollen sich mobile Anwendungen so verhalten, als ob stets auf dem zentralen Datenbestand operiert wird. Im mobilen Umfeld kommt deshalb dem Replikations- und Synchronisationsmodell eine grosse Bedeutung zu [DH00]. Dabei kann sich der Entwickler dem Zielkonflikt der Replikation nicht entziehen [MS04]: Die Forderungen nach Verfügbarkeit, Konsistenz der Daten und Performanz sind nicht gleichzeitig zu erreichen. Wo in diesem Spannungsfeld der drei Forderungen für die jeweilige mobile Anwendung das Optimum gelegt wird, gehört zum sorgfältigen Replikationsdesign und ist abhängig vom konkreten Replikations- und Synchronisationsverfahren des eingesetzten Systems.

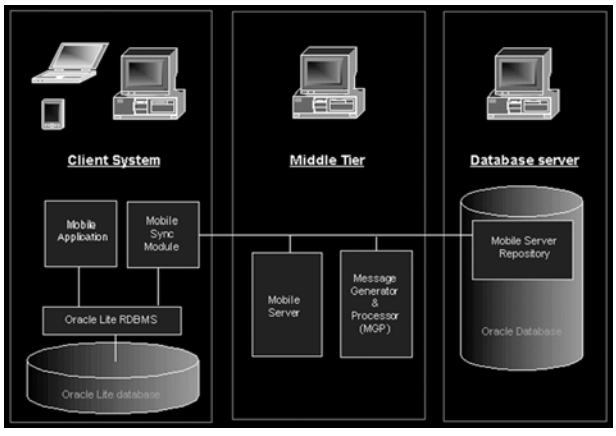


Abb. 1: Oracle Database Lite Architektur [Ora05a]

Für eine hohe Datenverfügbarkeit müssen viele Replikate erstellt werden, die im Disconnected Modus der mobilen Geräte unabhängig voneinander manipuliert werden können. Beim späteren Synchronisationsprozess können so Konflikte auftreten. In dieser Arbeit wird das Verhalten von Oracle Database Lite in Bezug auf die Konfliktbehandlung untersucht.

Im Folgenden wird zuerst die Architektur von Oracle Database Lite vorgestellt. Anschliessend wird auf das Replikations- und Synchronisationsverfahren eingegangen. Nach einer Übersicht über die Testkonfiguration wird anhand verschiedener Testszenarien der Synchronisationsprozess beleuchtet. Zum Schluss werden die Resultate zusammengefasst.

Oracle Database Lite Architektur

Das Oracle Database Lite Produkt ermöglicht die Entwicklung, Verteilung und Verwaltung mobiler Datenbank-Anwendungen. Für die verschiedenen Arbeiten wird eine Reihe von Werkzeugen zur Verfügung gestellt wie das Mobile Development Kit, die Mobile Database Workbench, der Application Packaging Wizard, der Mobile Workspace. Die für die Testumgebung relevanten Werkzeuge werden später dargestellt, einen Überblick gibt [Ora06].

Das System ist als dreischichtige Architektur gestaltet, bestehend aus dem Mobile Client mit dem Oracle Lite Datenbanksystem, dem Mobile Sync Modul und der mobilen Anwendung, dem

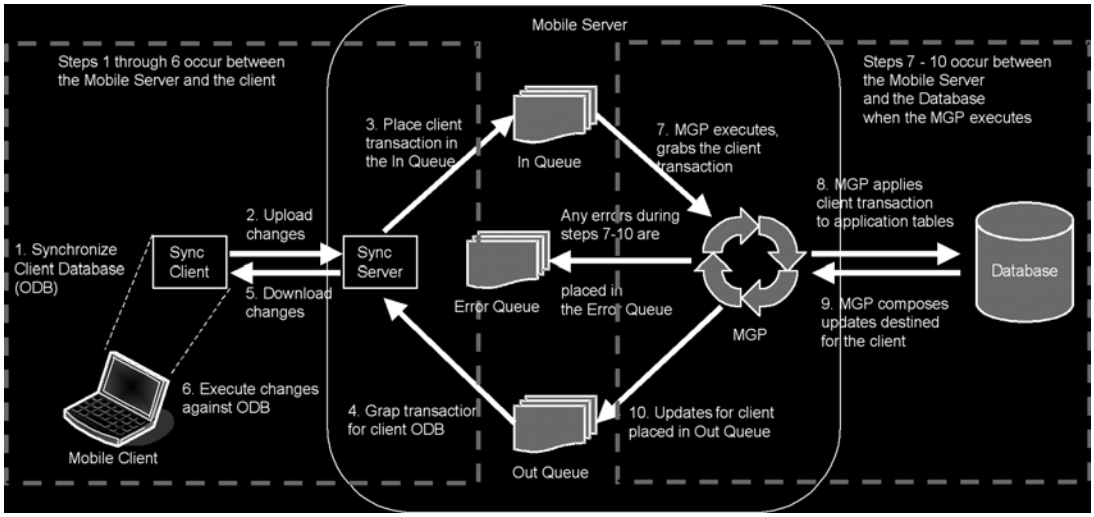


Abb. 2: Synchronisationsprozess [Ora05a]

1. Der Benutzer initiiert eine Synchronisation auf dem mobilen Gerät.
2. Das Mobile Sync Modul fasst alle Änderungen des Clients zu einer Transaktion zusammen und übergibt die Transaktion dem Sync Server im Mobile Server.
3. Der Sync Server platziert die Transaktion in die In-Queue.
4. Der Sync Server holt sich alle Transaktionen, die für den Mobile Client bestimmt sind, aus der Out-Queue.
5. Der Sync Server übergibt diese Transaktionen dem Mobile Sync Module.
6. Das Mobile Sync Module führt alle Änderungen für den Client aus.
7. Der MGP holt sich alle Transaktionen der Mobile Clients aus der In-Queue.
8. Der MGP wendet alle Transaktionen der Mobile Clients gegenüber den entsprechenden Tabellen an.
9. Änderungen, die für die Mobile Clients bestimmt sind, werden durch den MGP je Client zu einer Transaktion zusammengefasst.
10. Der MGP platziert diese Transaktionen in die Out-Queue.

Middle Tier mit dem Mobile Server und dem Message Generator & Processor (MGP) und einem Datenbankserver als Backend mit dem Mobile Server Repository (siehe Abb. 1 aus [Ora05a]).

Eine zentrale Rolle für die Synchronisation spielen der Mobile Server als Sync Server gegenüber den Mobile Clients und der MGP gegenüber dem Backend Datenbanksystem. Der Sync Server reagiert auf Synchronisationsanfragen der Client, der MGP läuft als Hintergrundprozess unabhängig vom Ersteren.

Replikations- und Synchronisationsmodell

Oracle Lite realisiert ein Publish/Subscribe Replikationsmodell mit Hilfe von Snapshots. Für die Anwendungen werden auf dem Mobile Server Publikationen erstellt, die ein oder mehrere Publikationsartikel enthalten können. Diese basieren auf Tabellen oder Views des Datenbankservers. Den Mobile Clients werden diese Publikationen anhand von Subskriptionen zugeordnet. Die Definition der Publikationsartikel erfolgt mittels SQL-Abfragen, die parametrisiert sein können, um benutzerspezifische Ausschnitte (Subsetting) zu definieren.

Mithilfe des Mobile Sync Modules können die mobilen Geräte ihre Snapshots mit den Daten auf dem zugehörigen Datenbankserver synchronisieren. Die Synchronisation zwischen den Oracle Lite Clients und dem Oracle Datenbank Server erfolgt asynchron über den Mobile Server (siehe Abb. 2 aus [Ora05a]). Das Mobile Sync Module arbeitet unabhängig vom MGP und umgekehrt.

Gemäss der Kategorisierung von Replikationsmodellen [GHN96] handelt es sich hier um eine „Lazy Group Replication“, auch bekannt unter der Bezeichnung „Lazy Update Everywhere“ [WPS00]. Änderungen können auf jedem Replikat vorgenommen werden (Group bzw. Update Everywhere), der Abgleich mit den anderen Kopien erfolgt asynchron zu einem beliebigen, späteren Zeitpunkt (Lazy).

Das Wesentliche bei diesem asynchronen Verfahren ist, dass nach den Schritten eins bis sechs im Normalfall dem Mobile Client eine erfolgreiche Synchronisation gemeldet wird, obwohl mögliche Konflikte erst später durch den MGP erkannt und entsprechend den Konfliktregeln aufgelöst werden. Als Konfliktregel können in Oracle Lite Database den Publikationsartikeln entweder „Server Wins“ oder „Client Wins“ hinterlegt werden. In [Ora05b] sind mögliche Synchronisationskonflikte aufgelistet, von denen hier die folgenden drei ausgewählt worden sind, da sie die Grundlage unserer Testszenarien bilden:

1. Der Client und der Server ändern dieselbe Zeile. Dieser Konflikt wird entsprechend den Konfliktregeln durch den Mobile Server aufgelöst.
2. Der Client löscht dieselbe Zeile, die der Server ändert. Dieser Konflikt wird entsprechend den Konfliktregeln durch den Mobile Server aufgelöst.

3. Der Server löscht dieselbe Zeile, die der Client ändert. Dieser Konflikt wird nicht vom Mobile Server aufgelöst. Ein entsprechender Fehler wird generiert und der Administrator muss entscheiden, wie dieser Konflikt aufgelöst werden soll.

Je nach Konfliktregel werden dann bei einer erneuten Synchronisation vorher erfolgreich durchgeführte Änderungen auf dem mobilen Gerät oder dem Server teilweise rückgängig gemacht. Dies geschieht ungeachtet von Transaktions-, ja sogar SQL-Statement-Grenzen. Der Datenabgleich wird für jede geänderte Zeile einzeln durchgeführt. In [PB99] wird dies als Data-Centric-Ansatz im Gegensatz zu einem Transaction-Centric-Ansatz bezeichnet

Testumgebung
Für die nachfolgenden Testszenarien wurde folgende Umgebung eingerichtet:

Computer	Betriebssystem	Installation
Hades	SuSE Linux Enterprise Server 9, Version 9, Patchlevel 3	Oracle Database 10g Oracle Lite Mobile Server
Laptop 1	Windows XP SP 2	Oracle Lite Mobile Development Kit
PC 1	Windows XP SP 2	Oracle Lite Mobile Client Alf
Laptop 2	Windows XP SP 2	Oracle Lite Mobile Client Beat

Die Grundlage des Publikationsartikels und der Subskriptionen ist die Tabelle example, die wie folgt erstellt wurde:

```
CREATE TABLE example(
  Id number(4) primary key,
  Name varchar(30) not null,
  Phone varchar(12));
```

Mit dem Packaging Wizard werden die Publikationsartikel, die gleichzeitig (nicht parametrisierte) Snapshots für die Clients sind, festgelegt:

