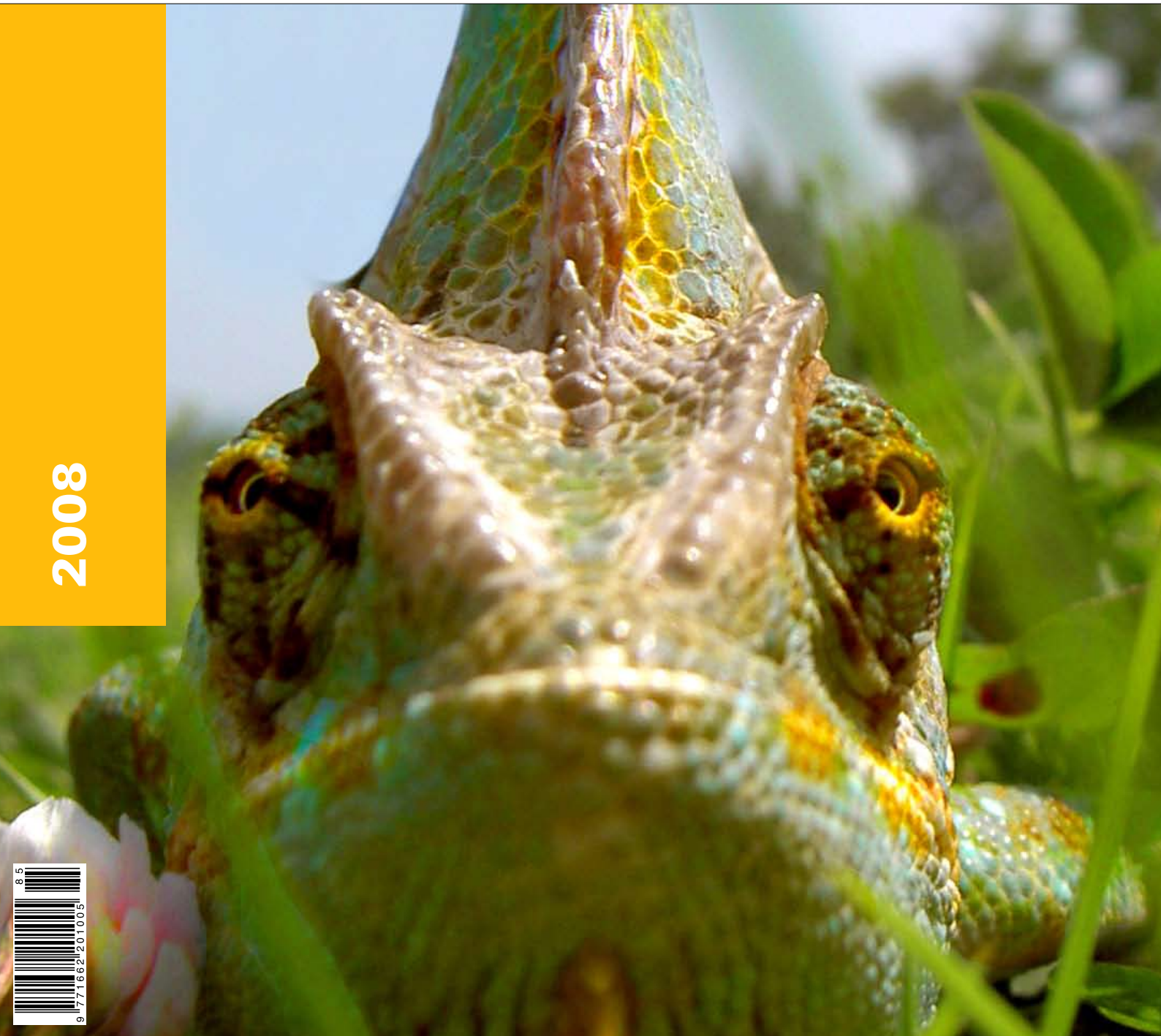
Fachhochschule Nordwestschweiz
Hochschule für Technik

Institut für Mobile und Verteilte Systeme

FOKUS REPORT

2008



Editorial

Chamäleons sind dafür berühmt, dass sie die Hautfarbe nach Belieben und innert kürzester Zeit verändern können. Zur Tarnung vor Feinden variieren sie vor allem die Helligkeit der Hautfärbung und manipulieren ihre Körperform. Sie passen sich damit der Umgebung an. Sobald sie aber einen Artgenossen erblicken, können sie mittels kurz andauernden Farbveränderungen ihre Stimmungen, Gefühle und Absichten zum Ausdruck bringen. Vor allem die Männchen betreiben einen verschwenderischen Aufwand, um Rivalen einzuschüchtern oder Weibchen anzulocken.

Von kontextsensitiven Systemen oder Software erwarten wir, dass sie sich ähnlich der Chamäleons ihrer Umgebung – dem Kontext – anpassen. Ein angepasstes Verhalten kann für die Benutzer einen Gewinn darstellen. Sitzt zum Beispiel eine PDA-Benutzerin gemütlich im Restaurant und plant ihre Heimkehr mit dem öffentlichen Verkehr, so wäre ihr bereits geholfen, wenn beim Abrufen des Tramfahrplans automatisch die nächstliegenden Haltestellen und die aktuelle Uhrzeit einbezogen würden. Die Software würde also den Kontext – Ort und Zeit – *geschickt* verwenden. Hierbei gibt es verschiedene Arten der Anpassungsfähigkeit von Software zu unterscheiden, wobei aus meiner Sicht die dynamische Adaptivität die weitaus spannendste ist. Innerhalb dieser lässt sich weiter zwischen kompositioneller und parametrischer Adaption unterscheiden[1]. Bei der kompositionellen Adaption werden einzelne Komponenten des Systems zur Laufzeit ausgetauscht oder neu zusammengestellt. Etwas einfacher zu verstehen ist parametrische Adaption, bei der Sensorwerte den Verlauf der Software beeinflussen; im Beispiel der angezeigte Tramfahrplan, welcher sich nach der Position des mobilen Gerätes richtet. In beiden Arten der Adaption ist zu beachten, dass ungeschickt eingesetzte Kontextinformation, welche zu einem undurchsichtigen, vielleicht sogar anmassenden Verhalten der Software führt, auch jede Menge Frust erzeugen kann. Um beim Beispiel zu bleiben, wäre der Benutzerin wenig gedient, wenn nur der Fahrplan der allernächsten Haltestelle erscheinen würde, obwohl eine wenig weiter gelegene Station eine schnellere, weil direkte Tramverbindung anbieten würde. Ganz generell reicht es nicht, ein (pseudo-)intelligentes Verhalten in einer Software nachzuahmen – was alleine schon schwierig genug ist –, denn aus unserem sozialen Miteinander wissen wir nur allzu gut, dass auch intelligentes Verhalten nicht immer verstanden wird und zu heftigstem Unverständnis führen kann. Entscheidend in erster Linie ist, dass wir das Ver-

1 K. Geihs. Selbst-adaptive Software. Informatik Spektrum, Band 31, Heft 2, 2008.

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: Th. Müller-Schenker
Erscheinungsweise: jährlich
Druck: Effingerhof Brugg
Auflage: 200

ISSN: 1662-2014

halten eines Systems erfassen und verstehen oder zumindest plausibel nachvollziehen können.

Die am häufigsten verwendete Kontextinformation ist die Zeit. Sie steht auf fast allen mobilen Geräten in meist ausreichender Genauigkeit zur Verfügung. Schon ganz anders sieht es mit der Verfügbarkeit von Positionsdaten zur genauen Ortsbestimmung aus. Zwar sind GPS-Empfänger heute recht weit verbreitet und werden auch immer öfter direkt in PDAs integriert, dennoch haben längst nicht alle mobilen Benutzer jederzeit und uneingeschränkt Zugriff auf solche Positionsdaten. Eine sehr grobe Lokalisierung eines Mobiltelefons wäre aber auch ohne GPS anhand der aktuell zugeordneten Basisstation und deren Koordinaten möglich. Üblicherweise geben die Betreiber der GSM-Netze diese Koordinaten jedoch nicht frei. Doch einmal mehr werden die hortenden und bewahrenden Instanzen vom innovativen Markt überrannt, wie das schöne Beispiel der Skyhook-Wireless-Technik in den neusten iPods und iPhones zeigt. So kann zum Beispiel in urbanen Zentren in den USA und Europa anhand der Signalcharakteristiken der bereits vorhandenen WLAN-Netze die eigene Position auf wenige Meter genau bestimmt werden. Dazu sind ständig aktualisierte Datenbanken mit positionsabhängigen Signalcharakteristiken notwendig.

Sofern das mobile Geräte über zusätzliche Sensoren verfügt, so können neben Ort und Zeit auch weitere physikalische Messgrössen wie Temperatur, Geschwindigkeit, Beschleunigung usw. sinnvoll einbezogen werden. Diesem physischen Kontext («knowledge in the world»)[2] stehen zwei weitere Kontexte zur Seite: Einerseits bringt die Benutzerin eigenes Wissen mit («knowledge in the head») und anderseits enthält das mobile Gerät unter Umständen bereits eine grössere Datensammlung («knowledge in the pocket»). Erst der korrekte Einbezug aller drei Kontexte würde die Voraussetzung schaffen, der Benutzerin das Gefühl einer intelligenten Software zu vermitteln. Während auf den direkten Zugriff des «knowledge in the head» noch längere Zeit verzichtet werden muss, stehen die beiden anderen Kontexte bereits zu einem gewissen Grad zur Verfügung. Es liegt nun an den Informatikern und anderen Wissenschaftern, entsprechende Datenmodelle und Komponenten zu entwickeln, um dieses vorhandene Umgebungswissen in Anwendungen sinnvoll einzubeziehen.

Jürg Luthiger und Philip Handschin zeigen in ihrem Beitrag, wie ein TomTom GO Navigationssystem um einen Webserver erweitert werden kann, so dass dieser mobile Webserver über das Internet ansprechbar wird. Dadurch lassen sich jederzeit die Positionsdaten des Navigationssystems über das Internet abrufen, wodurch lokalitätsabhängige Dienste und Anwendungen angeboten werden können.

Eine ganz andere Art, um zu Positionsdaten zu kommen, beschreibt Carlo Nicola in seinem Artikel über aktive RFID-Tags und das ZOMOFI-System von Siemens. ZOMOFI bietet mit einer sehr kleinen Anzahl von Controllern die Ortung von aktiven RFID-Tags an. Dadurch können Waren, welche über ein solches Tag verfügen, innerhalb einer Lagerhalle auf wenige Meter genau lokalisiert werden. Dies scheint vor allem im Bereich der Logistik neue Ansätze und Prozesse zu inspirieren.

Im Artikel von Hans-Peter Oser wird die Entwicklung einer lokalitätsabhängigen, mobilen Anwendung (HikeTracker) exemplarisch aufgezeigt. Mit dem HikeTracker können Wanderer und Biker ihre aktuelle Position mit elektronischen Versionen der Schweizer Landeskarten auf einem mobilen Telefon oder PDA darstellen. Um eine möglichst grosse Verbreitung zu ermöglichen, stehen einerseits verschiedene Versionen des HikeTracker für unterschiedliche Plattformen zur Verfügung und anderseits ist der Quellcode als Open Source frei erhältlich.

Das Thema Open Source wird im Beitrag von Andreas Hofmann und Christoph Stamm aus Sicht der richtigen Lizenzwahl und der Bewirtschaftung eigener Open Source Projekte genauer untersucht. Anhand der Bibliothek libpgf, die als Grundbaustein für ein Navigationssystem benutzt werden kann, zeigen die Autoren, welche Aspekte bei der Lizenzwahl berücksichtigt werden sollen und wie mit vertretbarem Aufwand ein Open Source Projekt der Öffentlichkeit zur Benutzung und Weiterentwicklung zur Verfügung gestellt werden kann.

Während Programmbibliotheken, wie zum Beispiel libpgf, sich problemlos mit Unit-Test-Werkzeugen überprüfen lassen, gilt dies für interaktive Programme mit Dialogen und vielfältigen Abhängigkeiten nur selten. Um ganze Arbeitsabläufe mit einer interaktiven Software zu testen, braucht es andere Werkzeuge, zum Beispiel FIT, das Open Source Framework für integrierte Tests. Das Autorentrio Cecilia Garcia Garcia, Martin Kropp und Wolfgang Schwaiger führt in dieses Framework ein und zeigt den Einsatz und Nutzen anhand mehrerer Beispiele auf.

Auch die sorgfältigsten Tests können lediglich die Fehlerhaftigkeit einer Software zu Tage bringen. Um die Korrektheit eines Programms zu beweisen genügen sie aber nicht. Zudem haben Software-Entwickler aus naheliegenden Gründen nur ein eingeschränktes Bedürfnis, Fehler in ihrer Software zu finden. Daher stellt Edgar Lederer in seinem Artikel die Java Modeling Language und verschiedene assoziierte Werkzeuge vor, welche von einer Software-Testabteilung eingesetzt werden können, um die Korrektheit von Java-Quellcode zu beweisen.

Prof. Dr. Christoph Stamm
Forschungsleiter IMVS

2 S. Timpf. „Location-based Services" – Personalisierung mobiler Dienste durch Verortung. Informatik Spektrum, Band 31, Heft 1, 2008.