Reiter	Feld	Input
Plattformen	Ausgewählte Plattform	Oracle Lite WIN32;US
Anwendung	Anwendungsname	Mobile Field Services
	Virtueller Pfad	/MFS
	Beschreibung	Field Task Assignment
	Lokales Anwendungsverz.	<local folder>/mfsdev
Datenbank	Datenbankname	mfs
Snapshots – Neu – Importieren	Tabelle	Example
Snapshots – Neu – Server	Snapshot-Name	Example
	Gewichtung	1
	Eigentümer	DEMO
Snapshots – Neu – Oracle Lite WIN32;US	Auf Client erstellen	Check
	Aktualisierbar?	Check
	Basisobjekttyp	Tabelle
	Konfliktauflösung	Server vorrangig
	Aktualisierungstyp	Schnelle Aktualisierung
	Vorlage	SELECT * FROM DEMO.EXAMPLE
	(Restliche Eingabefelder)	(leer lassen)

Der Zustand auf der Backend-Datenbank wird vor jedem Testszenario mit den folgenden Einträgen neu hergestellt und die beiden Clients damit synchronisiert:

```
INSERT INTO example VALUES(1,
      ,Andre‘, ,111111111111‘);
INSERT INTO example VALUES(2,
      ,Bruno‘, ,222222222222‘);
INSERT INTO example VALUES(3,
      ,Carlo‘, ,333333333333‘);
```

Test Szenarien
Als Grundlage der folgenden Testszenarien betrachten wir ein typisches Anwendungsgebiet: Zur Unterstützung der täglichen Arbeit der Angestellten einer Handelsfirma dient eine zentrale Datenbank. Alle Informationen über Kunden, Anbieter, Bestellungen, Mitarbeiter usw. werden auf einem zentralen Server gehalten. Die Aussendienstangestellten sind mit Laptops ausgestattet. Sie fahren zu den Kunden und nehmen vor Ort deren Bestellungen auf, erfassen oder mutieren Adressen usw., dabei sind sie in der Regel offline und manipulieren einen lokalen Datenbestand. Nach Rückkehr in die Firma werden die mobil erfassten Daten auf den Server übertragen.

In den folgenden Szenarien werden zwei Clients, Alf und Beat, verwendet. Beide Clients führen unabhängig voneinander Aktionen auf ihren lokalen Datenbanken durch und synchronisieren danach in einer bestimmten Reihenfolge. Bei diesen Tests wird kein Data-Subsetting verwendet, weil sich die einzelnen Clients ansonsten gar nie in die Quere kommen können, da sie von vornherein ihre eigenen, nur für sie benutzbaren Bereiche der Tabellen nutzen. Ohne Data-Subsetting können alle Clients auf den gesamten Inhalt der Basistabelle zugreifen, was zwangsläufig zu Konflikten führt.

Es werden keine Manipulationen direkt auf der Backend Datenbank ausgeführt. Die Änderungen des Clients, der zuerst synchronisiert, werden so auf dem Server angewendet, bei der späteren Synchronisation des anderen Clients können Konflikte auftreten.

Bei der Synchronisation werden alle bei einem Client auf der lokalen Datenbank durchgeführten Transaktionen bzw. Änderungen in die In-Queue des Mobile Servers gelegt, wo sie dann auf der Backend Datenbank ausgeführt werden. Die In-

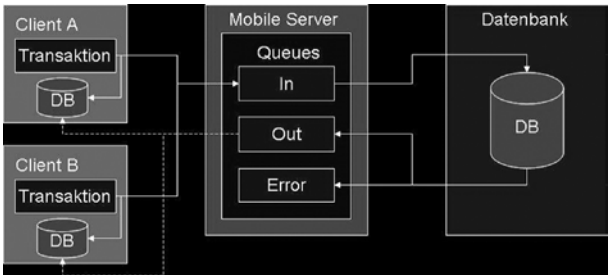


Abb. 3: Situation der Testszenarien

formationen, wo was geändert wurde, wird in die Out-Queue gelegt und bei der nächsten Synchronisation vom Client heruntergeladen, damit die lokale Datenbank mit der Backend Datenbank konsistent bleibt. Falls es auf der Backend Datenbank bei der Ausführung der durch die In-Queue erhaltenen Transaktionen zu einem Konflikt kommt, werden die inkonsistenten Aktionen nicht durchgeführt, sondern werden in die Error-Queue gelegt, wo sie durch einen Administrator bearbeitet bzw. gelöst werden sollten (siehe Abb. 3).

In den Testfällen werden die Updates U1.1, U2, U1.2, ein Insert I4 und ein Delete X1 verwendet. Die dafür verwendeten Statements sehen wie folgt aus:

```
U1.1:  UPDATE example SET
       Phone='updated1' WHERE Id=1;
U1.2:  UPDATE example SET
       Phone='updated3' WHERE Id=1;
U2:    UPDATE example SET
       Phone='updated2' WHERE Id=2;
I4:    INSERT INTO example
       VALUES(4, ,Isert',
       ,444444444444');
X1:    DELETE FROM
       example WHERE Id=1;
```

Testszenario 1

Ausgangslage: Alf macht einen Update U1.1 auf seiner lokalen Tabelle „example“ in seiner lokalen Datenbank. Beat ändert denselben Record mit Update U1.2 auf seiner eigenen lokalen Tabelle. Zusätzlich macht er einen Update U2 auf einem anderen Record. Nun synchronisiert Alf vor Beat mit dem Mobile Server.

Alf synchronisiert erfolgreich und seine lokalen Änderungen werden erfolgreich in die Backend Datenbank übernommen und danach für die beiden Clients in der Out-Queue bereit gelegt.

Out-Queue

Alf	Update U1.1
Beat	Update U1.1

Error-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Die nachfolgende Synchronisation mit Beat ergibt ebenfalls die Meldung „Synchronisation erfolgreich“. Der Update U1.2 durch Beat landet aber in der Error-Queue, da der betroffene Record bereits durch den Update U1.1 von Alf geändert wurde. Der Update U2 von Beat wird in die Backend Datenbank übernommen und auch in die Out-Queue gelegt.

Out-Queue

Alf	Update U1.1 Update U2
Beat	Update U1.1 Update U2

Error-Queue

Alf	Keine Einträge
Beat	Update U1.2

Resultat: Der MGP erkennt einen Konflikt, den er auflösen kann (Konflikt 1, siehe Replikations- und Synchronisationsmodell), und markiert ihn als „CONFLICT DEDECTED“. Nur die in direktem Konflikt stehenden Aktionen gelangen in die Error-Queue. Obwohl der Client Beat die beiden Updates als eine Transaktion ausgeführt hat, gilt die „Alles oder Nichts“-Regel bei der Synchronisation nicht mehr, die Wirkung von U2 bleibt erhalten, die Wirkung von U1.2 wird zurückgesetzt.

Testszenario 2

Ausgangslage: Alf macht einen Update U1.1 auf seiner lokalen Tabelle „example“ in seiner lokalen Datenbank. Beat löscht denselben Record mit dem Delete X1 auf seiner eigenen lokalen Tabelle. Zusätzlich macht er einen Update U2 auf einem anderen Record und erstellt einen neuen Record mit dem Statement I4 in derselben Tabelle. Nun synchronisiert Alf vor Beat mit dem Mobile Server.

Alf synchronisiert erfolgreich und seine lokalen Änderungen werden erfolgreich in die Backend Datenbank übernommen und danach für die beiden Clients in der Out-Queue bereit gelegt.

Out-Queue

Alf	Update U1.1
Beat	Update U1.1

Error-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Die nachfolgende Synchronisation mit Beat ergibt auch die Meldung „Synchronisation erfolgreich“. Der Delete X1 durch Beat landet aber in der Error-Queue, da der betroffene Record bereits durch einen Update U1.1 von Alf geändert wurde. Die weiteren Mutationen U2 und I4 von Beat werden in die Backend Datenbank übernommen und auch in die Out-Queue gelegt.

Out-Queue

Alf	Update U1.1 Update U2 Insert I4
Beat	Update U1.1 Update U2 Insert I4

Error-Queue

Alf	Keine Einträge
Beat	Delete X1

Resultat: Der MGP erkennt einen Konflikt, den er auflösen kann (Konflikt 2, siehe Replikations- und Synchronisationsmodell), und markiert ihn als „CONFLICT DEDECTED“. Nur die in direktem Konflikt stehenden Aktionen gelangen in die Error-Queue. Obwohl der Client Beat sein Delete, Update und Insert als eine Transaktion ausgeführt hat, gilt die „Alles oder Nichts“-Regel bei der Synchronisation nicht mehr, die Wirkung von U2 und I4 bleibt erhalten, die Wirkung von X1 wird zurückgesetzt.

Testszenario 3

Ausgangslage: Alf löscht einen Record mit dem Statement X1 auf seiner lokalen Tabelle „example“. Gleichzeitig ändert Beat auf seiner lokalen Tabelle „example“ denselben Record mit einem Update U1.1, fügt einen weiteren Record mit Insert I4 der Tabelle hinzu und macht einen weiteren Update U2 auf einer weiteren Zeile in der Tabelle. Nun synchronisiert Alf vor Beat.

Nach der Synchronisation von Alf mit dem Mobile Server erscheint die Meldung „Synchronisation erfolgreich“ und die durch Alf vollzogene Änderung wird zuerst in die In-Queue des Mobile Servers übernommen und dann auf die Backend Datenbank appliziert. Danach wird die Änderung für die einzelnen Clients in die Out-Queue des Mobile Servers gelegt. Fehler sind dabei keine aufgetreten.

Out-Queue

Alf	Keine Einträge
Beat	Delete X1

Error-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Bei der Synchronisation mit Beat erscheint wieder eine „Synchronisation erfolgreich“ Meldung, obwohl der Update U1.1 in der Error-Queue landet, da der zu mutierende Record bereits durch den Delete X1 von Alf in der Hauptdatenbank nicht mehr existiert. Auffällig dabei ist, dass alle weiteren Aktionen von Beat in der Error-Queue landen, auch wenn sie in keinem direkten Konflikt mit den Änderungen von Alf stehen. Wenn Beat nun ein zweites Mal synchronisiert, wird seine lokale Datenbank mit der Backend Datenbank überschrieben. Die Folge ist, dass seine lokalen Änderungen verloren gehen. D. h. der Delete X1 wird auf seiner lokalen Datenbank ausgeführt und seine Änderungen U1.1 und I4 werden durch die Backend Datenbank überschrieben und gehen somit verloren.

Out-Queue

Alf	Keine Einträge
Beat	Keine Einträge

Error-Queue

Alf	Keine Einträge
Beat	Update U1.1 Update U2 Insert I4

Resultat: Der MGP erkennt einen Konflikt, den er nicht auflösen kann (Konflikt 3, siehe Replikations- und Synchronisationsmodell), und markiert ihn als „ERROR“. Alle, also auch Aktionen, die nicht direkt in Konflikt stehen, werden nicht ausgeführt, sondern landen in der Error-Queue.

Zusammenfassung

Mit einfachen Testszenarien haben wir das Verhalten von Oracle Database Lite beim Auftreten von drei Synchronisationskonflikten dargestellt. Das Verhalten wurde aus Sicht zweier Mobile Clients beschrieben, obwohl die Konflikte immer nur zwischen einem Client und dem Server auftreten. In dieser künstlich erzeugten symmetrischen Konstellation zeigt sich am stärksten die asymmetrische Natur der Konfliktauflösung. Der UP-

DATE-UPDATE-Konflikt (Testszenario1) ist eher symmetrisch, die Reihenfolge der Synchronisation spielt keine Rolle. Der Konflikt wird mit der definierten Regel (hier Server Wins) aufgelöst. Der DELETE-UPDATE-Konflikt (Testszenarien 2 und 3) ist aus Sicht der Clients asymmetrisch, je nach Reihenfolge der Synchronisation wird der Konflikt aufgelöst oder der Datenabgleich führt zu einem Fehler. Wenn Konflikte erkannt und aufgelöst werden (Testszenario 1 und 2), ist es möglich, dass die Transaktionsgrenzen der Mobile Clients verschwinden, die „Alles oder Nichts"-Regel gilt nicht mehr, das Verfahren ist Data-Centric. Durch das asynchrone Verfahren erkennen die Mobile Clients nicht, ob ihr Datenabgleich zu Konflikten geführt hat.

Dies mag aus Sicht der Mobile Clients eher nachteilig erscheinen. Man muss sich aber bewusst sein, dass ein auf Performanz, insbesondere Skalierbarkeit ausgelegtes Replikations- und Synchronisationsverfahren, wie es das Oracle Database Lite bietet, bei der Forderung nach Konsistenz bzw. der Forderung nach Datenverfügbarkeit gewisse Abstriche machen muss (Zielkonflikt). Es liegt vollständig in der Hand des Entwicklers mobiler Anwendungen, welchen der beiden Forderungen mehr Priorität eingeräumt werden soll.

Zielt die Anwendung auf möglichst hohe Datenverfügbarkeit und wird deshalb auf Subsetting verzichtet, kann es zu Situationen kommen, wie sie in den Testszenarien beschrieben sind. Die Datenkonsistenz ist dabei insofern garantiert, als dass jeder Mobile Client für sich ein konsistentes Abbild der Daten hält, das aber verschieden sein kann von anderen Clients oder des Servers.

Vorteilhafter wäre es, mit einem sorgfältigen Replikationsdesign für die mobile Anwendung von vornherein Synchronisationskonflikte zu vermeiden. Es wird kaum nötig sein, allen Mobile Clients die vollständigen Publikationsartikel zur Änderung zur Verfügung zu stellen. Mithilfe parametrisierter Snapshots können den Mobile Clients nur die für sie relevanten Ausschnitte der Publikationsartikel bereitgestellt werden (Subsetting). Je kleiner die Menge der gemeinsamen änderbaren Zeilen ist, je geringer ist die Wahrscheinlichkeit eines Synchronisationskonflikts.

Referenzen

[DH00] Dunham, M., Helal, A.: Mobile Computing and Databases: Anything New?. SIGMOD RECORDS, 4(4), pp. 311-321, 2000.

[GHN96] Gray, J., Helland, P., O'Neil, P., Shasha, D.: The Dangers of Replication and a Solution, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, pp. 173 – 182, ACM, 1996.

[MS04] Mutschler, B., Specht, G.: Mobile Datenbanksysteme, Springer, Berlin, 2004.

[Ora05a] Oracle® Database Lite Developer's Guide 10g (10.2.0) Part No. B15920-01, 2005.

[Ora05b] Oracle® Database Lite Administration and Deployment Guide 10g (10.2.0) Part No. B15921-01, 2005.

[Ora06] Oracle Database Lite 10gR2 Technical White Paper, 2006

[PB99] Phatak, S. H., Badrinath, B. R.: Conflict resolution and reconciliation in disconnected databases. Proceedings of MDDS 1999, Sep. 1999

[WPS00] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding Replication in Databases and Distributed Systems, Proceedings of 20th International Conference on Distributed Computing Systems, 2000.

Es muss nicht immer Java sein: Python für mobile Anwendungen

Nokia hat mit Python für S60 eine für mobile Geräte neue Programmiersprache veröffentlicht, welche eine interessante Alternative zur herkömmlichen Softwareentwicklung in Java oder C/C++ darstellt. Wir stellen Python für S60 kurz vor und testen den Einsatz von Python sowohl für rechenintensive als auch kommunikationsorientierten Beispiele. Dabei kommen wir zum Schluss, dass sich Python vor allem für Anwendungen ohne grosse Ansprüche an die Rechenleistung und ganz allgemein für ein Rapid Prototyping eignet.

Oliver Ruf | oliver.ruf@fhnw.ch

Die Vor- und Nachteile von plattformspezifischen Maschinenprogrammen, welche zum Beispiel durch Kompilation von C/C++-Programmen entstehen, sind hinlänglich bekannt: sehr hohe Performanz bei mangelnder Plattformunabhängigkeit. Das Problem der fehlenden Unterstützung von verschiedensten Prozessortypen und Plattformen ist bei der Software-Entwicklung in Java weitestgehend durch den Einsatz von plattformunabhängigem Bytecode gelöst. Dieser Zwischencode wird von der Java Virtual Machine (VM) interpretiert oder just in time (JIT) kompiliert. Somit reduziert sich die Plattformabhängigkeit auf die VM. Der grosse Nachteil dabei ist die verminderte Performanz in rechenintensiven Anwendungen.

Python ist im Gegensatz zu C/C++ und Java eine reine Interpretersprache. Andere bekannte Interpretersprachen sind beispielsweise Perl, Ruby, PHP und auch BASIC. Die Programme werden also nicht kompiliert, sondern direkt zur Laufzeit interpretiert. Sie benötigen daher, ähnlich zu Java, eine plattformspezifische Laufzeitumgebung, um die Programme, auch Skripte genannt, auszuführen. Alleine die Tatsache, dass Python-Programme interpretiert werden, weist darauf hin, dass mit einer verminderten Performanz zu rechnen ist. Wie gross diese Einbusse ist und welche Vorzüge durch die Verwendung eines Interpreters anstatt eines Compilers resultieren, wollen wir hier genauer untersuchen.

Erfolg durch Einfachheit

Bevor sich ein Programmierer an die Lösung eines Problems macht, wäre es wünschenswert, dass er die verschiedenen Vor- und Nachteile der unterschiedlichen Programmierparadigmen und Programmiersprachen im gegebenen Problemfeld untersucht. Denn nicht immer ist der erste Griff in die Sprachenkiste der Richtige; doch fast immer gilt, dass ein falscher Griff später sehr teuer bezahlt werden muss. Falls sich also verschiedene Programmiersprachen anbieten, weil sie technisch gut unterstützt werden und weil sich die Programmierer damit wohl fühlen, so sollten sich die Entwickler unter anderem die Frage stellen: Wie schnell und wie effizient kann ich das gegebene Problem mit der Programmiersprache P lösen? Die redliche Beantwortung dieser Frage müsste zu einem Mix an Programmiersprachen führen, abhängig davon, welches Problem es zu lösen gilt. Denn gewisse Programmiersprachen, wie C/C++, bieten ein grosses Mass an Effizienz, sind aber im Entwicklungsprozess umständlich zu handhaben. Andere Programmiersprachen, wie beispielsweise Python, punkten hier, denn der Entwicklungsprozesses hängt dabei von verschiedenen Faktoren ab, unter anderem von der Einfachheit der Sprache, den unterstützten Konzepten, den beiliegenden Bibliotheken und dem benötigten Aufwand um ein lauffähiges Programm zur Ausführung zu bringen. Ein erstes Gefühl der Einfachheit Pythons erhalten wir dadurch, dass wir die Seitenzahl der Sprachdefinitionen vergleichen. Während für Python knapp hundert Seiten ausreichen [PySpc], benötigt es für Java bereits über sechshundert Seiten um die Sprache zu definieren [JaSpc]. Kurz zusammengefasst heisst das, dass sich keine der heute verbreiteten Programmiersprachen in einem absoluten Sinn als „die Beste" behaupten kann.

Um einen ersten Eindruck von Python vermitteln zu können, haben wir in Listing 1 den Klassiker Fibonacci programmiert. Anders als in vielen anderen strukturierten Programmiersprachen werden in Python die strukturellen Blöcke lediglich durch das Einrücken von Zeilen definiert und es sind keine expliziten Klammern oder Schlüsselwörter als Begrenzung nötig. Die Art der Einrückung ist somit vorgegeben und kann nicht dem persönlichen Geschmack des Programmierers überlassen werden. Dies wird vor allem bei der for-Schleife im Beispiel deutlich sichtbar: alle

```
import time

def fibonacci(n):
    if   n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
measurings = 100
amount     = 30
totaltime  = 0.0
for i in range( measurings ):
    begin = time.clock()
    fibonacci(amount)
    end = time.clock()
    totaltime += end - begin
average = 1000 * totaltime / measurings # in ms
```

Listing 1: Fibonacci mit Zeitmessung

nachfolgenden Anweisungen, welche einfach eingerückt sind, gehören zum Schleifenkörper. Für Java- oder Pascal-Programmierer ist der Quellcode etwas gewöhnungsbedürftig aber die Lesbarkeit ist insgesamt verbessert. Es besteht dadurch aber auch die Gefahr eines fälschlichen Einrückens und somit einer fehlerhaften Interpretation.

Variablen werden generell dynamisch alloziert und der Speicher automatisch verwaltet, so dass keine explizite Instanzierung notwendig ist. Es wird auch nicht vorgeschrieben, auf welche Art die Programme geschrieben werden müssen, und Python lässt dem Entwickler dadurch die Wahl funktional, objektorientiert oder aspektorientiert zu programmieren. Es ist sogar möglich, diese Paradigmen zu mischen.

Diese einfachen Grundsätze sind nicht unwesentlich, weil sich dadurch in kürzester Zeit ein lauffähiges Programm erstellen lässt.

Wenn man also eine spontane Idee hat und nur kurz mal ausprobieren möchte, ob sich diese auch als Programm umsetzten lässt, so wünscht man sich einen möglichst einfachen Entwicklungsprozess ohne schwerfällige Entwicklungsumgebung und ohne langwierigen Kompilations- und Bindeprozess. Ist ein solcher nicht vorhanden, verwirft man die Idee zu leicht oder verschiebt sie auf später.

Die Einfachheit des Python-Entwicklungsprozesses lässt sich gut im Vergleich zu Java am Standardprogramm „Hello, World!“ zeigen. Im Java-Programm (Listing 2) benötigen wir eine in einer Klasse eingebetteten main-Prozedur mit dem entsprechenden Print-Aufruf.

Dieses Java-Programm muss in der Datei «Hello.java» (entsprechend dem Klassennamen) gespeichert und danach kompiliert werden. Die Kompilation geschieht mit dem Befehl javac Hello.java, welcher die Bytecode-Datei Hello.class erstellt. Um das Ganze dann auch auszuführen, muss man die VM mit dem Aufruf java Hello dazu anweisen. Das Resultat ist, dass „Hello, World!“ auf der Konsole ausgegeben wird. Bei Java-Applikationen für Mobiltelefone muss die Funktion zusätzlich noch

```
public class Hello
{
  public static void main(
      String[] args )
  {
    System.out.println(
      „Hello, World!“ );
  }
}
```

Listing 2: „Hello World“ mit Java

Abb. 1: „Hello, World!“ in der Python-Konsole auf einem Nokia E61

Abb. 2: Das Nokia E61

in einem MIDlet eingebaut werden, was weiteren Aufwand bedeutet.