TomTom Navigationssystem als kontextsensitiver Webserver

In dieser Arbeit wird ein TomTom GO Navigationssystem um einen Webserver mit dynamischer Webseiten-Generierung erweitert, so dass dieser mobile Webserver wie ein normaler Webserver über das Internet (und das Telekom Netz) ansprechbar wird. Durch den Zugriff aus dem Internet können zum Beispiel die Positionsinformationen des Navigationsgerätes genutzt werden, um kontextsensitive Anwendungen zu entwickeln. Es wird auch gezeigt, dass der mobile Webserver eine zusätzliche Infrastrukturkomponente benötigt, um die Interaktion in das Netz des Telekomanbieters zu ermöglichen.

Philip Handschin, Jürg Luthiger | juerg.luthiger@fhnw.ch

Navigationsgeräte wie das TomTom [TomTom] sind mobile Kleinrechner mit einer erstaunlichen Funktionalität. Ein leistungsstarker GPS Empfänger und ausgereiftes Kartenmaterial erlauben eine exakte Positionsbestimmung. Diese Kontextinformation wird für das Finden eines Zielortes, für die Planung einer Reiseroute, für die Anzeige interessanter Orte und vieles mehr genutzt. Dank Bluetooth kann über ein mobiles Telefon eine Kommunikationsverbindung in die GSM Infrastruktur und in das Internet aufgebaut werden, um über diese Verbindung Sprache und Daten auszutauschen.

Navigationsgeräte werden in einer verteilten Anwendung ausschliesslich als Clients eingesetzt, obwohl sie in ihrer Rechenleistung, ihrer Speicherkapazität und ihrer Funktionalität zu den Servern der 90er Jahre, dem Beginn des Web Zeitalters, überlegen sind.

Die Idee, ein mobiles Gerät als Webserver zu nutzen, ist nicht neu. In [Pra03] wird eine Architektur für Web Server auf Microsoft Pocket PC präsentiert und in [Wik06] beschreibt Nokia eine interessante Umgebung für mobile Telefone. Ihr Webserver ist eine Portierung des bekannten Apache HTTP Deamon mit Python und WebDAV Unterstützung. Mittels Webzugriff lassen sich zum Beispiel Nachrichten übers Internet direkt aufs Handy schicken, ein auf dem Handy «gehostetes» Blog führen, SMS bequem auf dem Computer tippen oder auch die Kontakte bearbeiten. Dafür muss nur die entsprechende Server-Software auf dem Handy installiert werden. Der Zugriff aus dem Internet auf das Handy erfolgt über den normalen Webbrowser. Das Unternehmen stellt zu Testzwecken und Proof-of-Concept eine gesamte Infrastruktur [Nokia] für die Idee der mobilen Websites zur Verfügung.

Navigationsgeräte bieten die attraktive Möglichkeit, die Positionsdaten des Gerätes in die Anwendung zu integrieren und über diese Kontextinformation neuartige Applikationen zu entwickeln. Diese Idee wurde in einer Informatik Semesterarbeit [Han07] aufgegriffen. Das Ziel der Arbeit war die Umsetzung der Idee aus [Wik06], deren Portierung und die Integration der Positionsinformationen in eine Anwendung für die TomTom GO Plattform [TomTom]. Als Prototyp wurde ein Applikation realisiert, in der die Position des entsprechenden TomTom Gerätes in Google Maps visualisiert und in einem normalen Webbrowser dargestellt wird. So kann zum Beispiel eine Speditionsfirma über die bestehende Internet Informatikinfrastruktur die Position ihrer Lieferwagen verfolgen und benutzergerecht darstellen.

Ein Webserver für die TomTom Plattform

TomTom GO basiert auf einem ARM Linux Betriebssystem. Um auf dieser Plattform eine attraktive Webanwendung realisieren zu können, muss der entsprechende Webserver dynamische Webseiten unterstützen. Denn nur mit dynamischen Webseiten ist möglich, die Position jederzeit dem aktuellen Standort nachführen zu können. Die Wahl fiel schliesslich auf den Klone [Klone] Webserver. Klone ist ein Multiplattform-Webserver speziell für Embedded-Geräte entwickelt, der dank C/C++-Scripting auch dynamische Inhalte erlaubt, ohne auf zusätzliche Komponenten wie PHP oder Perl angewiesen zu sein. Durch den Verzicht auf Erweiterungen des Websevers wie PHP soll vor allem die Leistung gesteigert und die CPU-Auslastung gesenkt werden.

Mittels des Software Development Kit (SDK) können Entwickler ihre dynamischen Inhalte in C oder C++ schreiben. Normale HTML Seiten werden durch sogenannte Scriptlets ergänzt. Analog wie bei den Java Server Pages umfassen diese Scriptles den dynamischen Teil, der in diesem Falle jedoch in C/C++ geschrieben wird.

Das Resultat der dynamische Webpage aus Listing 1 ist eine Auflistung des Root Ordners. Ein spezieller Compiler übersetzt anschliessend den Quelltext in nativen Code, der dann als Bina-

```
<%!
    #include <string.h>
    #include <sys/types.h>
    #include <dirent.h>
%>
<html>
<head>
    <title>My first page</title>
    <link rel="stylesheet" href="kl.css" type="text/css">
</head>
<body>
    <h3>Root dir content:</h3>
    <ul>
    <%
        DIR *dirp;
        struct dirent *dp;

        dirp = opendir("/");
        while ((dp = readdir(dirp)) != NULL)
            io_printf(out, "<li>%s</li>", dp->d_name);
        closedir(dirp);
    %>
    </ul>
</body>
</html>
```

Listing 1: Beispiel einer dynamischen Webpage mit dem Klone Webserver

ry gegen den Klone Server gelinkt und integriert werden kann.

Kommunikationsinfrastruktur

Ein Webserver auf der TomTom Plattform macht erst dann Sinn, wenn dieser Server auch aus dem Internet angesprochen werden kann. Ein explizites Ziel dieser Arbeit war es, dass ein beliebiger Webbrowser den TomTom Server jederzeit über das Internet adressieren und nutzen kann. In [Wik06] wurde gezeigt, dass diese Aufgabe auch über gängige Telekommunikationsnetze wie GSM oder GPRS erreicht werden kann. Der Zugang in solche Kommunikationsnetze ist aber nicht trivial, da normalerweise ein Verbindungsaufbau von aussen in das Netz des Telekomanbieters über Firewalls und NAT Systeme (Network Address Translation) blockiert wird. Diese Restriktion kann jedoch umgangen werden, wenn die Verbindung von einem Gerät initiiert wird, das sich im Telekomnetz und hinter dem Firewall befindet. Um diesen Verbindungsaufbau entgegen nehmen zu können, muss aber ein zusätzlicher Gateway vor dem Firewall, also im Internet, installiert werden. Dieser Gateway ist die Schnittstelle zwischen dem Telekom-Netz und dem Internet. Er muss, wie in Abbildung 1 dargestellt:

- den Verbindungsaufbau vom mobilen Geräten entgegennehmen;
- die Verbindung zum mobilen Gerät verwalten;
- die Anfrage des Browser entgegennehmen;
- die Kommunikation zwischen Webbrowser und mobilem Endgerät herstellen.

Der HTTP Request vom Browser zum mobilen Webserver erfährt mit diesem Lösungsansatz eine Umleitung über den Gateway. Da dieser Gateway ausserhalb des Telekom-Netzes steht, ist er jederzeit über das Internet erreichbar. Der Gateway kann deshalb die Anfrage entgegennehmen und diese über eine proprietäre Socket Verbindung zum entsprechenden mobilen Webserver weiterleiten. Die Socket Verbindung muss vorher jedoch vom mobilen Webserver aufgebaut werden, was zum Beispiel bei einer Anmeldung beim Gateway erfolgen kann. Sobald die Socket Verbindung steht, ist der mobile Webserver über das Internet ansprechbar. Um aus dem Internet adressierbar zu werden, muss der mobile Webserver eine eindeutige Identifikation erhalten.

TomTom Webserver Identifikation im Internet

Wie bereits beschrieben, wird sich der TomTom Benutzer am Gateway anmelden müssen, um das mobile Gerät im Internet sichtbar machen zu können. Nach der Anmeldung ist eine proprietäre, aber eindeutige Verbindung über TCP/IP Sockets zwischen dem mobilen Gerät und dem Gateway vorhanden. Der Gateway muss alle Verbindungen zu den entsprechenden mobilen Geräten verwalten können. Der TomTom Handler wird diese Aufgabe übernehmen. Er wurde in dieser Arbeit konzipiert und implementiert. Der Partner auf der mobilen Seite ist der Gateway Connector. Er stellt die Interaktion mit dem mobilen Webserver her.

Für die Verwaltung der Verbindung erstellt der TomTom Handler eine HandlerMap. In dieser werden alle aktiven TomTom Verbindungen eingetragen. Als Schlüssel kann zum Beispiel der übermittelte Name des TomToms verwendet werden. Der Vorteil dieses Ansatzes ist es, dass der Name nur einmal vorkommen darf und deshalb auch für

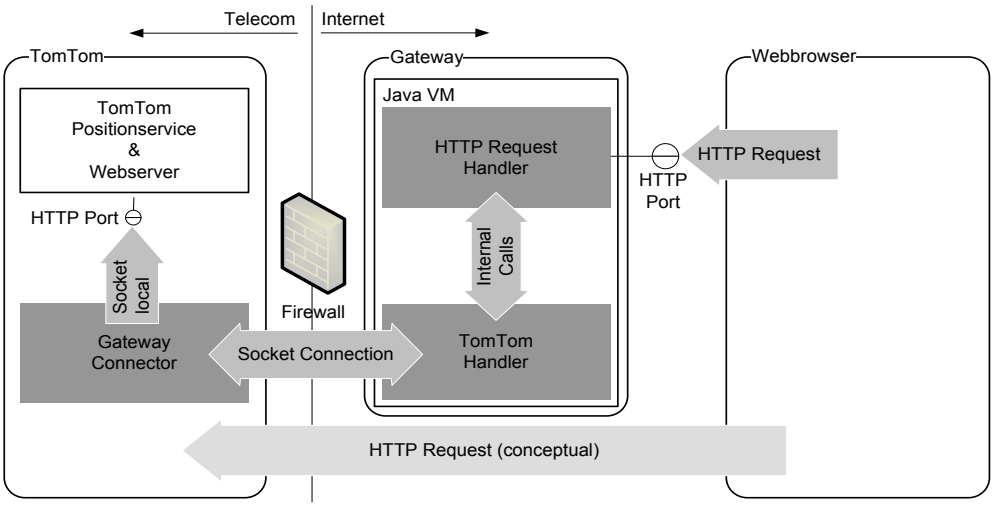


Abbildung 1: Das Gateway Konzept

die Generierung einer eindeutigen URL eingesetzt werden kann. Sobald der Gateway eine TomTom Anmeldung akzeptiert hat, wird die Verbindung über einen eigenen Thread kontrolliert.

Im Internet ist das TomTom Gerät anschliessend unter der HTTP Adresse `http://<gateway-host>:<gateway-port>/<tomtom-name>` sicht- und ansprechbar.

Kommunikation zwischen Browser Client und TomTom Webserver

In Abbildung 2 sind die wichtigen Kommunikationsverbindungen aufgezeichnet.

Verbindung 1:
Die Verbindung 1 ist eine TCP/IP Socket Verbindung zwischen dem mobilen Gerät und dem Gateway. Sie wird beim Starten des Gateway Connectors auf dem TomTom aufgebaut. Dieser Verbindungsaufbau entspricht der Anmeldung auf dem Gateway unter einem eindeutigen Namen.

Verbindung 2:
Die Verbindung 2 zeichnet den HTTP Request nach. Der HTTP Request kann nicht direkt an den mobilen Webserver erfolgen, da er hinter der Firewall des Telekom Anbieters versteckt ist. Der HTTP Request muss deshalb über das Internet an den HTTP Port des Gateways geleitet werden. Hier kann der HTTP Request Handler aus der URL das TomTom detektieren, das angefragt ist. Der TomTom Handler übernimmt den Request und leitet die Anfrage über die bestehende TCP/IP Socket Verbindung 1 an den Gateway Connector des TomTom's weiter. Vom Gateway Connector gelangt der HTTP Request schliesslich zum mobilen Webserver. Der mobile Webserver liest mit Hilfe des Positionsservices die aktuelle Position des mobilen Gerätes aus und generiert die HTML Antwort.

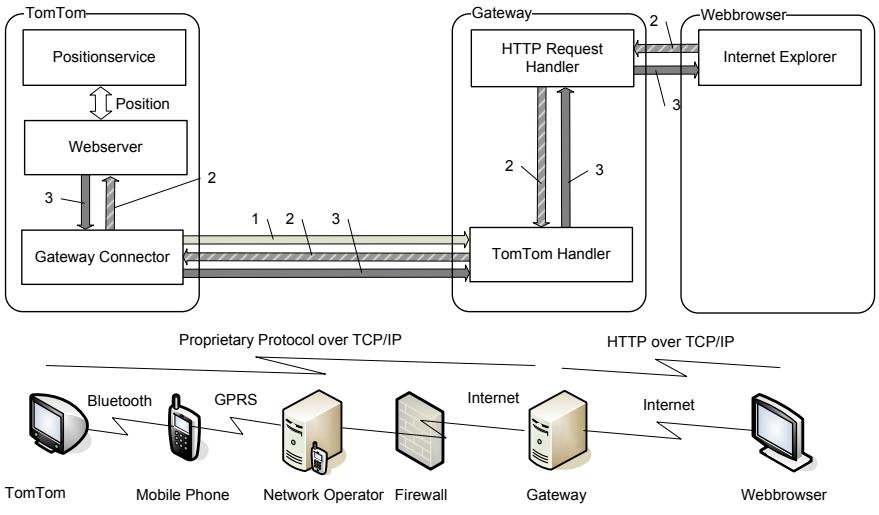


Abbildung 2: Kommunikation zwischen Browser und mobilem Web Server

Verbindung 3:
Die HTML Response verfolgt den gleichen Verbindungsstrang in umgekehrter Richtung wie der HTTP Request, um anschliessend im Webbrowser als Webseite dargestellt zu werden.