Bei Python erfüllt lediglich ein einzelner Befehl nach Aufstarten des Interpreters seinen Dienst (siehe Abb. 1). Dieser reduzierte initiale Aufwand spielt speziell bei kleinen Applikationen ein wichtiges Kriterium, fällt aber bei grösseren Projekten nicht mehr stark ins Gewicht, da das Verhältnis zum restlichen Quellcode mit dessen Zunahme abnimmt. Daher eignet sich Python besonders gut für Rapid Prototyping.

Wie eingangs schon angesprochen, ist ein weiteres wichtiges Merkmal von erfolgreichen Programmiersprachen die Güte der zugehörigen Bibliotheken. Ähnlich wie die Standardbibliothek von Java [JaLib] besitzt auch Python eine umfangreiche Standardbibliothek [PyLib] mit vielen nützlichen Funktionen. Im Python-Jargon heissen die Bibliothekseinheiten Module und reichen von einfachen Datenstrukturen über Internet- und Netzwerk-Funktionen bis hin zu XML-Parsern und Kompressionsalgorithmen.

Python für Mobiltelefone

Im Januar 2006 veröffentlichte Nokia für die S60 Serie ihrer Mobiltelefone eine Implementierung der Programmiersprache Python in der Version 2.2 [SFS60]. Dass sich Nokia gerade für Python als Skripting-Sprache entschieden hatte, hängt damit zusammen, dass Python generell eine Schnittstelle zu C/C++-Programmen besitzt. Somit besteht die Möglichkeit, vorhandenen C++-Code in eine von Python aus aufrufbare DLL zu kompilieren und dort zu verwenden. Die Python-Implementierung von Nokia enthält einen Interpreter, einen Grossteil der Standardbibliothek und ein API für die Geräte. Nokia stellt diese Implementierung als open source unter einer Apache License V2.0 frei zur Verfügung [ApLic]. Eine aktive Community wie auch Nokia selbst arbeiten kontinuierlich an der Weiterentwicklung der Implementierung.

Als zentrale Stelle dient der Community die offizielle Webseite der open-source-Projekte von Nokia [NoWik], welche als Wiki realisiert ist und somit allen Benutzern die Möglichkeit bietet, Seiten ändern und erweitern zu können. Als Support- und Diskussionsplattform stehen ein offizielles Forum [NoFor] und ein IRC-Chat zur Verfügung.

Eines der Kernstücke von Python für S60 ist die Nokia-API. Mit viel Aufwand hat Nokia ihre C/C++-APIs nach Python exportiert. Diese C/C++-APIs bieten Zugriff auf die meisten Funktionen des Telefons. Entsprechend gibt es auch in Python für S60 diverse Module für alle möglichen Funktionen des entsprechenden mobilen Geräts: Module für GUI- und Grafik-Funktionalität, der Zugriff auf die interne Kamera, die Möglichkeit SMS-Nachrichten zu versenden, oder auch Bluetooth-Verbindungen aufzubauen sind einige Beispiele dafür.

Im Gegensatz dazu sind bei Python für S60 noch nicht alle Module der Standardbibliothek umgesetzt worden. Die Betonung liegt hier speziell auf dem „noch nicht“, denn die Entwickler von Nokia arbeiten an der Erweiterung der bisherigen Umsetzung und veröffentlichen mittlerweile fast monatlich neue Versionen.

Als konkretes Beispiel sind die XML-Funktionen der Standard-Bibliothek noch nicht auf dem Mobiltelefon verfügbar. Ein übliches import xml endet abrupt mit einem ImportError. Gerade hier zeigt sich die Community von der flexibelsten Seite. Einer der interessierten Benutzer hat zu diesem Zweck ein anderes freies XML-Modul, cElementTree, für die mobile Plattform übersetzt und bietet dieses zum herunterladen auf seiner Webseite an [PyXml]. Wie dieses Beispiel zeigt, füllen die Benutzer die noch vorhandenen Lücken durch eigene Module und Programme. Dennoch bedeutet dies für den Programmierer immer zusätzlichen Aufwand; vor der Verwendung eines Moduls lohnt es sich zu überprüfen, ob es auf der mobilen Platt-

Abb. 3: Gefundene Suchresultate zur Titelsuche „Python in a Nutshell“

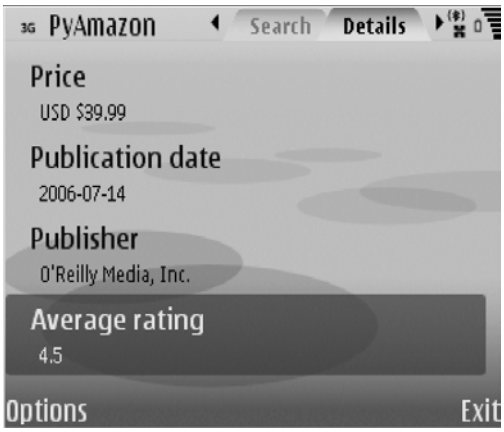


Abb. 4: Daten zu einem Buch

form verfügbar ist. Falls nicht, ist eine Suche im Internet oder eine eigene Alternative notwendig.

PyAmazon

Viele Leute bestellen heute ihre Bücher direkt von zuhause über das Internet. Dies tun sie vor allem dann, wenn sie genau wissen, welches Buch sie kaufen wollen. Das gezielte und ungezielte Schmökern in Büchern ist aber nur in einer Bibliothek oder Buchhandlung richtig gut möglich. Wenn man nun vor dem endlosen Regal steht und vor sich einen Stapel Bücher zum gleichen Thema hat, so fällt es einem oft nicht einfach, sich für eins der Bücher zu entscheiden. Dieser Entscheid könnte durch vorhandene, abrufbare Rezensionen beeinflusst werden. Bei Amazon zum Beispiel können die Kunden direkt zu jedem Artikel eine Bewertung abgeben. Gleichzeitig können andere Kunden diese dann beurteilen, ob sie sinnvoll und zutreffend ist, wodurch man eine ehrliche und gute Rezension auch erkennen kann.

Um solche Rezensionen und andere Buchinformationen direkt im Buchladen abrufen zu können, haben wir unter Einsatz von Python für S60 die kleine, mobile Applikation PyAmazon entwickelt [PyAma]. PyAmazon bietet die Möglichkeit, Buchtitel bei Amazon abzufragen (siehe Abb. 3) und den Preis, Bewertungen und andere Daten in einer benutzerfreundlichen Darstellung zu präsentieren (siehe Abb. 4). Durch die Eigenentwicklung unter Verwendung von Python haben wir die Möglichkeiten und Grenzen von Python für S60 kennengelernt und das ganze open source Projekt von Nokia genauer unter die Lupe nehmen können.

Die Bücherinformationen fragen wir über den von Amazon angebotenen E-Commerce Dienst (ECS) ab. Dabei kann die ECS-Schnittstelle auf zwei verschiedene Arten angesprochen werden: entweder mit dem Simple Object Access Protocol (SOAP) oder über den Representational State Transfer (REST). Während man bei SOAP entsprechende Parameter in XML verpackt und an die Schnittstelle schickt, kodiert man bei REST-Abfragen die benötigten Parameter direkt in der URL.

In PyAmazon haben wir uns für die REST-Methode entschieden, da die ganze Kodierung der Parameter in XML entfällt und dadurch auch weniger Daten übermittelt werden müssen. Dieser Umstand ist speziell bei mobilen Applikationen nach wie vor wichtig, da die Kosten für den Datentransfer recht hoch sind.

Den Python-Code des Kerns der REST-Abfrage ist in Listing 3 dargestellt. Mit dem Standard-Modul urllib kann über die Funktion urlopen(…) ein HTTP-Request an die URL geschickt werden. Dabei werden die Parameter speziell kodiert und mittels der Variablen post_data mitgeschickt. Als Rückgabewert erhält man dann einen Datenstrom f. Mit dem anschliessenden Befehl f.read() wird die Antwort empfangen und der Variablen response zugewiesen. f.close() schliesst danach die offene Verbindung wieder. Bei den Antworten der ECS-Schnittstelle handelt es sich immer um in XML gespeicherte Daten. Der Umfang der Antworten des Amazon-Webservices ist direkt von den

```
# url:       URL als String
# post_data: codierte Ab-
frageparameter

f = urllib.urlopen(url, post_data)
response = f.read()
f.close()
```

Listing 3: URL-Abfrage mit dem Standard-Modul urllib

```
import math
import time

pi          = 3.14159265359
measurings = 100
degrees     = 360
degsteps    = 10
totaltime   = 0.0
for m in range( measurings ):
    begin = time.clock()
    for i in range( degrees*degsteps ):
        angle = (i * pi) / (180.0 * degsteps )
        math.sin( angle )
        math.cos( angle )
        math.tan( angle )
    end = time.clock()
    totaltime += end - begin
totaltime *= 1000 # <- convert seconds to ms
average = int( totaltime / measurings )
```

Listing 4: Berechnung von Winkelfunktionen

verwendeten Parametern abhängig. Die vielfältigen Möglichkeiten und Kombinationen werden gut in der Amazon ECS-Dokumentation [AmECS] beschrieben.

Für eine kommunikationsorientierte Applikation wie PyAmazon ist Python für S60 eine gute Wahl. Da die Applikation hauptsächlich dem Beschaffen und Darstellen von Information dient und keine rechenintensiven Verarbeitungen benötigt, ist die Performanz völlig ausreichend. Das Abfragen und Empfangen der Informationen über das Internet benötigt dabei die meiste Zeit. Zudem sind alle wichtigen Funktionen in den verschiedenen Modulen vorhanden. Auch die Benutzerschnittstelle kann mit Hilfe der Nokia-API und den vorhandenen Komponenten effizient erstellt werden.

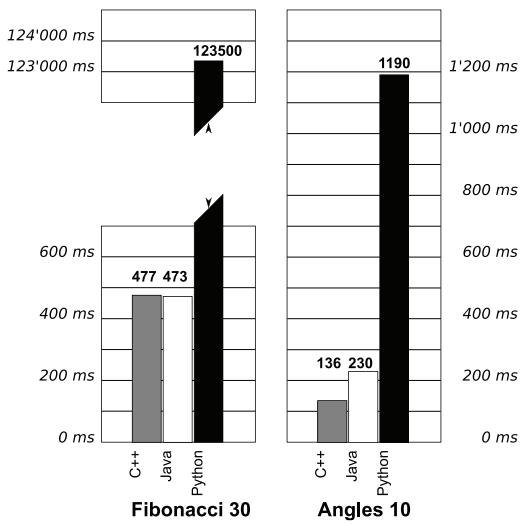


Abb. 5: C++ versus Java versus Python

Performanz

Die Flexibilität einer interpretierten Sprache muss üblicherweise mit einem Verlust an Performanz bezahlt werden. Um die Grösse dieses Verlustes abschätzen zu können, haben wir zwei unterschiedliche Benchmark-Programme je einmal in Python, Java und C/C++ implementiert.

Beim ersten Programm handelt sich um die mehrfache Berechnung einer Fibonacci-Zahl mittels einer zweifach rekursiven Funktion (siehe Listing 1). Mit der Hilfsfunktion fibonacci(n) wird die n-te Fibonacci-Zahl berechnet. Im Hauptprogramm wird diese Funktion hundertmal wiederholt aufgerufen und für jeden Aufruf die Zeit gemessen. Diese Zeiten werden aufsummiert, so dass schliesslich ein gemittelter Wert ausgegeben werden kann. Bei diesem Programm handelt es sich weniger um ein rechenintensives Programm, aber durch die Rekursion wird die Effizienz der Funktionsaufrufe auf die Probe gestellt.

Beim zweiten Programm werden die Winkelfunktionen Sinus, Kosinus und Tangens für verschiedene Winkel berechnet und dabei die Rechenkapazität ausgelotet (siehe Listing 4). Wiederum wird die Zeit gemessen und über hundert Iterationen gemittelt.

Beide Programme bringen wir in Python, Java und C++ auf einem Mobiltelefon Nokia E61 zur Ausführung. Das E61 ist ein neueres S60-Gerät der dritten Generation und verfügt über einen ARM9-kompatiblen Prozessor mit etwa 200 MHz und 25 MB Arbeitsspeicher für Programme (Abb. 2). Zudem verfügt noch kaum ein Gerätemodell über eine Hardwareunterstützung für Fliesskommaarithmetik und deswegen muss diese entweder durch die vorhandene Fixkommaarithmetik ersetzt oder aufwändig emuliert werden.

Die Ergebnisse dieses Versuchs sind in Abb. 5 klar ersichtlich. Der Performanzverlust von Python für S60 im Vergleich zu Java und C++ ist sehr deutlich. Während die Rechenperformanz beim

Programm mit den Winkelfunktionen im Vergleich mit Java um einen Faktor fünf, und im Vergleich mit C++ um fast einen Faktor neun gelitten hat, so haben die vielen rekursiven Funktionsaufrufe bei Fibonacci gleich ein 260faches an Zeit benötigt.

Wenn man aber den Arbeitsaufwand betrachtet, sieht die Sache etwas anders aus. Während man die beiden Python-Programme direkt wie in Listing 1 resp. Listing 4 auf das Mobiltelefon kopieren und ausführen kann, müssen die Java-Versionen zuerst in ein MIDlet verpackt und dann als Paket auf dem Telefon installiert werden.

Fazit und Ausblick

Mit dem Python für S60 Projekt hat Nokia eine ganz neue Art von Softwareentwicklung für Mobiltelefone möglich gemacht. Mit ihrer gut dokumentierten API ermöglichen sie den einfachen Zugriff auf die Funktionen des Telefons. Die Umsetzung der Python-Standardbibliothek ist im Allgemeinen sehr gut, lediglich einige wenige Module fehlen noch. Durch die laufende Weiterentwicklung des Projekts ist längerfristig damit zu rechnen, dass auch die fehlenden Module von Nokia noch umgesetzt werden. Zwischenzeitlich lassen sich durch die Hilfe der aktiven Community die meisten dieser Schwierigkeiten umgehen.

Die klare Struktur und die dynamischen Eigenschaften dieser Programmiersprache führen schnell und einfach zu Resultaten und man sieht während der Entwicklung schön, wie sich die einzelnen Teile zu einem Ganzen zusammenfügen. Es ist eine elegante Art zu programmieren.

Wenn man die Performanz von kompilierten Programmen mit der von interpretierten Skripten vergleicht, sind die Skript-Programme deutlich langsamer. Die geringere Performanz ist bei vielen, vor allem kommunikationsorientierten Applikationen für den Benutzer nicht spürbar und ausreichend. Bei rechenintensiven Applikationen muss sich der Entwickler überlegen, wie gut sich der rechenaufwändige Teil separieren und auslagern lässt, so dass er als externes Modul in C/C++ realisiert werden kann. Während auf handelsüblichen Computern die Rechenkapazität und Speichermengen auch bei Python-Programmen für fast jeden Zweck ausreichend sind, fallen die Ressourcen auf Mobiltelefonen recht knapp aus.

Für grössere Projekte lohnt sich der generelle Einsatz von C/C++ oder Java, insbesondere wenn die Applikation performanzkritisch oder rechenintensiv ist.

Aus unserer Sicht eignet sich Python für S60 allerdings sehr gut, um schnell einen ersten Prototypen einer Applikation zu erstellen. Und wer sich einmal intensiver mit Python auseinandergesetzt hat, wird sich schnell damit anfreunden und die Vorzüge dieser dynamischen Programmiersprache zu schätzen lernen.

Referenzen

[AmECS] Dokumentation Amazon ECS Dienst, http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=19

[ApLic] Apache Lizenzen, http://www.apache.org/licenses

[JaLib] Dokumentation der Java-Standardbibliothek, http://java.sun.com/j2se/1.4.2/docs/api/

[JaSpc] Java Language Specification, http://java.sun.com/docs/books/jls/

[PyAma] PyAmazon, http://pys60.ifi.ch/PyAmazon/

[PyLib] Dokumentation der Python-Standardbibliothek, http://docs.python.org/lib/

[PySpc] Python Language Specification, http://docs.python.org/ref/ref.html

[PyWeb] Offizelle Python Webseite, http://www.python.org

[PyXml] cElementTree für PyS60, http://ssalmine.googlepages.com/somepys60extensions

[NoFor] Offizielles Nokia Forum, http://discussion.forum.nokia.com/forum/forumdisplay.php?forumid=102

[NoWik] Offizielles Nokia Wiki, http://wiki.opensource.nokia.com/projects/Python_for_S60

[SFS60] PyS60 auf SourceForge, http://sourceforge.net/projects/pys60/

Digitale Bildverarbeitung in der Röhreninspektion

Bei der Inspektion von Abwasser- und Frischwasserröhren wird seit gut einem Jahrzehnt auf die Hilfe von fahrbaren Kanalrobotern mit aufgesetzter Videokamera gesetzt. Während früher die Videobilder nur manuell ausgewertet wurden, so kommen heute Methoden der digitalen Bildverarbeitung zum Zuge. Wir beschreiben in unserem Artikel sowohl einen Verarbeitungsschritt zur Erstellung einer Innenwandabwicklung der Röhre als auch einen Analyseschritt zur automatischen Detektierung von Rohrmuffen. Die Erkennungsrate bei den Rohrmuffen hängt wesentlich von der Qualität der Innenwandabwicklung ab. Unsere Algorithmen erreichen auf einem herkömmlichen PC eine Erkennungsrate von ca. 99% bei einer Geschwindigkeit von mehr als drei Metern pro Sekunde.

Martin Schindler, Christoph Stamm | martin.schindler@fhnw.ch

In einer typischen, europäischen Stadt wird ein im Untergrund liegendes Röhrensystem für Frisch- und Abwasser eingesetzt. Ein solches Rohrleitungsnetz muss periodisch (ca. alle 8 Jahre) auf kleinere und grössere Schäden untersucht werden, um einerseits eine unerwünschte Versickerung von Frischwasser und anderseits eine gefährliche Kontaminierung des Frischwassers durch in die Röhren eindringende Fremdstoffe zu vermeiden. Solche regelmässigen Inspektionen des Rohrleitungsnetzes sind eine zeitaufwändige und daher auch kostspielige Aufgabe. Um besser verstehen zu können, welcher Aufwand damit verbunden ist, betrachten wir hier beispielsweise die Stadt Zürich. Das Abwassereinzugsgebiet von Zürich (und ein paar weniger benachbarter Gemeinden) umfasst gut 59 Quadratkilometer Bauzone und beherbergt rund 390'000 Einwohner. Die Gesamtlänge der öffentlichen Abwasserkanäle in diesem Gebiet ist 925 Kilometer. Dazu kommen private Leitungen von total 3'051 Kilometern [Neu07].

Bereits seit zehn Jahren wird bei der Inspektion auf die Hilfe von fahrbaren Kanalrobotern mit aufgesetzter Videokamera gesetzt. Sind früher die Videodaten in einem Begleitfahrzeug auf VHS-Bändern aufgezeichnet und von Sachverständigen begutachtet worden, so geht der Trend heute deutlich in Richtung der automatisierten Schadensdetektion mit Hilfe digitaler Bildverarbeitung.

Die Firma CDLab AG in Murten (Schweiz) [CDLab] gehört zu den weltweit führenden Anbietern von Röhreninspektions-Software. In einem von der KTI (Förderagentur für Innovation des Bundes) finanziell unterstützten einjährigen Projekt[1] haben Mitarbeiter der CDLab zusammen mit Angehörigen des Instituts für Mobile und Verteilte Systeme der Fachhochschule Nordwestschweiz die Röhreninspektions-Software in einzelnen Teilen wesentlich verbessert und um eine automatische Rohrmuffenerkennung erweitert.

Digitale Röhreninspektion

Auf einem Kanalroboter befindet sich eine analoge Videokamera mit einer Fischaugoptik. Die analogen Videodaten werden über ein Kabel zum Begleitfahrzeug gesendet und dort mit Hilfe von zusätzlicher Hardware digitalisiert. Diese digitalisierten Rohdaten der Fischaugansicht werden dann von der Röhreninspektions-Software in eine entsprechende Innenwandabwicklung transformiert (Abb. 1). Dabei werden in regelmässigen Abständen von der Fischaugansicht ringförmige Bilder von schmalen Rohrabschnitten extrahiert, diese zu Bildstreifen transformiert, und die Streifen schliesslich durch Aneinanderreihen in eine kontinuierliche Innenwandabwicklung der Röhre überführt. Im Gegensatz zur bisherigen, analogen Inspektionstechnik erlaubt diese Darstellung der Innenwandabwicklung einen rascheren Überblick über den Leitungszustand eines kompletten Rohrabschnittes und ein einfacheres Ausmessen von Objekten und Schäden.