In der Abbildung 2 sind die notwendigen Hardware Komponenten aufgezeichnet, die es für diese Kommunikationsaufgabe braucht. Das TomTom baut mit Hilfe eines mobilen Telefons, das über Bluetooth gekoppelt ist, die GPRS Verbindung in das Internet, zum Gateway auf.

Die Positionsdaten des TomTom können nun auf der Client Seite ausgewertet werden, die z. B. mit entsprechendem Kartenmaterial visualisiert werden.

Abbildung 3: Tomsite mit dem BMW Zeichen als Positionsmarkierung

Visualisierung der TomTom Position im Browser Client

Google Maps bietet nicht nur schöne Karten und Satellitenbilder an, sondern auch eine einfach zu nutzende Programmierschnittstelle. Der Einsatz von Google Maps zur Visualisierung der TomTom Position bringt den grossen Vorteil, dass die aufwändigen Kartendaten unabhängig vom mobilen Navigationsgerät zur Verfügung stehen. Durch die externe Verknüpfung der Positionsdaten mit dem Kartenmaterial kann auf dem mobilen Gerät das kostenintensive Verwalten der Kartendaten weggelassen werden. Zwischen Webbrowser und mobilen Webserver werden lediglich die Positionsdaten ausgetauscht.

Über eine Programmierschnittstelle (API) können Webentwickler die Google Karten in ihre eigenen Websites einbauen. Dadurch werden die Funktionen der Google Maps auf der eigenen Seite nutzbar. Zusätzlich können in den Karten eigene Markers gesetzt werden, um auf die Weise eine Position explizit erkennbar zu machen (Abbildung 3). Listing 2 stellt die entsprechende Umsetzung für den Klone Webserver dar.

```
<%!
    // some includes, variable definitions, ...
%>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>TomTom Position Service</title>
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;
            key=<my google key>" type="text/javascript"></script>

    <script type="text/javascript">

    //<![CDATA[
    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.addControl(new GSmallMapControl());
        map.addControl(new GMapTypeControl());
        map.setCenter(new GLatLng(
            <% // call to the mobile webserver which returns the
               // position of the TomTom device %>
            ,true),13);

        var icon = new GIcon();
        icon.image = "bmwlogo.png";
        icon.iconSize = new GSize(50 ,47);
        icon.iconAnchor = new GPoint(31, 15);
        icon.infoWindowAnchor = new GPoint(5, 1);

        function createMarker(point, name) {
            var marker = new GMarker(point,icon);
            GEvent.addListener(marker, "click", function() {
                marker.openInfoWindowHtml("Position von <b>" + name +
                "</b>");
            });
            return marker;
        }
```

```
          marker = createMarker(map.getCenter(),"My Car");
          map.addOverlay(marker,icon);

          window.setTimeout(function() {
              location.reload()
          }, 30000);

        } // end if()
      } // end load()

      //]]>
      </script>
  </head>

  <body onload="load()" onunload="GUnload()">
      <p>TomTom Position Service</p>

      <div id="map" style="width: 500px; height: 500px"></div>

      <p>Diese Seite wird alle 30 Sekunden automatisch aktualisiert.</p>
      <p>F&uuml;r Manuelle Aktualisierung, bitte Knopf dr&uuml;cken:
          <input type="button" value="Aktualisieren"
          onClick="location.reload()"
          class="altButtonFormat"></p>

  </body>
  </html>
```

Listing 2: Integration von Google Maps in die dynamische Webpage

Herausforderungen

In diesem Abschnitt werden ein paar Aspekte aufgegriffen, die typisch für eine Tomsite sind und eine nähere Betrachtung bedürfen. Es sind dies das Offline Verhalten, die Kosten und die Sicherheit, bzw. Datenschutz.

Offline

Im Gegensatz zu einer normalen Website wird eine Tomsite nicht immer online sein. Es ist eine typische Eigenschaft einer mobilen Site, dass sie oft nicht ansprechbar ist, weil z.B. das Navigationsgerät nicht in Betrieb oder die entsprechende Verbindung in das Internet nicht aufgebaut ist. Wäre die Tomsite direkt aus dem Internet adressierbar, hätte der Entwickler fast keine Möglichkeiten diesen offline Modus benutzerfreundlich und aussagekräftig darzustellen. «Server not found» ist in dieser Situation die normale Fehlermeldung.

Dank der Umleitung über den Gateway aber, er wird als normaler Webserver auf eine hohe Verfügbarkeit ausgelegt, kann mit der vorliegenden Infrastruktur eine benutzergerechte Antwort gegeben werden. Aufgrund der An- und Abmeldung weiss der Gateway wann eine Tomsite online ist. Im offline Status kann der Gateway diesen Zustand mit einer aussagekräftigen HTML Response beantworten.

Findet ein Verbindungsunterbruch ohne Abmeldung statt, erkennt dies der Gateway erst bei einem konkreten Aufruf der entsprechenden Tomsite. In diesem Falle kann ein Timeout abgewartet werden, um anschliessend alle vorhandenen Informationen aus dem Gatway zu entfernen und die Tomsite als offline zu markieren.

Kosten

Die Verbindung aus dem Telekomnetz kostet. Die Kosten berechnen sich aus der kommunizierten Datenmenge.

In einem Feldversuch wurde deshalb das System über eine Zeitperiode von 2 Stunden aktiv genutzt. In einer ersten Phase wurde alle 30 Sekunden eine Abfrage abgesetzt; total 120 * 2 = 240 Abfragen. Der anschliessenden Stresstest mit 80 Anfragen kurz hintereinander, führten zu insgesamt 320 Tomsite Anfragen. Das Mobiltelefon musste in dieser Zeitspanne 0.7 Megabytes (MB) senden und 0.3 MB empfangen, das sind 1 MB Datenverkehr. Mit den Tarifen von Sunrise (Erste Hälfte 2007) würde dies ca. Fr. 3.50 kosten.

Sicherheit, Datenschutz

Da das TomTom Navigationsgerät persönliche Daten speichern kann, ist es wichtig, dass der Besitzer die Kontrolle darüber hat, wer seine Tomsite nutzen darf und wer nicht. Deshalb sollte die Tomsite ein entsprechendes Administrationsinterface zur Verfügung stellen, über das der Benutzer eine einfache Benutzeradministration zur Verfügung hat. Da die vorliegende Arbeit jedoch als Machbarkeitsstudie ausgelegt ist, wurde das Thema Sicherheit, Datenschutz bewusst aus der Aufgabestellung gestrichen.

Zusammenfassung

In dieser Arbeit wurde gezeigt, dass es ist möglich ist, einen voll funktionsfähigen Webserver mit dynamischer Unterstützung auf einem TomTom GO Gerät zu installieren und auszuführen. Mit Hilfe eines Gateways, der die Verbindung zwi-

schen dem Netz des Telekomanbieters und dem Internet herstellt, wird diese sogenannte Tomsite wie eine normale Website von einem Browser aufrufbar.

Es stellt sich hier die Frage, ob der Gateway, der wie der mobile Webserver ebenfalls einen HTTP Port für die HTTP Requests zur Verfügung stellen muss, nicht auch die Webserver Funktionalität des mobilen Gerätes übernehmen kann. Der Gateway und das mobile Geräte nutzen eine Socket Verbindung, um die Kommunikation zwischen Telekomnetz und Internet sicherstellen zu können. Diese Verbindung kann teuer sein, da sie über die kommunizierte Datenmenge abgerechnet wird. Bei der Entwicklung der mobilen Applikation muss man deshalb den notwendigen Datentransfer beachten. Im Extremfall kann die Generierung der HTML Seite deshalb auch auf den Gateway ausgelagert werden. Das TomTom wird in einem solchen Falle nur die Positionsdaten und nicht die gesamte Webseite an den Gateway senden, was den Datentransfer zwischen diesen beiden Partner stark reduziert. Dieses Vorgehen ist aber nur dann sinnvoll, wenn alle über den Gateway angeschlossenen mobilen Geräte die gleiche internetfähige Applikation bereitstellen. Sobald verschiedene Anwendungen über den Gateway abgehandelt werden, wird der Aufwand auf dem Gateway gross, die unterschiedlichen Inhalte performant aufzubereiten und zu verwalten. In dieser Situation ist ein dezentralisierter Ansatz und eine klare Aufgabenteilung sinnvoller, da er einfacher wartbar, erweiterbar und flexibler ist. Ebenfalls ist der Einsatz eines mobilen Webservers unumgänglich, wenn das mobile Endgerät auch über WLAN eine Verbindung in das Internet aufbauen kann und in diesem Falle ein Gateway nicht mehr notwendig ist.

Der mobile Webserver auf einem Navigationsgerät ist vor allem für kontextsensitive Applikationen interessant, welche die Positionsinformationen des Gerätes nutzen möchten. Dies wurde beispielhaft mit einer Integration der Fahrzeugposition in Google Maps demonstriert.

Leider hat TomTom in der Zwischenzeit die Entwicklung des Software Development Kit (SDK) für die TomTom GO Plattform eingestellt. Das SDK stellte dem Programmierer eine umfangreiche Bibliothek zur Verfügung, um auf die spezifischen Funktionen des Linux-basierten Navigationssystems zugreifen zu können, um zum Beispiel die GPS Position auszulesen.

Wir sind jedoch überzeugt, dass solche Ideen und Ansätze in Zukunft vermehrt Anwendung finden werden. Diese Arbeit hat uns gezeigt, dass die technischen Möglichkeiten bereits heute vorhanden sind.

Referenzen

[Pra03] Pratistha D., Nicoloudis N., Cuce S., A Micro-Services Framework on Mobile Devices, International Conference on Web Services, 2003

[Wik06] Wikman J., Dosa F., Tarkiainen M., Personal Website on a Mobile Phone, Nokia Research Center Helsinki, 2006

[Han07] Handschin P., TomSite, Semesterarbeit Studiengang Informatik, Fachhochschule FHNW, 2007

[Klone] KoanLogic, Klone Embedded Webserver, http://koanlogic.com/klone/index.html

[Nokia] Nokia, Mobile Web Server, http://mymobilesite.net/

[TomTom] TomTom International, http://www.tomtom.com

Genauigkeit eines Lokalisierungssystems für aktive RFID-Tags

In diesem Artikel werden die Lokalisierungseigenschaften des RFID-Systems ZOMOFI der Firma Siemens AG untersucht. Zwei Fragestellungen werden beantwortet: Wie gross ist die Fehlerfläche bei der Lokalisierung unter der Bedingung, dass die Signalausbreitung der RFID-Leser genau kreisförmig ist und welchen Einfluss hat die Lage der RFID-Leser auf die Präzision, mit der ein Objekt lokalisiert wird?

Zdena Koukolikova, Carlo Nicola, Christoph Stamm | carlo.nicola@fhnw.ch

Im Bereich verteilter und mobiler Softwaresysteme spielen «intelligente Sensoren» (wie RFID: Radio Frequency Identification) eine grundlegende Rolle. Von ihnen hängt nicht nur der Grad der Granularität bei der Erfassung der einzelnen Elemente innerhalb eines Systems ab, sondern von ihnen wird auch die Grenze bestimmt, innerhalb der dezentrale Sensoren autonom entscheiden können.

Seit einigen Jahren erforschen wir am Institut für Mobile und Verteilte Systeme, wie die Signale von RFID-Sensoren zuverlässig erfasst, übers Internet verteilt und in einer Applikation integriert und analysiert werden können. Wir haben mit dem Problem des *track and trace* von Luxusgütern (am Beispiel von Weinflaschen des oberen Preissegments) begonnen, bei dessen Lösung neben den klassischen 1D- und 2D-Barcodes auch einfache RFID-Etiketten benutzt wurden [NK02]. Besonderes Gewicht wurde auf die Mobilität der Erfassung der Daten gelegt, weswegen Mobiltelefone als RFID-Lesegeräte eingesetzt wurden. In einem anderen Projekt untersuchten wir in Zusammenarbeit mit der Firma Siemens, wie weit sich das Mobiltelefon mit einer NFC-RFID-Schnittstelle (NFC: *Near Field Communication*) als intelligenten Sensor umfunktionieren lässt. Dabei entwickelten wir erfolgreich Prototypen für die sichere Barbezahlung von Musikstücken (*music on demand*) mit Mobiltelefonen und RFID. Der gleiche NFC-Ansatz wurde dann konkret in einem Projekt zur Entwicklung eines intelligenten Schranks mit der Firma Lista angewendet. «Intelligent» bedeutet hier, dass jede Materialentnahme bzw. Einlage im Schrank in Echtzeit an eine Datenbank nicht nur mitgeteilt wird (automatische bzw. chaotische Lagerverwaltung), sondern auch dass jegliche Materialveränderung nur von authentifizierten und autorisierten Personen durchgeführt werden kann. Sowohl die Erkennung des gelagerten Materials als auch die Authentifizierung der Personen erfolgen mit RFID-Technik.

Mit den bisher verwendeten passiven RFID-Tags [MW06] waren wir nur in der Lage, Objekte in geringer Distanz zu identifizieren, da passive RFID-Tags einen Teil der elektromagnetischen Energie des Lesevorgangs als Energiequelle benötigen. Eine allfällige Lokalisierung war zudem nur sehr ungenau möglich. Ein neues aktives RFID-Tag-System von Siemens, welches Radiosignale im 2.45 GHz-Bereich senden und empfangen kann, ermöglicht über die Objektidentifizierung hinaus die Lokalisierung des Objekts anhand der Stärke des Rücksignals. Aktive RFID-Tags sind zusätzlich mit langlebigen Batterien bestückt. Daher muss die Stärke des Lesesignals keine konstante Komponente für die Energieversorgung des Tags beinhalten. Dies hat den grossen Vorteil, dass grössere Reichweiten möglich sind.

In diesem Artikel beschreiben wir erste Untersuchungen mit aktiven RFID-Tags in einem von der KTI geförderten Projekt (INOLOG[1]). Zusammen mit dem Institut für Business Engineering[2] und der PostLogistics entwickeln wir neue Ansätze zum Einsatz von aktiver RFID-Technologie in der Logistik. Konkret untersuchen wir, ob diese Technologie die Logistik von Paletten von verschieden zusammengesetzten Küchengeräten (WG: *white goods*) entscheidend vereinfachen und effizienter gestalten kann. Nach der genauen Problemstellung stellen wir das ZOMOFI-System zur Identifizierung und Lokalisierung aktiver RFID-Tags vor und kommen dann zum Hauptteil, wo wir die Lokalisierungseigenschaften von ZOMOFI genauer untersuchen. Anschliessend besprechen wir kurz eine Verfeinerung des Lokalisierungsalgorithmus mit einem probabilistischen Ansatz

1 KTI-Projektnummer: 8912.1 PFES-ES
2 Institut für Business-Engineering (IBE): http://www.fhnw.ch/technik/ibe

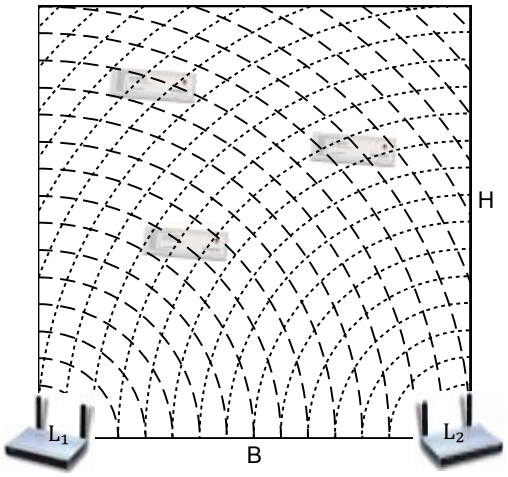


Abbildung 1: Die Lokalisierungszelle mit den zwei durch WLAN verbundenen RFID-Controllern L_1 und L_2 und drei aktiven RFID-Tags. Die Reichweite des Empfangssignals ist als ideale, kreiswellenförmige Begrenzung skizziert.

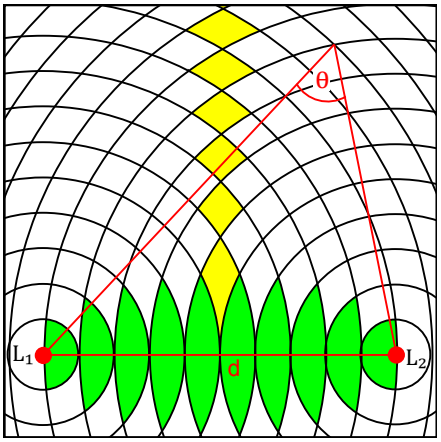


Abbildung 2: Die diskreten Ringe, welche durch die Abschwächstufen begrenzt werden, bestimmen für einen RFID-Controller den Ort, wo ein RFID-Tag lokalisiert werden kann. Die dazwischen liegenden, Rhombus-ähnlichen Flächen nennen wir Fehlerflächen. Die Grössen der Fehlerflächen hängen sowohl von der Position im Raum als auch von der Anordnung der Controller L_1 und L_2 ab.

(*Bayes filtering*). Ein Ausblick aufs weitere Vorgehen rundet den Bericht ab.

Das konkrete Problem

Ein Lagerraum wird in quadratische Zellen unterteilt. Die Seitenlänge von 80 Metern entspricht dem maximalen Empfangsbereich des ZOMOFI-Controllers (R_{MAX}). Mindestens zwei Ecken jeder Zelle werden mit je einem Access Point, einem ZOMOFI-Controller (siehe Abbildung 1), versehen. Wir nennen so eine Zelle *Lokalisierungszelle*. Ein Gabelstapler transportiert eine mit verschiedenen Haushaltsgeräten bestückte Palette (WG-Ladung) in irgendeinen freien Bereich des Lagers. Die WG-Ladungen dürfen freilich auch vertikal aufgestapelt werden. Die Aufgabe der Lokalisierungszelle besteht darin, sowohl die Position der WG-Ladung innerhalb der Zelle als auch deren Zusammensetzung zu bestimmen. Jede WG-Ladung wird mit einem globalen aktiven RFID-Tag identifiziert und ihre einzelnen Haushaltsgeräte zusätzlich mit individuellen RFID-Tags gekennzeichnet. Das Sammeln dieser Information wird von einem Server gesteuert, der via WLAN mit den ZOMOFI-Controllern verbunden ist.

Das ZOMOFI System

Das hier untersuchte kommerzielle ZOMOFI System benutzt RFID-Leser und aktive Tags, die mit einer Frequenz von 2,4 GHz miteinander kommunizieren. Die RFID-Controller können über Ethernet, WiFi oder serielle Schnittstelle (RS-485) gesteuert werden.

Ein RFID-Controller kann den Inhalt von maximal 80 Tags pro Sekunde analysieren und insgesamt etwa 1000 aktive RFID-Tags kontrollieren. Eine Besonderheit dieser RFID-Controller besteht darin, dass sie die Reichweite des empfangenen Signals stufenweise (in 32 Schritten, die via Applikation einfach programmierbar sind) kontrollieren können. Die maximale Reichweite beträgt etwa 80 Meter. Die aktiven RFID-Tags beinhalten in der Grundversion eine eindeutige ID, 112 Bytes benutzbaren Speicher, verschiedene programmierbare Funktionen wie z.B. *mute* und *beacon rate* und zudem eine Batterie, deren Lebensdauer etwa 8 Jahre beträgt. Spezielle Ausführungen der RFID-Tags können auch Bewegungs- bzw. Temperatursensoren beinhalten.

Lokalisierung von RFID-Tags

Ein aktives RFID-Tag, welches periodisch ein Radiosignal sendet, kann mit einem Peilgerät lokalisiert werden. Man müsste im Prinzip also nichts anderes tun, als ein RFID-Lesegerät so zu modifizieren, dass es durch kontrollierte Abschwächung der Empfangsempfindlichkeit die Schwelle der Empfangsreichweite des Signals des RFID-Tags bestimmt. Dadurch kann die Distanz zwischen RFID-Controller und RFID-Tag abgeschätzt werden (falls davon ausgegangen werden kann, dass die Ausbreitungsmerkmale einigermassen gut bekannt sind). Sobald jedoch mehrere Peilgeräte (in unserem Fall ZOMOFI-Controller) vorhanden sind, können die gewünschten (x,y)-Koordinaten mithilfe einfacher geometrischer Berechnungen ermittelt werden, wobei eine Unschärfe infolge der diskreten Abschwächung der Empfangsempfindlichkeit bleibt. Die der Unschärfe entsprechenden geometrischen Flächen nennen wir fortan Fehlerflächen (siehe Abbildung 2). Eine solche Fehlerfläche bestimmt schliesslich den Bereich, worin der RFID-Controller die gleichen (x,y)-Koordinaten für verschiedene Objekte misst. Die Grössen der Fehlerflächen variieren stark je nach Position im Raum und der Anordnung der Controller. Andere Eigenschaften von elektromagnetischen Systemen (z.B. Reflexionen, konstruktive und destruk-

tive Interferenz und Beugung der verschiedenen RF-Signale), welche zu weiteren Ungenauigkeiten führen, sollen erst im späteren Verlauf des Projektes mitberücksichtigt werden. Dabei stellen sich verschiedene Fragen: Zum Beispiel, wie viele Controller sind erforderlich und wie sollen sie im Raum angeordnet werden, damit die Ungenauigkeit bei der Bestimmung der Position minimiert werden kann?

Das Modell

Zur Beantwortung der zuvor aufgeworfenen Frage nach der Anzahl von Controllern und deren räumlichen Anordnung führen wir vorerst einige Analysen und Matlab-Simulationen durch. Das zugrundeliegende Modell ist nachfolgend detailliert beschrieben. Die Ergebnisse der Analysen und Simulationen sollen dann in einem späteren Projektschritt in einem 1:1-Experiment verifiziert werden.

Wir betrachten in 2D eine rechteckige Zelle (mit Seitenlängen B und H) vorerst mit zwei RFID-Controllern L_1 und L_2. L_1 und L_2 werden in zwei nebeneinander liegenden Ecken platziert (siehe Abbildung 1). Eine solche Anordnung ist derjenigen vorzuziehen, wo die beiden Controller in gegenüberliegenden Ecken platziert werden, weil insgesamt mehr unterschiedliche Fehlerflächen entstehen und diese somit im Mittel eine kleinere Ausdehnung, also eine kleinere Unsicherheit, haben. Dieser Anordnung legen wir ein kartesisches Koordinatensystem zu Grunde, so dass L_1 im Ursprung des Koordinatensystems liegt und L_2 an der Position $(B, 0)$. Nun ist hinlänglich bekannt (siehe z.B. [Pro95]), dass ein Radiosignal auf seinem Weg zum Empfänger durch räumliche Hindernisse, durch Beugung, Interferenz und Absorption in den Wänden abgeschwächt wird. Dadurch nimmt die Empfangsstärke Pr in der Praxis nicht nur quadratisch mit der Distanz R, sondern oft kubisch oder noch stärker ab:

$$Pr = K \cdot R^{-\lambda}$$

In dieser Formel ist R die Distanz zwischen Controller und Tag, λ der Dämpfungsfaktor ($2 \le \lambda \le 4$) und K eine Konstante, welche die Frequenz und den S/N-Faktor des ZOMOFI-Controllers subsumiert. Wenn K und λ für beide Controller bekannt sind, lassen sich aus den zwei Empfangsstärken Pr_j ($j \in \{1, 2\}$) die beiden Distanzen R_j abschätzen. Danach gilt es folgendes (überbestimmtes) Gleichungssystem zu lösen, um die (x,y)-Koordinate des Tags zu erhalten:

$$x^2 + y^2 = R_1^2$$
$$(B - x)^2 + y^2 = R_2^2$$
$$0 \le x \le B$$
$$0 \le y \le H$$

Wenn man bedenkt, dass in unseren Lokalisierungszellen etwa 850 Europaletten (1,2 m × 0,8 m) Platz haben und mehrere Lokalisierungszellen in einer Lagerhalle vorhanden sind, ist der Rechenaufwand für die Berechnung der (x,y)-Koordinaten recht hoch. Dazu treten noch verschiedene technische Schwierigkeiten auf, die eine analytische Lösung des Problems erschweren (siehe auch [BM02]).

Wir nutzen die Möglichkeit des ZOMOFI-Controllers, die Empfangsempfindlichkeit *schrittweise* zu verkleinern, um den Aufwand zur Bestimmung der Position in einem vernünftigen Rahmen zu halten. Die Reduktion der Empfangsempfindlichkeit wir durch einen Dämpfungsparameter n ($1 \le n \le 32$) erreicht. Die nichtlineare Abhängigkeit der Empfangsempfindlichkeit von der Distanz wird bereits im Controller berücksichtigt, so dass die resultierenden Radien R einfach als R_{MAX}/n angenommen werden können.

Mit der schrittweisen Reduktion handeln wir uns aber auch eine erhöhte Unsicherheit über die genaue RFID-Tag-Position ein. Zur Abschätzung der Distanzen R_j gehen wir von folgenden zwei stark vereinfachenden Annahmen aus: 1. Das Radiosignal breitet sich als Kugelwelle aus (was nur im freien Raum gültig ist) und 2. die Reichweite des Controllers bildet in der 2D-Ebene Kreise, deren Radius R linear mit dem programmierbaren Dämpfungsparameter n abnimmt. Mit diesen zwei Annahmen kann man nun wie folgt vorgehen:

- Die Reichweite von L_1 wird schrittweise reduziert bis zu demjenigen Wert n_1, unter dem das Tag nicht mehr identifizierbar ist. Dieser Schwellwert bestimmt die erste Abschätzung der Distanz R_1 zwischen L_1 und dem RFID-Tag: $R_1 = H \cdot n_1/32$.
- Man verfährt ähnlich mit L_2 und bestimmt dabei die Distanz R_2 zwischen L_2 und dem RFID-Tag: $R_2 = B \cdot n_2/32$.
- Die (x,y)-Koordinaten lassen sich wie folgt berechnen: $x = (R_1^2 - R_2^2 + B^2)/(2B)$; $y = \sqrt{(R_1^2 - x^2)}$.
- Diese Koordinaten (x,y) können eventuell als Ausgangspunkt für einen probabilistischen Algorithmus dienen, der die Koordinaten schärfer eingrenzen kann.