Erstellung der Innenwandabwicklung

Der Kanalroboter ist über verschiedene Kabel (Strom/Video), welche über eine Kabeltrommel laufen, mit der Hardware zur Analog/Digital-Wandlung verbunden (Abb. 2). Anhand der Länge des abgewickelten Kabels lässt sich an der Kabeltrommel die zurückgelegte Distanz des Kanalroboters ungefähr ablesen. Die zurückgelegte Distanz wird benötigt, um alle paar Zentimeter aus den Halbbildern des PAL-Videosignals ein digitales Gesamtbild (Fischaugansicht) zu erstellen. Aus dieser Fischaugansicht wird ein ringförmiger Bereich entnommen und in eine rechteckige Zy-

1 KTI-Projektnummer: 7785.2 ESPP-ES

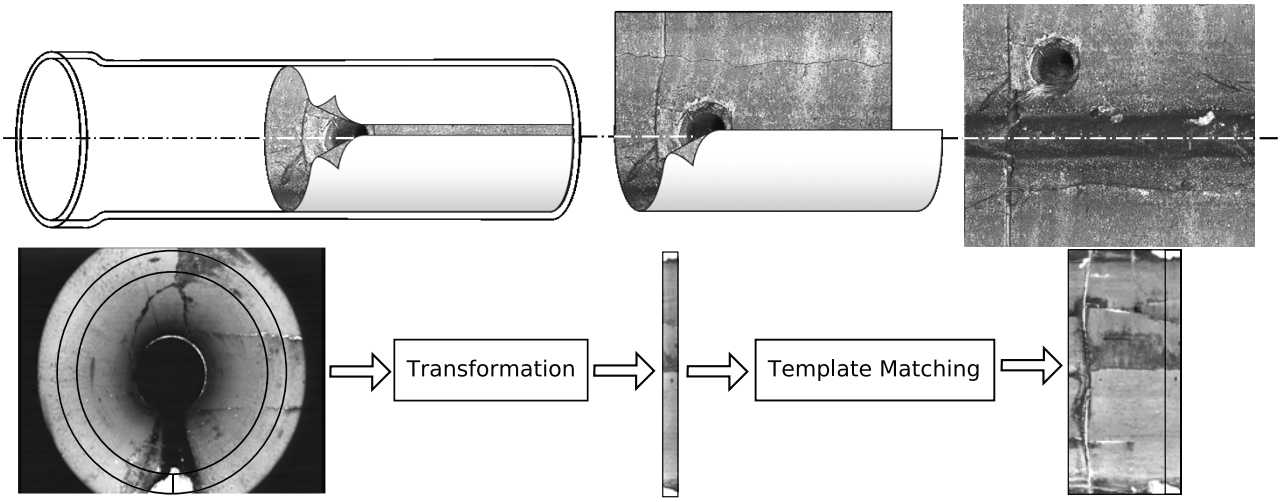


Abb. 1: Vom Fischaugbild zur Innenwandabwicklungsansicht

linderprojektion (Streifen) umgerechnet. Da die Messung der zurückgelegten Distanz an der Kabeltrommel relativ ungenau ist, entstehen ungleiche Abstände zwischen den Streifen. Zudem wird der ringförmige Bereich immer an derselben Position des Gesamtbildes entnommen, was zu Verzerrungen in der Abwicklungsansicht führen kann, sollte der Roboter sich nicht exakt parallel zur Röhre bewegen. Eine korrekte, entzerrte Abwicklungsansicht ist aber eine wichtige Voraussetzung für eine möglichst fehlerfreie, automatische Detektierung und Vermessung von Strukturen im Rohr. Eine falsche Aneinanderreihung der Streifen könnte zu fehlenden oder mehrfach abgebildeten Strukturen führen.

Ein weiteres Problem ist die starke Helligkeitsabnahme (bis zu 40% auf nur wenigen Zentimetern) beim Fischaugbild in Richtung des Bildzentrums. Solche Helligkeitsunterschiede müssen so gut wie möglich korrigiert werden, damit die resultierenden, starken Helligkeitssprünge an den Streifenrändern nicht fälschlicherweise als Rohrmuffen detektiert werden.

Eine erste Möglichkeit, die Streifen korrekt zu einer Innenwandansicht zusammenzusetzen, besteht darin, die Lage des Roboters in der Röhre genau zu bestimmen, und dadurch den passenden Bereich des Fischaugbildes in den entsprechenden Streifen zu transformieren. Die so gewonnen Streifenbilder könnten dann ohne weitere Bearbeitung zu einer korrekten Abwicklungsansicht aneinandergefügt werden. Die exakte Positions- und Lagebestimmung des Roboters innerhalb der Röhre erfordert jedoch Anpassungen am Roboter und ist somit nicht direkter Gegenstand dieses Projektes. Allerdings besteht eine Möglichkeit, zumindest die Geschwindigkeit und damit die zurückgelegte Distanz des Roboters nur mit Hilfe des Videobildes zu bestimmen und zwar unter Ausnutzung der ansonsten störenden Interlacing-Effekte der PAL-Videobilder. Dieser Ansatz ist in [Wäl07] untersucht worden. Die Geschwindigkeit direkt an

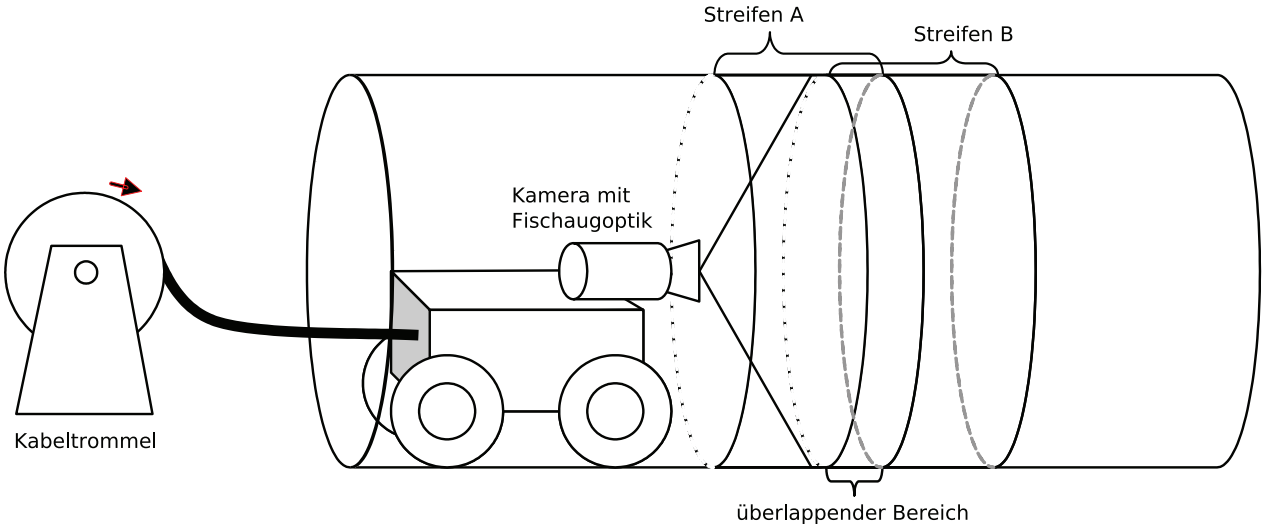


Abb. 2: Überlappung der einzelnen Streifen

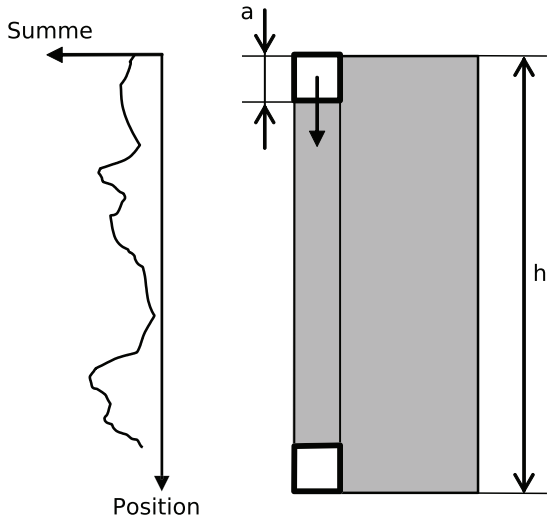


Abb. 3: Erzeugung des Kontrasthistogramms

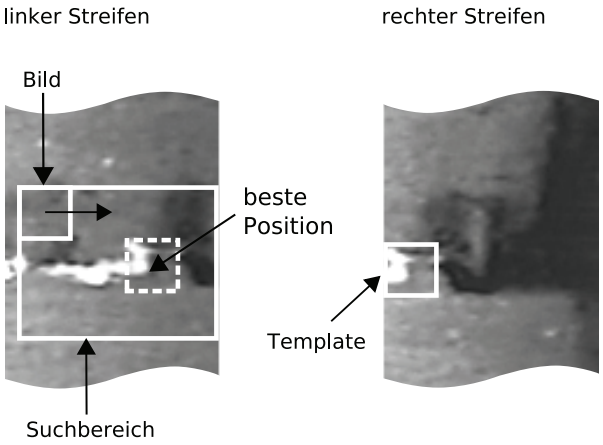


Abb. 4: Template mit entsprechendem Suchbereich

den Rädern des Roboters zu messen, kann infolge des Durchdrehens der Räder in glitschigen Passagen zu falschen Ergebnissen führen.

Eine zweite und von uns favorisierte Möglichkeit besteht darin, die ungenau transformierten Streifen mit Hilfe der digitalen Bildverarbeitung so gut wie möglich zu einer Innenwandabwicklung zusammenzusetzen. Da zwei aufeinander folgende Streifen einen überlappenden Bereich haben (Abb. 2), können diese mit Hilfe eines Template-Matching-Verfahrens zusammengesetzt werden.

Unter Template-Matching wird der Vergleich eines Suchbildes (Template) mit einem gleich grossen Bildbereichs verstanden [WikiPM]. Für jede mögliche Position des Suchbildes innerhalb des Bildes wird die Ähnlichkeit bzw. der Grad der Übereinstimmung zwischen dem Suchbild und dem korrespondierenden Bildbereich ermittelt. Als Mass für die Ähnlichkeit kann zum Beispiel die Summe der quadratischen Differenzen verwendet werden. Da die einzelnen Streifen über eine gegenseitige, schmale Überlappung verfügen, lässt sich durch Anwendung des Template-Matching-Verfahrens ein optisches Flussfeld zwischen Bildpunkten am rechten Rand des linken Streifens und Bildpunkten am linken Rand des benachbarten rechten Streifens berechnen. Dabei wird der rechte Rand des linken Streifens zum Suchbereich und Teile des linken Randes des rechten Streifens zum Suchbild. Somit kennen wir für Randpixel von benachbarten Streifen zusätzlich einen Verschiebungsvektor. Mit Hilfe dieses Flussfeldes können die benachbarten Streifen anschliessend punktweise transformiert und nahtlos aneinander gereiht werden. Gleichzeitig können auch Helligkeitsunterschiede zwischen benachbarten Streifen ausgeglichen werden.

Die Auswahl der Suchbilder ist entscheidend für ein erfolgreiches Matching. Bildbereiche mit wenig oder gar keinem Kontrast, zum Beispiel neue PVC-Röhren ohne Ablagerungen oder Rohrbereiche unter Wasser, sind oft ungeeignet für ein erfolgreiches Matching. Um geeignete Suchbilder zu finden, werden die ersten paar Pixelspalten des rechten Streifens mit einem Sobel-Filter [WikiSob] auf Kanten untersucht und anschliessend ein Kontrasthistogramm erstellt, indem alle Kantenpixel innerhalb der möglichen Suchbildflächen gezählt werden (Abb. 3). So erhält man zu h – a + 1 Positionen einen Summenwert, wobei h der Bildhöhe und a der Höhe des Suchbildes entsprechen.