Analytische Lösung für zwei Controller

Die Fehlerfläche für zwei Controller gemäss Abbildung 2 können relativ einfach analytisch bestimmt werden. Dazu halten wir den Radius eines Controllers (L_1) konstant und denjenigen des anderen Controllers (L_2) variieren wir in Schritten von 2,5 m ($\Delta = R_{MAX}/32$). Die Berechnung der Linsenschnittfläche $A(a,b)$ zwischen zwei Kreisen mit Radien a und b ist in [Wei] angegeben. Durch einfache algebraische Manipulation erhalten wir folgende Formel für die Rhombus ähnliche Fehlerflächen:

$$A_x = \tfrac{1}{2}(A(R_1+\Delta, R_2+\Delta) + A(R_1, R_2) - A(R_1, R_2+\Delta) - A(R_1+\Delta, R_2))$$

wobei:

R_1: Radius des 1. Controllers

R_2: Radius des 2. Controllers

Δ: Inkrementschritt (2.5 m)

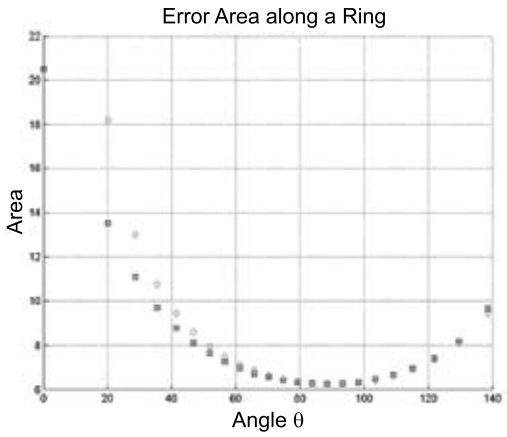


Abbildung 3: Die Fehlerfläche A_x (analytisch berechnet) als Funktion des Winkels θ für zwei Controller, angeordnet wie in Abbildung 2. R_1 = 30 m (konstant), R_2 = 52. 5 m bis 107.5 und Δ = 2.5 m. Der Abstand d zwischen den zwei Controllern beträgt 80 m. (□): Berechnet mit dem analytischen Modell. (x): Berechnet mit dem Programm [Vak]. (◊): Berechnet mit der Approximation von [NSB03]: $A_x = 4\varepsilon^2/\sin(\theta)$, wobei $\varepsilon = \Delta/2 = 1.25$ m. Der Winkel θ wird gemäss dem Kosinussatz für alle Messpunkte berechnet: $\theta = \arccos(d^2 - R_1^2 - R_2^2)/(2R_1R_2)$.

Die Abbildung 3 zeigt die Fehlerfläche A_x in Funktion des Zwischenwinkels θ (dargestellt in Abbildung 2) bei festgelegtem R_1 von 30 Metern.

Die Grösse und Varianz der Fehlerflächen A_x sind zudem abhängig vom Abstand d zwischen den beiden Controllern (siehe Abbildung 4). Ein grobes Mass des optimalen Abstands d kann somit aus dem Verlauf der Ableitungskurve $\partial A_x(R_1,R_2,d)/\partial d$ als Funktion von d bestimmt werden. Für den Fall, dargestellt in der Abbildung 3 (R_1,R_2= konstant), beträgt d_{opt} = 55 m. Dies gilt aber höchstens als Mass für die Unsicherheit einer einzelnen Messung. Man sollte eher den durchschnittlichen Messfehler über alle Fehlerflächen bestimmen. Dabei gilt, dass die Summe aller Fehlerflächen genau der Fläche der Lokalisierungszelle entspricht. Mit anderen Worten, je mehr Fehlerflächen eine Lokalisierungszelle enthält, desto kleiner wird der durchschnittliche Fehler. Bei der folgenden Konfiguration der beiden ZOMOFI-Controller (L_1 = (0,0) und L_2 = (80,0)) beträgt die durchschnittliche Fehlerfläche 7.4 m² bei einer Standardabweichung von σ = ± 1.9 m².

Falls die beiden Controller sich in zwei verschiedenen Ecken der Lokalisierungszelle befinden, d. h. deren Distanz d möglichst gross ist, so befinden sich die grössten Fehlerflächen entlang der Verbindungslinie zwischen den beiden Controllern (siehe Abbildung 4a). Daraus lässt sich ableiten, dass die beiden Controller auf der gleichen Seite der Lokalisierungszelle liegen sollen. Denn dadurch liegen die grossen Fehlerflächen entlang der Verbindungslinie nur zur Hälfte in der Lokalisierungszelle drin.

Lösung für drei Controller

Bei drei Controllern verzichten wir auf eine analytische Berechnung der Fehlerflächen und nutzen stattdessen numerische Simulationen. Mithilfe eines Matlab-Programms simulieren wir, wie sich die Fehlerflächen in Funktion der Position des RFID-Tags verhalten. Zur Berechnung der Fehlerflächen verwenden wir ein Programm, welches auf einem Kreisschnitt-Programm von A. Vakulenko [Vak] basiert. Um die Korrektheit unseres Programms abschätzen zu können, haben wir die Resultate mit den analytischen Berechnungen verglichen (siehe Abbildung 3).

Obwohl unser Programm erlaubt, dass die Controller beliebige Positionen innerhalb der Lokalisierungszelle annehmen, zeigen wir hier nur ein Beispiel unter den vielen möglichen Konfigurationen (siehe Abbildung 5). In dieser Konfiguration sind zwei Controller fest in je einer der unteren Ecken und der dritte in der Mitte der oberen Kan-

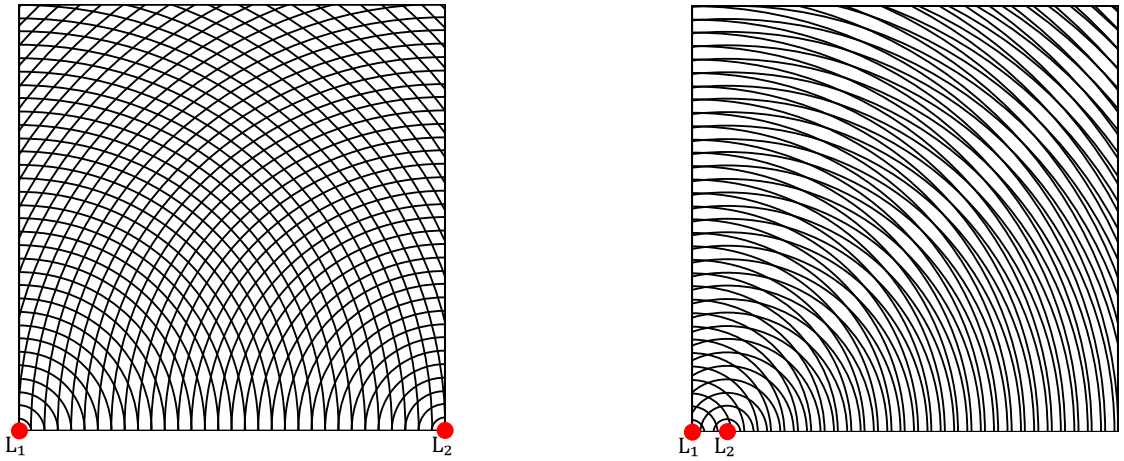


Abbildung 4: Die Grössen der Fehlerflächen hängen stark von der Distanz d zwischen den beiden Controllern ab. a) Bei einem grossen d sind die grössten Fehlerflächen nur entlang der direkten Verbindung der beiden Controller zu finden. b) Bei einem kleinen d treten Moirée-Effekte auf, welche mit wenigen grossen Fehlerflächen einhergehen.

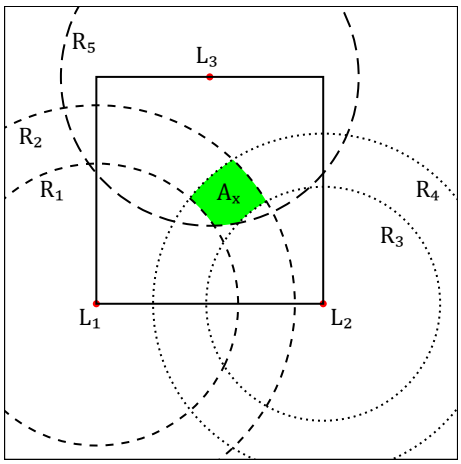


Abbildung 5: Bestimmung der Fehlerfläche A_x gemäss Variante a) bei drei Controllern L_1, L_2 und L_3.

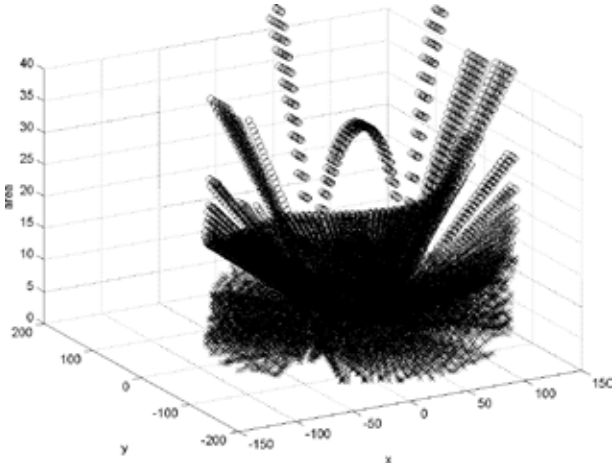


Abbildung 6: Grösse der Fehlerflächen (z-Achse) für eine fixe Stellung der Controller. (o): zwei Controller L_1 = (0,0) und L_2 = (80,0). (x): drei Controller L_1 = (0,0), L_2 = (80,0) und L_3 = (40,80).

te der Lokalisierungszelle verankert: L_1 = (0,0), L_2 = (B,0) und L_3 = (½B,H). Es gibt jeweils zwei Kreise (Signale), die von den Controllern L_1 und L_2 ausgehen. Die zwei Kreise links entsprechen den Signalen mit Radius R_1 und R_2; die zwei rechts den Signalen mit Radius R_3 und R_4. Da nur die Signale mit Radius R_2 und R_4 die WG-Ladung detektieren, kann sich diese nur innerhalb der Überlappungszone der zwei Kreise befinden. Wir können aber diese Unsicherheitszone beträchtlich einschränken, wenn wir auch die Information einbeziehen, dass die WG-Ladung nicht innerhalb der Kreise R_1 und R_3 liegt. Die Ladung kann sich somit nur dort befinden, wo sich die beiden Ringe, begrenzt durch die Kreise R_1 und R_2 bzw. R_3 und R_4, überlappen. Mithilfe des dritten Signals kann man nun bestimmen, ob die WG-Ladung in der oberen oder der unteren Überlappungsfläche liegt und in welchem Teilbereich.

Die Position der WG-Ladung wird somit wie folgt ermittelt:

1. Es wird für jeden der drei Controller bestimmt, welchen Radius das Signal haben muss, damit es möglichst knapp die WG-Ladung detektieren kann. Dies bestimmt die folgenden Werte: L_1: Radius R_2, L_2: Radius R_4 und L_3: Radius R_5.
2. Um die Fehlerfläche für die Position der WG-Ladung zu berechnen, brauchen wir noch für L_1 und L_2 die Werte der vorherigen Radii, innerhalb deren die WG-Ladung nicht detektiert werden kann: L_1: Radius $R_1 = R_2 - \Delta$ und L_2: Radius $R_3 = R_4 - \Delta$.

Das von uns verwendete Kreisschnitt-Programm berechnet die Flächen, wo sich mehrere Kreise überlappen, wobei die Anzahl der überlappenden Kreise frei wählbar ist. Wir benützen für unsere Berechnungen die Überlappungsflächen von jeweils zwei (A_{xy}) und drei (A_{xyz}) der fünf verschiedenen Kreise. Mit Hilfe solcher 2-Kreis- und 3-Kreis-Überlappungsflächen können die Unsicherheitszonen einfach berechnet werden. Je nach dem wie der

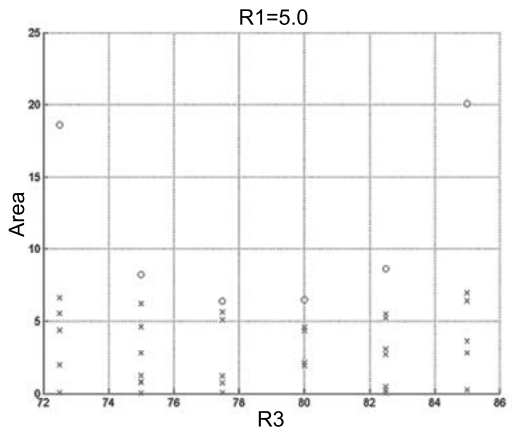


Abbildung 7: Grösse der Fehlerflächen in einem Vertikalschnitt der Abbildung 6. (o): zwei Controller. (x): drei Controller.

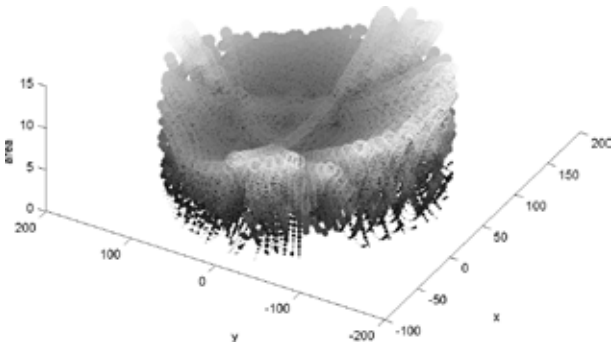


Abbildung 8: Grösse der Fehlerflächen für alle Punkte in einer mit drei Controllern (gleiche Anordnung wie in Abbildung 6). Helle Punkte: grosse Fehlerflächen; dunkle Punkte: kleine Fehlerflächen. Die eigentliche Lokalisierungszelle erstreckt sich von 0 bis +80 m auf beiden x,y-Achsen.