Aus diesem Kontrasthistogramm lassen sich die Suchbilder mit dem stärksten Kontrast herauslesen. Zu jedem dieser Suchbilder (Template) wird nun ein eingeschränkter, lokaler Suchbereich im linken, benachbarten Streifen definiert (Abb. 4). Die Breite des Suchbereichs ist direkt abhängig von der Breite der Streifenüberlappung. Die Abweichungen in vertikaler Richtung sind wesentlich kleiner als die horizontalen Verschiebungen, deshalb kann der Suchbereich in vertikaler Richtung kleiner gewählt werden.

Ausgehend vom Suchbild und dem entsprechenden Suchbereich gilt es nun eine Position zu finden, bei der die Korrelation zwischen Suchbild und einem gleich grossen Bildbereich innerhalb des Suchbereichs maximal ist. Dazu gibt es mehrere Möglichkeiten: Wie bereits erwähnt, können die Summen der absoluten oder quadratischen Differenzen zwischen Bildbereich und Suchbild berechnet werden, was zwar sehr einfach und schnell ist, aber hier nicht geeignet, da Unterschiede in der Helligkeit zwischen Bildbereich und Suchbild zu unnötig schlechten Korrelationen führen. Was wir brauchen ist ein Mass, welches invariant gegenüber linearen Helligkeitsänderungen ist. Der Korrelationskoeffizient Kor(S, T) erfüllt diese Bedingung,

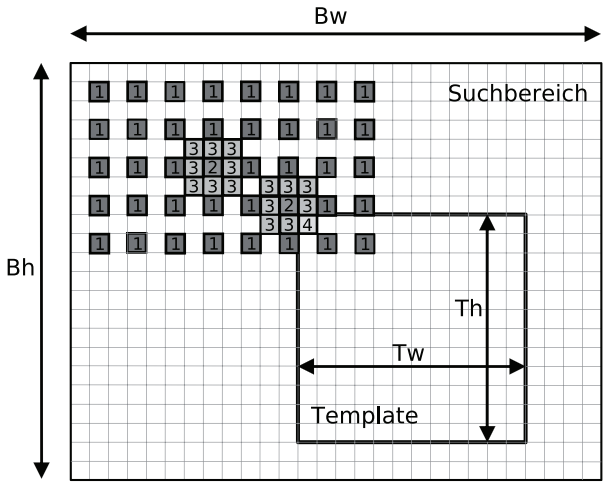


Abb. 5: Reduziertes Suchverfahren

$$Kor(S,T) = \frac{Cov(S,T)}{\sqrt{Var(S) \cdot Var(T)}} = \frac{n\sum_{i=1}^{n} s_i t_i - \sum_{i=1}^{n} t_i \cdot \sum_{i=1}^{n} s_i}{\sqrt{\left(n\sum_{i=1}^{n}(s_i)^2 - \left(\sum_{i=1}^{n} s_i\right)^2\right) \cdot \left(n\sum_{i=1}^{n}(t_i)^2 - \left(\sum_{i=1}^{n} t_i\right)^2\right)}}$$

wobei S dem Suchbereich [s1..sn] und T dem Template (Suchbild) [t1..tn] entsprechen. Er nimmt Werte zwischen –1 und +1 an. Bei einem Absolutwert von eins besteht ein vollständig linearer Zusammenhang zwischen Suchbereich und Suchbild. Ist der Korrelationskoeffizient null, so gibt es keinen linearen Zusammenhang.

Der Aufwand der Berechnung des Korrelationskoeffizienten Kor(S, T) ist linear in der Anzahl Bildpunkte n des Suchbildes. Somit ergibt sich ein Gesamtaufwand für ein Template-Matching mit einem Suchbereich bestehend aus m Bildpunkten in der Grössenordnung von O(n m). Das heisst, mit einer Halbierung des Suchbereichs halbiert sich auch der Rechenaufwand. Da es sich bei unseren Bilddaten um natürliche Bilder handelt, darf davon ausgegangen werden, dass der unmittelbare Nachbar der besten Position auch eine hohe Korrelation hat. Somit ist es möglich, nur für jede zweite Position innerhalb des Suchbereichs den Korrelationskoeffizienten zu berechnen und anschliessend lokal (in einer 3x3-Umgebung) um die zwei besten Positionen herum, den vermeintlich höchsten Korrelationswert und somit die beste Position mit höchster Übereinstimmung zu finden (Abb. 5).

Dieses Template-Matching-Verfahren wird auf alle zuvor selektierten Suchbilder angewendet. Dadurch erhalten wir zu jedem Template ein entsprechendes Bild im Suchbereich und somit die gesuchten Verschiebungsvektoren (Abb. 6a). Diese Verschiebungsvektoren bilden dann die Stützwerte des optischen Flussfeldes. Um eine möglichst gute Qualität bei der Aneinanderreihung der Streifen zu gewährleisten, werden allfällige Ausreisser unter den Verschiebungsvektoren vorgängig entfernt. Wir definieren dazu drei Bedingungen, die erfüllt sein müssen, damit ein Verschiebungsvektor als gültig klassifiziert wird: Der Absolutwert des Korrelationskoeffizienten muss mindestens 0.9 betragen, die Abweichung eines einzelnen Vektors zum Durchschnitt aller Vektoren darf höchstens 15 Bildpunkte und der Abstand zu einem direkten Nachbarvektor darf höchstens 5 Bildpunkte betragen. Diese Werte haben wir anhand verschiedener Testreihen empirisch ermittelt.

Nach Bestimmung des optischen Flussfeldes zwischen zwei benachbarten Streifen wird dieses auf den rechten Streifen angewendet (Abb. 6b), so dass sich dieser nahtlos an den vorangehenden, linken Streifen anfügen lässt. In einem letzten Schritt werden dann noch die Helligkeitsunterschiede zwischen den beiden Streifen ausgeglichen (Abb. 6c).

Rohrmuffenerkennung

Eine automatische Detektierung von Rohrverbindungen (Rohrmuffen) dient vorwiegend der schnelleren Beurteilung des Bildmaterials über allfällige Röhrenschäden. Da vor allem Muffen schadenanfällig sind, müssen diese zwingend untersucht werden. Sind die genauen Positionen der Muffen bekannt, so kann der Inspekteur per Mausklick direkt zur nächsten Muffe springen und somit den weniger interessanten Zwischenteil überspringen und damit Zeit gewinnen. Zudem gibt es auch Vorschriften, welche ein Bildprotokoll von sämtlichen Muffen verlangen.

Wenn wir davon ausgehen, dass der Kanalroboter genau parallel und zentral ein Rohr abfährt, so sind Rohrmuffen in einer korrekten Innenwandansicht als klar hervortretende Linien erkennbar (Abb. 7a). Wie bereits angedeutet, fährt der Kanalroboter aber keineswegs immer parallel zum Röhrenverlauf, was zu sinusförmig geschwungenen Ansichten von Muffen führt (Abb. 7b/c). In manchen Situationen sind die Muffen zudem nicht als durchgehende Linien erkennbar, da Rohrverschmutzungen die Linienstruktur überschatten. Eine automatische Muffenerkennung muss mit diesen Schwierigkeiten umgehen können. In einer anderen Diplomarbeit [Man07] ist wiederum mit zusätzlicher Roboter-Hardware – drei fest aufgesetzten Lasermodulen – der Rohrdurchmesser auf wenige Millimeter genau ermittelt worden, was ausreicht, um die leichte Durchmesseränderung

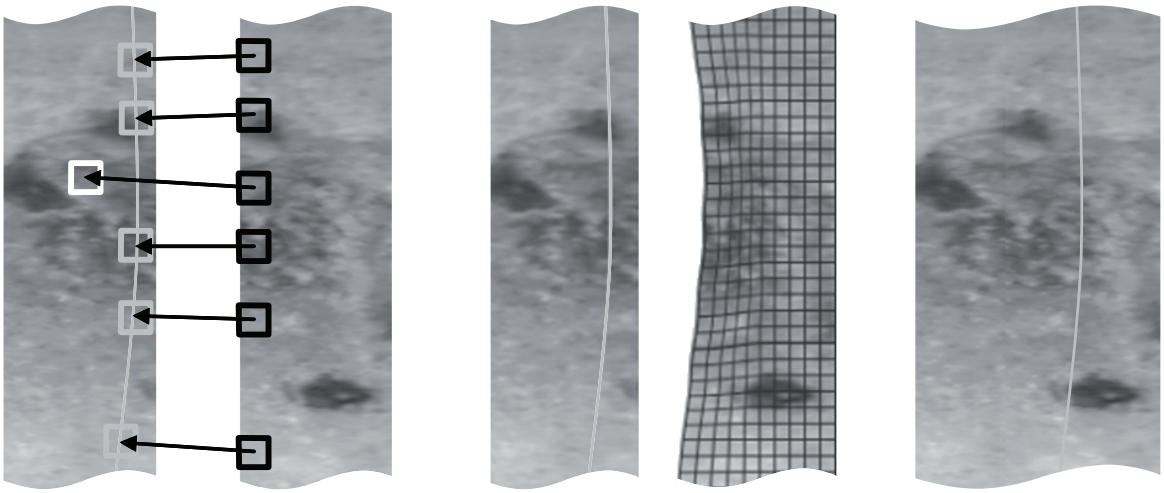


Abb. 6: a) Verschiebungsvektoren b) optisches Flussfeld c) Innenwandabwicklung

bei Rohrmuffen hinreichend genau zu erkennen. In unserem Projekt sind wir wiederum von einem Standard-Kanalroboter ohne Laser-Module ausgegangen. Das heisst, wir haben nur mit einfachen Mitteln der digitalen Bildanalyse der zuvor erstellten Innenwandansichten eine automatische Muffenerkennung realisiert.

Das Grundprinzip unserer Erkennungsmethode basiert auf der Erkennung von vertikal verlaufenden Strukturen und dem Zählen von Bildpunkten in vertikalen Kanten (Kantenpixel) pro Bildspalte. Damit nur die vertikalen Kanten sichtbar werden, wird wiederum ein Sobel-Filter eingesetzt. Das gefilterte Bild wird zudem mit Hilfe einer Schwellwertfunktion auf ein Schwarzweissbild reduziert. Weil es aber in Röhren noch andere Objekte mit vertikalen Strukturen gibt, müssen weitere, heuristische Nebenbedingungen erfüllt werden, wie zum Beispiel die Verteilung der Kantenpixel.