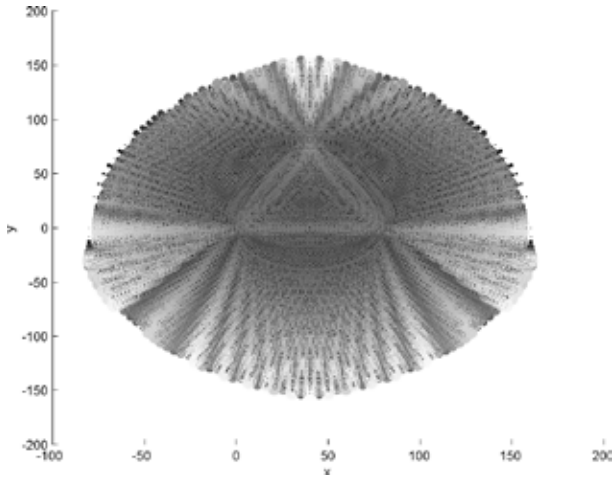


Abbildung 9: Parallelprojektion der Abbildung 8 (z=0). Helle Punkte: grosse Fehlerflächen; dunkle Punkte: kleine Fehlerflächen

Kreis mit Radius R_5 die anderen vier Kreise schneidet, wird eine der folgenden drei Formeln angewendet:

a) $A_x = A_{245} - A_{145} - A_{235} + A_{135}$

b) $A_x = A_{245} - A_{145} - A_{235} + A_{135} - ½(A_{24} - A_{14} - A_{23} + A_{13})$

c) $A_x = ½(A_{245} - A_{145} - A_{235} + A_{135})$

Resultate

Die Fehlerflächen, sind in der Abbildung 6 für zwei Konfigurationen von Controllern dargestellt, wobei die z-Achse die Grösse der Fehlerflächen in Quadratmeter angibt. Kreissymbole entsprechen den Fehlerflächen der Konfiguration mit zwei Controllern L_1 = (0,0) und L_2 = (B,0) und Kreuze denjenigen der Konfiguration mit drei Controllern L_1 = (0,0), L_2 = (B,0) und L_3 = (½B,H). Aus der Abbildung 6 geht eindeutig hervor, dass die Konfiguration mit drei Controllern wesentlich kleinere Fehlerflächen aufweist. Diese Tatsache wird noch einmal von der Abbildung 7 bestätigt, welche einen Vertikalschnitt aus Abbildung 6 (Schnitt senkrecht zur x-y-Ebene) darstellt. Die Abbildung 7 zeigt die Fehlerfläche in Funktion von R_3 mit R_1 als Parameter. Man beachte, dass R_2 und R_4 klar festgelegt sind (siehe Abbildung 5). Die Kreissymbole (o) liegen eindeutig höher als die Kreuze (x). Aus der Abbildung ist auch ersichtlich, dass die Werte von R_5 (Stapel von vertikalen Messpunkten (x) bei konstantem R_3) die Fehlerflächen vermindern.

In Abbildung 8 haben wir alle (x,y)-Koordinaten dargestellt, welche mit drei Controllern erreichbar sind. Der Grauton und die Grösse der Kreissymbole sind proportional zur Fehlerfläche, wobei ein heller Grauton eine grosse Fehlerfläche repräsentiert. Eine genaue Betrachtung der Grafik zeigt, dass die kritischen (grossen) Zonen entlang der Verbindungslinien zwischen den Controllern liegen und dass die Fehlerflächen sich im Bereich zwischen 0.01 m² und 13 m² bewegen. Eine Projektion dieser Grafik auf die x-y-Ebene ist in Abbildung 9 dargestellt. Sie zeigt die kritischen Zonen besonders klar an, wo die Fehlerflächen am grössten sind.

Die Frage nach der optimalen Anordnung der drei Controller in der Lokalisierungszelle ist damit natürlich noch nicht beantwortet. Aus der Analyse der Konfiguration mit zwei Controllern wissen wir, dass die grössten Fehlerflächen sich entlang der Verbindungslinie zwischen den Controllern befinden. Mithilfe des dritten Controllers können diese grossen Fehlerflächen jedoch auf knifflige Weise verkleinert werden, indem darauf geachtet wird, dass die Distanz vom dritten Controller zu seinen beiden Nachbarn nicht ein Vielfaches von Δ ist (siehe Abbildung 10).

Empirisch haben wir die folgenden vier Controller-Anordnungen grob untersucht. Die ersten drei Anordnungen basieren auf der optimalen Anordnung von zwei Controllern:

1. Konfiguration A: L_1 = (0,0); L_2 = (B,0); L_3 = (½B,H)
2. Konfiguration B: L_1 = (0,0); L_2 = (B,0); L_3 = (B,H)
3. Konfiguration C: L_1 = (0,0); L_2 = (B,0); L_3 = (¾B,H)
4. Konfiguration D: L_1 = (0,0); L_2 = (B,0); L_3 = (½B,½H)

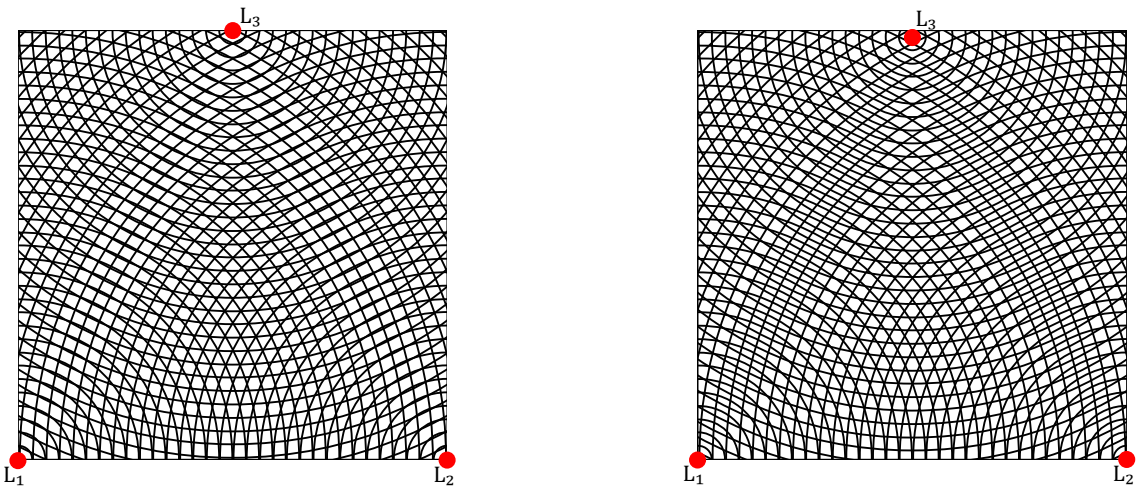


Abbildung 10: a) Konfiguration A mit drei Controllern: die grössten Fehlerflächen liegen in einem breiten Band entlang der direkten Verbindungen zwischen den Controllern. b) Der dritte Controller ist um ein halbes Δ in y-Richtung verschoben worden.

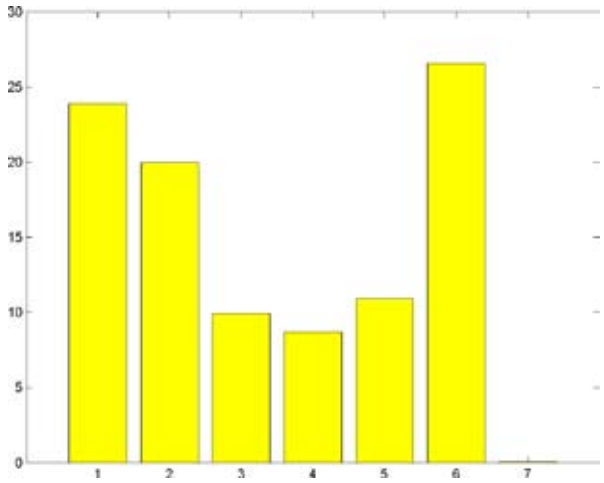


Abbildung 11: Verteilung der Fehlerflächen in Prozent in Funktion deren Mittelwertklasse (Mittelwert ± n·σ [n = 1,2,…,7]). Mittelwertklassen: Kategorie 1: Fehlerfläche < 0.6 m^2; Kategorie 2: 0.6 m^2 < Fehlerfläche < 1.7 m^2; Kategorie 3: 1.7 m^2 < Fehlerfläche < 2.7 m^2; Kategorie 4: 2.7 m^2 < Fehlerfläche < 3.8 m^2; Kategorie 5: 3.8 m^2 < Fehlerfläche < 4.8 m^2; Kategorie 6: 4.8 m^2 < Fehlerfläche < 6.9 m^2; Kategorie 7: Fehlerfläche > 6. 9 m^2.

Die Abbildung 11 zeigt die Verteilung der Fehlerflächen für die Konfiguration A, welche 2462 Fehlerflächen im Innern der Lokalisierungszelle beinhaltet und deren durchschnittliche Fläche 2.7 m^2 beträgt mit einem $\sigma = \pm$ 2.1 m^2.

Die Simulationen ergeben kein eindeutiges Resultat. Lediglich die Konfiguration B enthält etwa 20% mehr Positionen mit kleineren Fehlerflächen als die schlechteste Konfiguration D, was nicht verwunderlich ist in Anbetracht der Tatsache, dass mit kleinsten Positionsänderungen (z. B. ½ Δ) bereits sehr stark unterschiedliche Verteilungen der Fehlerflächen erzielt werden können.

Ausblick

Die kleineren Fehlerflächen, welche die Konfiguration A liefert, deuten auf eine mögliche Verbesserung des Lokalisierungsverfahrens hin:

1. Man permutiert die Reihenfolge, mit der die Controller L_i gestartet werden (z. B. P = (1,2,3), (2,3,1) oder (1,3,2)), um einerseits systematische Fehler der Geräte zu vermeiden und um anderseits eine bessere Bayes'sche Prior-Verteilung zu berechnen und bestimmt mit dem bereits besprochenen Verfahren für jede Permutation die entsprechenden (x_P, y_P) Koordinaten.
2. Die drei verschiedenen Koordinaten-Paare bilden den Ausgangspunkt eines *Bayesian Location Estimation* Algorithmus [AMGC02], um die durch Interferenz und Rauschen verursachten Fehler zu verringern.

Referenzen

[AMGC02] Arulampalam, S., Maskell, S., Gordon, N., Clapp, T. A Tutorial on ParticleFilters for On-line Non-linear/Non-Gaussian Bayesian Tracking. IEEE Trans. on Signal Processing, Vol. 50, No. 2, 174-188, 2002.

[BM02] Bergamo, P., Mazzini, G. Localization in Sensors Networks with Fading and Mobility. IEEE PIMRC, 750-754, 2002.

[MW06] Marzucco, G., Wyss, M. RFID Lagerschrankverwaltung. Diplomarbeit der FHNW, November 2006.

[NK02] Nicola, C.U., Koukolikova, Z. Stamping out fraud in luxury goods. 2002. http://lis.fh-aargau.ch/src/wbi.pdf

[NSB03] Nagpal R., Shrobe H., Bachrach J., Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network, 2nd International Workshop on Information Processing in Sensor Networks (IPSN ‹03), Palo Alto, April, 2003

[Pro95] Proakis, J. G. Digital Communications. McGraw-Hill Intern. Ed., 3. Auflage, 1995.

[Vak] Vakulenko, A. http://www.mathworks.com/matlabcentral/fileexchange

[Wei] Weisstein, E. W. Circle-Circle Intersection. http://mathworld.wolfram.com/Circle-CircleIntersection.html

HikeTracker

HikeTracker ist eine von uns entwickelte Navigations-Software für Smartphones. Sie dient Wanderern oder Bikern als Ersatz für gedruckte Karten und ermöglicht neben der Darstellung von Schweizer Landeskarten auch die Lokalisierung mittels GPS. Wir zeigen in diesem Artikel exemplarisch auf, wie eine Minimalvariante einer Navigations-Software aufgebaut sein kann und wie das Zusammenspiel zwischen quelloffenem Programmcode und lizenzrechtlich geschütztem Kartenmaterial aussehen kann. Zudem beleuchten wir die groben Unterschiede zwischen der J2ME- und der .NET-Variante des HikeTrackers.

Hans-Peter Oser, Christoph Stamm | hanspeter.oser@fhnw.ch

Die meisten heute in der Schweiz verfügbaren Navigationsgeräte (z.B. Blaupunkt TravelPilot, Falk, Garmin, TomTom usw.) haben alle einen gemeinsamen Mangel: Die verwendeten Karten haben selten die Qualität der offiziellen Schweizer Landeskarten. Dieser Mangel tritt vor allem dann in den Vordergrund, wenn man auf eine präzise und aktuelle Darstellung des Geländes angewiesen ist; dies gilt insbesondere für Bergsteiger, Wanderer und Biker, die sich in unübersichtlichem und kargem Gelände bewegen. Da wir in der Schweiz zum Glück über vorzügliches Kartenmaterial in den verschiedensten Massstäben (auf Landesebene: 1:25'000 bis 1:500'000) verfügen, liegt es nahe, dieses auch in einer Navigations-Software einsetzen zu wollen. Jedoch nur selten greifen kommerzielle Navigationssoftwares (z.B. FUGAWI oder GeoLives Mobile Swiss Edition) auf die Schweizer Landeskarten zurück.

Zu einer Navigations-Software gehören aber nicht nur die Geographie bzw. Topographie, sondern auch die Lokalisierung und eventuell sogar die automatische Routenplanung.

In unserem Produkt HikeTracker (siehe Abbildung 1) verwenden wir das Kartenmaterial der Swiss Map Serie der SwissTopo und ermöglichen die Lokalisierung mittels GPS, sofern ein solcher Empfänger vorhanden ist. Dabei spielt es keine Rolle, ob der GPS-Receiver als externes Gerät vorhanden ist und über Bluetooth angesprochen wird, oder ob er bereits im mobilen Gerät integriert ist. Auf die dritte Komponente einer Navigations-Software, die automatische Routenplanung, haben wir aus Gründen der Einfachheit verzichtet. Stattdessen sollen zuvor festgelegte oder zurückgelegte Wege angezeigt, gespeichert und eingelesen werden können. Zentral für uns ist, eine möglichst einfache, robuste und intuitiv bedienbare Minimalvariante einer Navigations-Software für Smartphones und PDAs exemplarisch zu entwickeln und frei zur Verfügung zu stellen. Dabei gilt es zu beachten, dass eine einfache Bedienbarkeit der Software im Gelände nur dann möglich ist, wenn der Bildschirm des mobilen Gerätes auch bei vollem Sonnenlicht gut lesbar bleibt. Dies ist jedoch nur bei wenigen Geräten wirklich gegeben, wie unterschiedlichste Tests gezeigt haben.

Abbildung 1: Windows Mobile Version des HikeTracker in Ausführung

Als öffentliche Institution liegt uns viel daran, Software nicht nur für den Hausgebrauch und in Kooperation mit Firmen zu entwickeln, sondern auch den Gedanken der frei verfügbaren – OpenSource – Software zu pflegen und somit unsere Entwicklungen der öffentlichen Hand zurückzuspielen. Da SwissTopo aber ein (berechtigtes) Interesse am Verkauf ihrer Kartenprodukte hat, sind die kommerziell am weitesten verbreiteten digitalen Produkte, die Swiss Map Serie (Swiss Map 25, 50 und 100), durch Verschlüsselung geschützt. Jedes Programm, welches nun die verschlüsselten Kartenbilder unverschlüsselt darstellen will, muss Zugriff auf das Entschlüsselungsprogramm erhalten. Da eine Offenlegung dieses Programms den Schutzanforderungen von SwissTopo jedoch widersprechen würde, sind der freien Verfügbarkeit des Quellcodes der Gesamtsoftware somit Grenzen gesetzt. Alle Programmteile ausser der Entschlüsselungskomponente sind Lesser GPL (LGPL) lizenziert.

In diesem Artikel gehen wir vor allem auf den Systementwurf und die Schnittstellen zwischen GPS-Empfänger und HikeTracker einerseits und zwischen Entschlüsselungskomponente und HikeTracker anderseits ein. Zudem betrachten wir die beiden aktuell verfügbaren Varianten des HikeTra-

ckers: Die ursprüngliche Variante des HikeTrackers wurde als Personal-Java-Applikation konzipiert. Zu einem späteren Zeitpunkt wurde dann der Java-Quellcode für J2ME angepasst (Midlet-Version) und zusätzlich nach C# portiert (Windows Mobile .NET Version).

Smartphone

Auf dem Markt gibt es eine Vielzahl von mobilen Geräten. Neben den weitverbreiteten mobilen Telefonen und PDAs ist eine Klasse von Geräten durch die Verschmelzung der beiden entstanden. Man spricht in diesem Zusammenhang von Smartphones. Die teureren Geräte verfügen heute meistens über eine Bildschirmauflösung von 320x240 Pixel (QVGA). Wie bereits angesprochen, gibt es nur wenige Geräte, deren Bildschirm auch bei Sonnenlicht lesbar bleibt. Zu diesen zählen die Geräte P990i und P1i von Sony Ericsson. Die Bildschirme vieler Geräte, z.B. der Firma HTC (Marktführer für Windows Mobile), sind bei Sonnenlicht nur ungenügend lesbar.

GPS-Receiver

Für den HikeTracker verwenden wir vorwiegend GPS-Receiver, die über Bluetooth mit dem Smartphone verbunden werden. Das hat den Vorteil, dass der GPS-Receiver sich in geringer Distanz zum Smartphone befinden darf. Somit ist auch eine Verwendung der Lokalisierung innerhalb eines geschlossenen Fahrzeuges möglich, indem man den GPS Receiver, um guten Empfang zu haben auf dem Armaturenbrett ablegt. Nachteilig ist es aber, dass die beiden Geräte miteinander verbunden werden müssen.

Gute Erfahrungen haben wir mit den Receivern von EMTAC [EMTAC] gemacht. Wir verwenden vorwiegend den EMTAC Trine, der gleichzeitig vier Verbindungen (zu Smartphones) unterstützt. Viele andere GPS Receiver sind erhältlich. Alle bieten das NMEA-0183-Protokoll [NMEA] an. Leider sind die Geräte nicht in jedem Fall austauschbar: oft ist das Protokoll nicht ganz vollständig oder oft variiert die Datenrate des Gerätes. Bei unerwartet hoher Datenrate können das Programm respektive die CPU eines einfachen Smartphones durch das Einlesen der Daten stark ausgelastet oder sogar überfordert sein.

Betriebssystem

Bei den Betriebssystemen der Smartphones ist man noch weit weg von jeder Standardisierung. Auch innerhalb einer Betriebssystemfamilie sind die Schnittstellen noch sehr starken Veränderungen von Version zu Version unterworfen. Das erstaunt aber nicht weiter, da Betriebssysteme für mobile Geräte nur den starken Wandel im Bereich der mobilen Geräte reflektieren. Das unveränderte Übertragen einer Anwendung von einem auf ein anderes Gerät, auch bei gleichem Betriebssystem, ist praktisch fast unmöglich. Immerhin ist nicht bei jedem Upgrade der Betriebssystemversion eine komplette Überarbeitung notwendig. Oft löst ein Upgrade des SDKs und ein anschliessendes *build* in der Entwicklungsumgebung das Problem.

Für Geräte mit einem Symbian Betriebssystem (z.B. UIQ3) bieten wir eine J2ME-Version (Midlet) des HikeTrackers an. Midlets laufen in einer SandBox-artigen Laufzeitumgebung ab und bieten deshalb ein hohes Mass an Sicherheit gegenüber unerlaubten Zugriffen. Die Kehrseite davon sind die rigorosen Beschränkungen der Zugriffsmöglichkeiten. Jeder Zugriff auf Ein- und Ausgabeschnittstellen wird erst dann zugelassen, wenn die Benutzerin diesen Zugriff erlaubt. Mittels eines signierten Zertifikats können Zugriffe auf gewisse Schnittstellen jedoch ein für allemal ermöglicht werden. Um diese Sicherheit wirklich gewähren zu können, muss natürlich auch der Zugriff auf die native Schnittstelle (Java Native Interface, JNI) unterbunden werden.

Auf den mobilen Windows-Geräten kommen die Versionen Windows Mobile V5.0 und V6.0 zum Einsatz. Die angesprochenen Inkompatibilitäten stammen vor allem aus der nicht freien Entschlüsselungskomponente der Software. Diese wird als Dynamic Link Library (DLL) erstellt und über JNI, wo vorhanden, ansonsten mithilfe eines Localhost-Daemons angesprochen. Ohne diesen nativen Teil würden die .NET Programme gut auf unterschiedlichen Betriebssystemversionen ablaufen. Die Symbian Betriebssysteme sind hingegen so aufgebaut, dass kein Programm ohne Anpassungen von einer auf die andere Version übernommen werden kann (zum Beispiel von Version UIQ v2.0 auf die Versionen UIQ v3.0 oder Series60: diese Versionen unterscheiden sich eben nicht nur im Look and Feel des GUIs sondern auch im Aufbau der Programmpakete). Es ist zu hoffen, dass diese Kompatibilitäts-Probleme in Zukunft durch den Einsatz von Android [And] als Smartphone-Betriebssystem behoben werden.

Systementwurf

Wie bereits angesprochen, soll unsere Software hauptsächlich auf die digitalen Pixelkarten von SwissTopo abgestimmt sein, konkret auf die kommerziell weit verbreiteten Produkte der Swiss Map Serie. Diese erlauben den Zugriff auf die Pixelkarten zu einem sehr günstigen Preis, jedoch mit der Einschränkung, dass die Daten nur in der Grösse eines A4-Ausschnittes jeweils decodiert werden können. Obwohl bei Swiss Map 25 alle Landeskarten des Massstabs 1:25'000 auf der DVD enthalten sind, können Sie nur jeweils ausschnittsweise eingesehen und allenfalls exportiert werden. Da Swiss Topo sich vom HikeTracker einen weiteren Kundenkreis verspricht, begrüsst SwissTopo den Einsatz ihrer Produkte im Zusammenhang mit mobilen Navigationsgeräten und verlangt neben dem

Kauf eines Swiss Map Produktes keine weiteren Lizenzkosten.

Bei der Installation eines Swiss Map Produktes auf einem Desktop-PC werden die verschlüsselten Kartendaten von einer Valentina-Datenbank [Val] verwaltet und ausschnittsweise entschlüsselt (siehe Abbildung 2). Ein Einsatz der gleichen Datenbank auf einem mobilen Gerät würde voraussetzen, dass die Valentina-Systemkomponente für mobile Betriebssysteme vorhanden wäre. Da dies nicht der Fall ist, muss eine alternative Lösung gewählt werden. Die Kartendaten müssen jedoch nach wie vor verschlüsselt abgelegt sein und die Entschlüsselungskomponente, welche auf dem mobilen Gerät verwendet wird, darf nicht offen gelegt sein und somit nicht Teil des OpenSource-Projektes sein.

Als Datenträger für die verschlüsselten Kartendaten werden Speicherkarten (z.B. Sony Memory Stick) eingesetzt, da die meisten Smartphones und PDAs über entsprechende Einschübe verfügen und die Speicherkarten die erforderliche Speichergrösse aufweisen. Die Datenübertragung zwischen der Valentina-Datenbank und den Speicherkarten muss in der Lage sein, die verschlüsselten Kartenbilder aus der Valentina-Datenbank zu extrahieren, und sie wiederum verschlüsselt auf die Speicherkarte zu kopieren. Auch hier gilt es die Lizenzrechte von SwissTopo zu berücksichtigen und daher wird hier auf den Quellcode und den Aufbau dieser Komponente nicht weiter eingegangen.

Die mobile Applikation mit grafischer Benutzungsschnittstelle hat primär die Aufgabe, die verschlüsselten Kartenbilder ausschnittsweise von der Speicherkarte einzulesen und darzustellen. Zusätzlich soll sie GPS-Daten von einem Receiver auslesen, interpretieren und die aktuelle Position im Kartenbild markieren.

Mobile Applikation

Ausgehend vom dargelegten Systementwurf lassen sich nun konkrete Anforderungen an die mobile Applikation auflisten und daraus mögliche Software-Designs ableiten. Um den Text nicht allzu sehr auszuweiten, beschränken wir uns hier auf dasjenige Design, welches wir auch wirklich umgesetzt haben.

Die wichtigsten Anforderungen sind:

- Datenbereitstellung: Die Kartendaten liegen verschlüsselt auf einer Speicherkarte. Die Entschlüsselung darf nicht mit einem OpenSource-Programm erfolgen.
- Interaktivität: die Anwendung wird über ein GUI gesteuert (Kartenausschnitt anpassen, Massstab verändern, GPS ein- und ausschalten). Der Bildschirm wird laufend aktualisiert.
- Lokalisierung: Vom GPS Gerät kommt ein sequentieller Datenstrom, der die aktuelle Position beschreibt. Dieser muss interpretiert werden.

Um die beiden Datenströme (Kartenbilder und GPS-Daten) quasi-gleichzeitig zu verarbeiten, bietet sich ein Ansatz basierend auf parallelen Threads an. Ein solcher Ansatz vereinfacht oft die Programmlogik, setzt aber voraus, dass die daraus entstehenden Synchronisationsprobleme professionell gehandhabt werden. Da die Smartphones nur über wenig Speicher und kleine Rechenleistung verfügen, ist die richtige Aufteilung der Aufgaben in möglichst wenige Threads und das Schaffen von geeigneten Schnittstellen zwischen den Threads sehr wichtig. Wird diese Aufteilung richtig durchgeführt, so können die gestellten Anforderungen auch mit einem Gerät mit einer bescheidenen CPU-Leistung erfüllt werden.