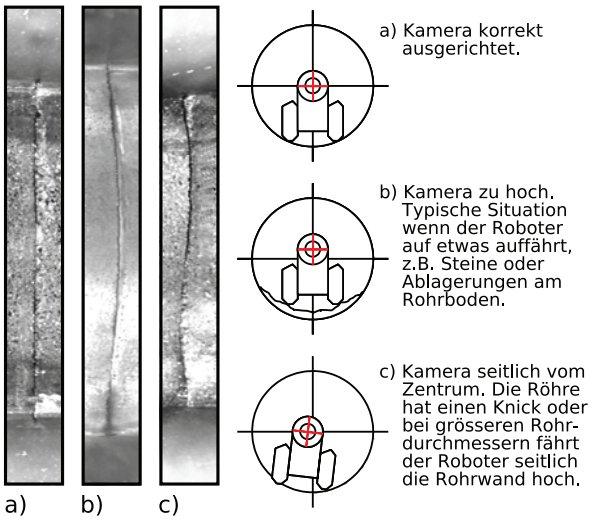


Abb. 7: Ursache der sinusförmigen Rohrmuffen

Da ein aufgenommener Röhrenabschnitt oft mehrere hundert Meter lang sein kann, wird die erstellte Innenwandansicht oft mehrere 1000 Pixel breit. Solch grosse Bilder lassen sich nur mit erheblichem Speicheraufwand verarbeiten und werden daher vorgängig in sich überlappende Einzelbilder mit vorgegebener Breite unterteilt (Abb. 8). Die Überlappung dient dazu, dass Muffen, welche auf einem Einzelbild nur teilweise sichtbar, auf dem nächsten Einzelbild vollständig ersichtlich sind.

Für jedes Einzelbild wird ein rechteckiges Fenster, welches über die gesamte Bildhöhe geht, von links nach rechts über das Bild verschoben und zu jeder Fensterposition die Anzahl Kantenpixel gezählt, die sich innerhalb des Fensters befinden (Abb. 9). Das Fenster muss genügend breit sein, damit auch Muffen, welche nicht genau senkrecht im Bild liegen, möglichst gut erkannt werden. Auf der andern Seite, sollte das Fenster nicht zu breit sein, um hauptsächlich lineare, vertikale Strukturen ausfindig zu machen. Aus den ermittelten Summen der Kantenpixel werden die lokalen Maxima ausgewählt und als potentielle Muffenpositionen selektiert. Damit Muffen von anderen Objekten, welche ebenfalls vertikale Kanten haben, z.B. Wasserrinnen eines seitlichen Rohreinlaufs, unterschieden werden können, bedarf es oft weiterer heuristischer Selektionen. In einem ersten Schritt überprüfen wir, ob die potentielle Muffe durchgängig ist, d.h. ob es zwischen den einzelnen Kantenpixel keine grösseren vertikalen Unterbrüche gibt. Falls dem so ist, dann überprüfen wir in einem zweiten Schritt ob, diese Kantenpixel gleichmässig über die gesamte Bildhöhe verteilt sind. Dazu wird der Mittelwert der vertikalen Positionen pro Fenster berechnet. Falls sich dieser Mittelwert stark von der halben Höhe des Bildes unterscheidet, dann ist es wahrscheinlich keine Muffe.

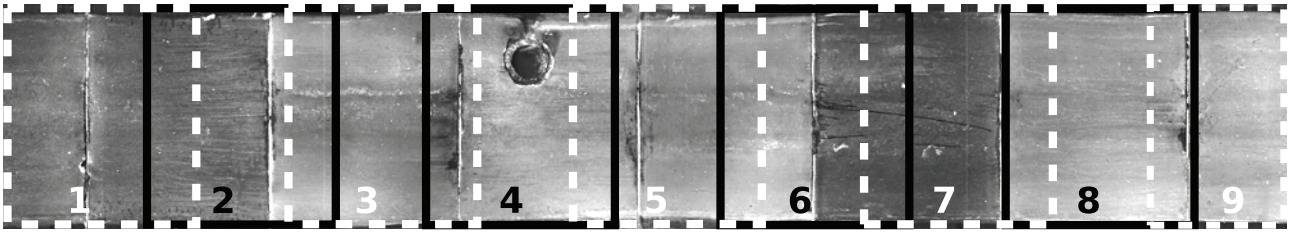
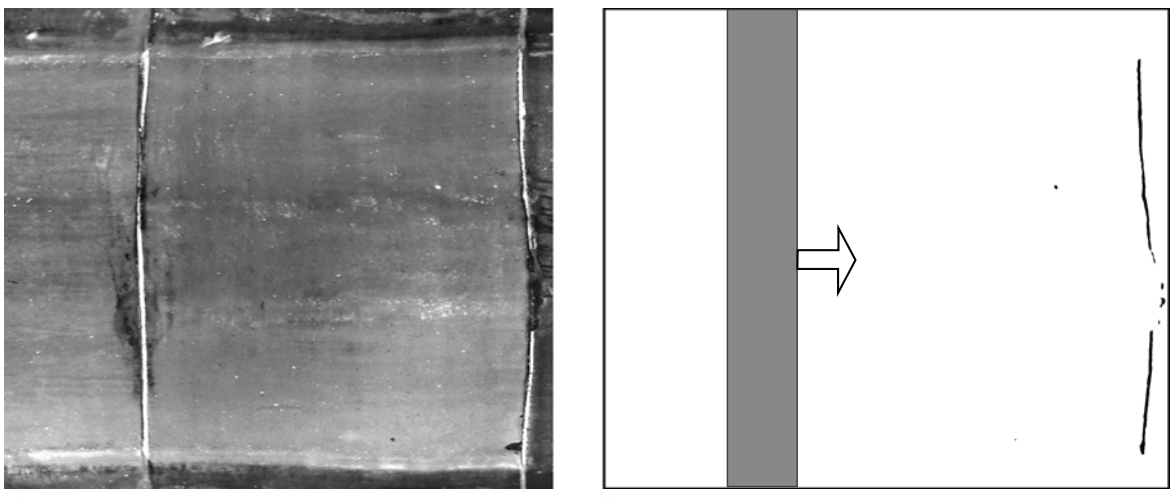


Abb. 8: Aufteilung der Bilder der Innenwandansicht

Nachdem die Rohrmuffen zumindest teilweise erkannt worden sind, werden sinusförmig geschwungene Muffen zusätzlich begradigt, um ein möglichst genaues Abbild der Realität zu erhalten. Bei der Begradigung wird eine auf die Muffe interpolierte Sinuskurve benötigt. Da bekannt ist, dass es sich bei den sinusförmig geschwungenen Muffen um genau eine Periode einer Sinusschwingung handelt, müssen nur die Phasenverschiebung, die Amplitude und der Abstand zum Bildrand berechnet werden. Dazu betrachten wir die einzelnen Kantenpixel als lineares Signal im Bildraum, so dass mittels eindimensionaler Fourieranalyse diese drei Parameter berechnet werden können. Den Abstand zum Bildrand erhält man direkt aus der ersten Frequenzkomponente. Die Phasenverschiebung und die Amplitude werden in der zweiten Frequenzkomponente abgebildet. Daher werden alle Frequenzen bis auf die ersten beiden auf null gesetzt und anschliessend mit der inversen Fouriertransformation wieder in ein Bildsignal transformiert. Das Resultat ist eine interpolierte Sinuskurve über der Muffe (Abb. 10b). In einem von der Amplitude des Signals abhängigen Fenster werden die Pixel so gestreckt oder gestaucht, dass die Muffe eine vertikale Gerade bildet (Abb. 10c/d).

Einsatz und Ausblick

Wir haben ein erstes heuristisches Verfahren zur Erstellung von Rohrinnenwandansichten und ein zweites zur Erkennung von Rohrmuffen entwickelt. Die entwickelten Algorithmen für das korrekte Zusammensetzen der einzelnen Bildstreifen zu einer Innenwandansicht haben eine hohe Effizienz, so dass das Zusammensetzen der Bilder noch während der Rückfahrt des Roboters aus der Kanalröhre geschehen kann. Die Erkennung von Muffen ist zeitaufwändiger und erfolgt üblicherweise erst in einem nachgeschalteten Post-Pro-



a) b)

Abb. 9: Kantenbild der Innenwandansicht mit zwei Rohrmuffen

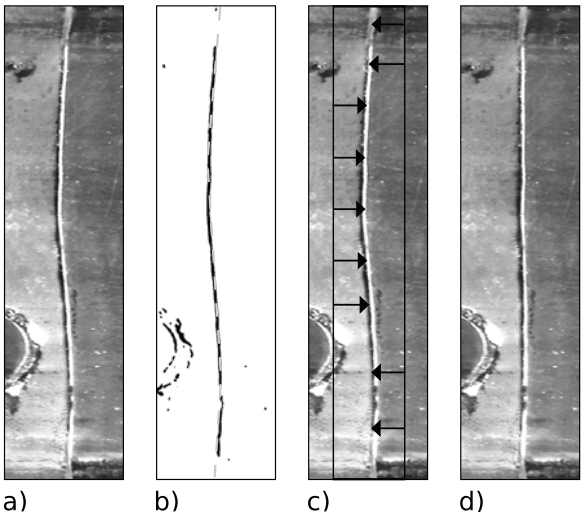


Abb. 10: Begradigung einer Rohrmuffe

cessing. Mit Hilfe unserer neu entwickelten Algorithmen, lassen sich auf einem herkömmlichen Personalcomputer (Pentium 4 Prozessor, 3GHz) Innenwandansichten von Röhren mit einer Geschwindigkeit von ca. 3.5 Meter pro Sekunde auf Muffen untersuchen. Bei Tests von 28 unterschiedlichen Röhrenabschnitten mit einer Gesamtlänge von 860 Metern und insgesamt 667 Muffen, haben unsere Algorithmen 660 Muffen korrekt erkannt und sieben Strukturen fälschlicherweise als Muffe deklariert. Dies entspricht einer Erkennungsrate von ca. 99%.

In diesem KTI-Projekt wurde auch ein erster Ansatz zur automatischen Erkennung von seitlichen Rohreinläufen entwickelt. Die Erkennungsrate von Rohreinläufen liegt momentan noch bei weniger als 70%. Ein Hauptgrund für diesen schlechten Wert liegt im momentanen Algorithmus, der nur runde Objekte erkennt, die Rohreinläufe aber bedingt durch deren Anschlusswinkel und Zustand eine andere Form aufweisen. Deshalb ist die Erkennung von Rohreinläufen auch ein wichtiges Thema im Nachfolgeprojekt.

Referenzen

[CDLab] Kanalinspektion mit WinCan, http://www.wincan.com

[Man07] Mancktelow M., Image Processing in Pipe Inspection, Diplomarbeit Studiengang Informatik, Fachhochschule FHNW, 2007, http://www.fhnw.ch/personen/marcus.hudritsch

[Neu07] Neuhold G., Sanierung der Abwasserverhältnisse Zürich-Nord (SAN), ERZ, 2007, http://www.stadt-zuerich.ch/internet/erz/home/.../Artikel_SAN_032007.pdf

[Wäl07] Wälchli Y., Image Processing in Pipe Inspection. Diplomarbeit Studiengang Informatik, Fachhochschule FHNW, 2007, http://www.fhnw.ch/personen/marcus.hudritsch

[WikiPM] Wikipedia, Pattern Matching, Mai 2007, http://de.wikipedia.org/wiki/Pattern_Matching

[WikiSob] Wikipedia, Sobel-Operator, http://de.wikipedia.org/wiki/Sobel-Operator

Fachhochschule Nordwestschweiz FHNW
Institut für Mobile und Verteilte Systeme

Steinackerstrasse 5
CH-5210 Brugg-Windisch

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