Zur Bestimmung der Anzahl Threads und der Aufgaben der einzelnen Threads haben wir mit MASCOT [MAS] gearbeitet. Damit haben wir die Struktur des Programms wie in Abbildung 3 gezeigt aufgebaut. Die Beschreibung der einzelnen Threads folgt weiter unten. Der Zugriff auf die verschlüsselten Kartendaten erfolgt über eine native Dynamic Link Library (DLL). In Java lassen sich DLLs über das Java Native Interface (JNI) ansprechen. Da in J2ME/Midlet jedoch kein JNI vorhan-

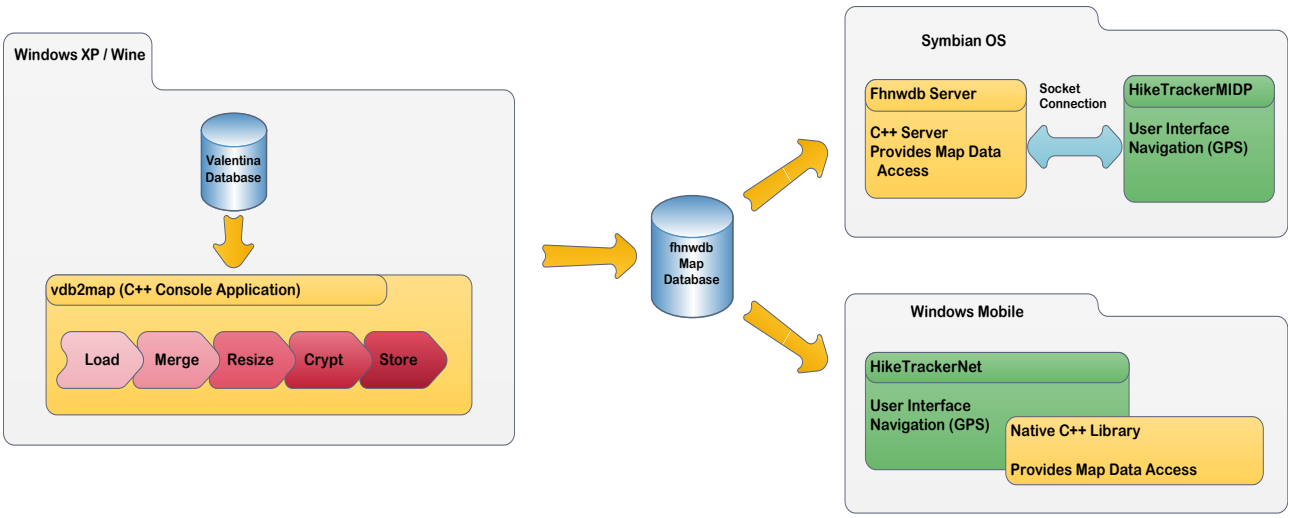


Abbildung 2: Systemaufbau der Windows Mobile und J2ME-Varianten

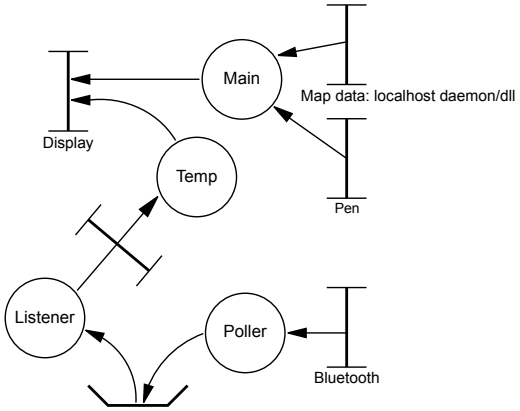


Abbildung 3: MASCOT ACP Diagramm

den ist, übernimmt ein localhost-Daemon [GdJ05] den Zugriff auf die Kartendaten. Unter Windows Mobile ist der Zugriff auf DLLs auch mittels .NET unproblematisch.

Der *Poller-Thread* ist dafür verantwortlich, dass die Daten vom GPS-Receiver richtig gelesen und interpretiert werden. Um nicht unnötig viel CPU-Zeit zu beanspruchen, liest er nur einmal pro Intervall (adaptiv zwischen 1 und 5 Sekunden) die Koordinaten vom GPS und legt sich dann wieder schlafen. Die erhaltenen Positionsdaten werden zwischengespeichert und dadurch den andern Threads zur Verfügung gestellt. Leider funktioniert dieses Prinzip nicht in jedem Fall für spezielle GPS-Geräte, weil diese so viele Daten senden, dass der Empfangs-Puffer im Smartphone überläuft, wenn er nicht öfter geleert wird. Deshalb leert der Poller-Thread den Puffer wenn er gerade keine neuen Koordinaten empfangen will.

Der *Listener-Thread* überprüft nach jedem Intervall die vom Poller-Thread erhaltene Position und löst entsprechende Ereignisse bei seinen Clients aus. Hat das GPS eine gültige Position gemeldet und ist diese auch aktuell (nicht älter als einige Sekunden) wird bei den Listeners ein Update ausgeführt. Ist die Position jedoch ungültig (z. B. weil das GPS zu wenige Satelliten in Reichweite hat), wird dies den Listeners mit einer Statusänderung mitgeteilt.

Der *Draw-Thread* wird jeweils vom Listener-Thread gestartet und frischt nur das Bild auf dem Gerätemonitor auf.

Das UML-Sequenzdiagramm in Abbildung 4 stellt die einzelnen Threads des MASCOT Diagramms im zeitlichen Zusammenspiel dar.

Schnittstellen

Nachdem wir den generellen Aufbau der Applikation aufgezeigt und die Wirkbereiche der einzelnen Threads kurz umrissen haben, verbleibt noch die Besprechung der drei Schnittstellen: die Graphische Benutzungsschnittstelle (GUI), der Zugang zu den GPS-Daten und der Zugang zu den Kartendaten.

In Abbildung 1 ist der Aufbau des Bildschirms dargestellt. Auf der ersten Zeile ist ersichtlich, dass man zu fünf Satelliten eine Verbindung hat und sich auf einer Höhe von 327 Meter über Meer befindet. Am unteren Rand des Bildschirms be-

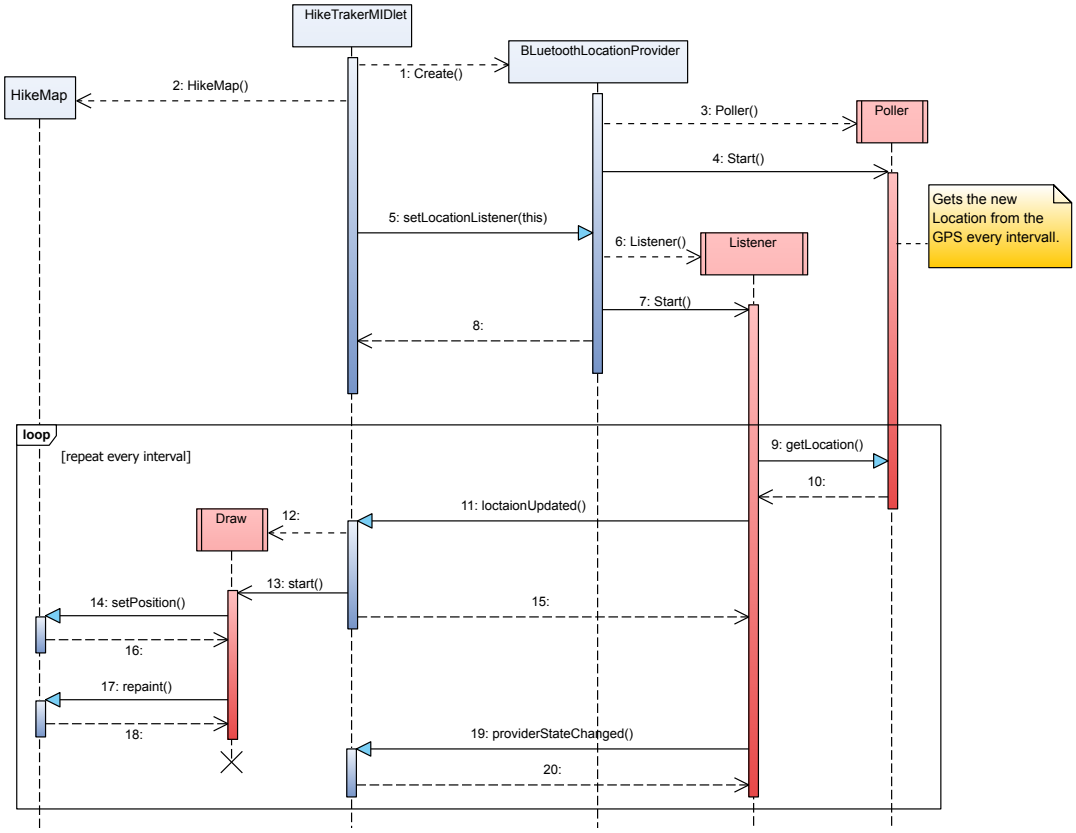


Abbildung 4: UML-Sequenzdiagramm

findet sich eine Menüliste: Mit *Maps* können die verschiedenen Kartenmassstäbe ausgewählt und mit *Menu* die nachfolgenden Optionen eingestellt werden.

- *goto:* Positioniert die Karte auf eine Ortschaft (in der verfügbaren Karte).
- *Road dots:* Damit kann ein zurückgelegter Weg aufgezeichnet, abgespeichert, wieder geladen und gelöscht werden.
- *Zoom:* Verändert den Kartenmassstab. Die aktuelle Kartenposition erhalten.
- *Status:* Ein- und Ausblenden von Statusinformationen (z. B. Anzahl verfügbarer Satelliten, Höhe über Meer).
- *GPS centered:* Wenn diese Option eingeschaltet und ein GPS Receiver vorhanden ist, wird die aktuelle Position immer in der Bildschirmmitte angezeigt. Diese Option kann ausgeschaltet werden wenn ein anderer Kartenausschnitt betrachtet werden möchte.
- *Position:* Beim Start des Programms zeigt das Programm den zuletzt gezeigten Kartenausschnitt (vorhergehender Programmstart).

Das GPS-Gerät sendet alle Daten in Form von speziell codierten Zeilen. Das Format dieser Zeilen wird von der National Marine Electronics Association (NMEA[1]) spezifiziert. Jede der Zeilen beginnt mit einem Codewort, das den Typ der Information angibt, die in den restlichen Feldern der Zeile stehen. Typische Codewörter sind z. B.: $GPRMC (empfohlener Minimumsdatensatz), $GPGGA (wichtigsten Informationen zur GPS-Position und Genauigkeit), $GPGSA (aktive Satelliten). Die einzelnen Werte sind durch Kommata abgetrennt. Für weitere Details des NMEA-Protokolls ist auf [NMEA] verwiesen.

Für den HikeTracker sind vor allem die Zeilen mit dem $GPGGA-Code von Interesse [G007, JR07]. Sie sind wie folgt aufgebaut:

```
0     |1         |2         |3|4         |
$GPGGA,092941.973,4729.1104,N,00811.9014,
$GPGGA,123519    ,4807.038 ,N,01131.000 ,

|5|6|7 |8  |9    |A|B   |C|D  |E   |F |
,E,1,06,2.7,420.3,M,48.0,M,0.0,0000*7
,E,1,08,0.9,545.4,M,46.9,M,   ,    *47
```

Feld	Bedeutung	Format oder Wert
0	Codewort	z. B. $GPGGA
1	Zeit	hhmmss.xx
2	Breitengrad in Dezimalgrad	HHMM.xx
3	Hemisphäre	N = Nord, S = Süd
4	Längengrad in Dezimalgrad	HHMM.xx
5	östliche oder westliche Länge	E = Ost, W = West
6	Qualität	0 = ungültig, 1 = GPS, 2 = DGPS
7	Anzahl Satelliten	
8	horizontale Abschwächung (dilution)	
9	Höhe über Meer (über Geoid)	
A	Masseinheit der Höhe über Meer	M = Meter
B	Höhe Geoid minus Höhe Ellipsoid	Ellipsoid gemäss WGS84
C	Masseinheit der geoidalen Höhe	M = Meter
D	Alter der DGPS-Daten	Sekunden
E	ID der DGPS-Referenzstation	
F	Checksumme	

Sobald eine Zeile gefunden wird, die mit dem richtigen Codewort beginnt, wird der Rest der Zeile anhand der Kommata in die einzelnen Felder aufgetrennt. Mittels der Checksumme können gröbste Fehler entdeckt werden (CheckSum = XOR der Bytes zwischen $ und *, ohne $ und *).

Im Falle des HikeTrackers interessieren uns nur die Felder 2 bis 5. Feld 6 muss in unserem Fall immer 1 sein. Um die Aktualität der Positionsdaten zu überprüfen, wird die Systemzeit des Smartphones zum Zeitpunkt des Einlesens verwendet. Dadurch kann auf die Zeitangabe im Feld 1 verzichtet werden. Der Systemzeit des Smartphones wird der Vorzug gegeben, weil man nicht davon ausgehen möchte, dass das Smartphone über eine korrekt eingestellte Uhrzeit verfügt, was aber für einen direkten Vergleich der GPS-Datenzeit mit der Systemzeit der Fall sein müsste.

Wie bereits erwähnt, liegen die Kartendaten verschlüsselt auf einer Speicherkarte. Die Entschlüsselung der Kartenbilder übernimmt entweder ein Localhost-Daemon (J2ME) oder eine unmanaged DLL (.NET). Beide werden über das nachfolgende Schnittstellenprotokoll angesprochen.

List DBs
Anfrage: l
Antwort: <length><n><name_1><name_2>…<name_n>
length: Länge der Antwort in Bytes (ohne die 4 Bytes von length)
n: Anzahl der vorhandenen Kartendatenbanken (1 Byte)
name_i: null-terminierter Namen der i-ten Datenbank

Tile Size
Anfrage: m
Antwort: <size>
size: quadratische Kachelgrösse in Metern (4 Bytes)

Get Tile
Anfrage: g<dbID><tileID>
dbID: nullindizierte, eindeutige Datenbanknummer
tileID: Nummer der gewünschten Kachel
Antwort: <length><tile>
length: Anzahl Bytes der Kacheldaten, falls positiv (4 Bytes)
tile: Kacheldaten in Portable Network Graphics Format (PNG)

Kill Server
Anfrage: k
Antwort: 0 (1 Byte)

Um bei der Midlet-Version den Programmstart zu vereinfachen, d.h. zu verhindern, dass der Localhost-Daemon vorgängig gestartet werden muss, verwendet das Midlet das so genannte *PushRegistry*. Dabei wird das Midlet so installiert, dass

1 NMEA: www.nmea.org

dieses über eine Meldung auf einem definierten Socket gestartet werden kann. Der Start erfolgt durch den Localhost-Daemon, welcher dann über eine Socket-Meldung das Midlet startet. Beim Beenden des Programms wird der Localhost-Daemon vom Midlet abgeschlossen.

Stand der Entwicklung und Ausblick

Die erste Version des HikeTrackers wurde 2004 [RW04] für das P910i von SonyEricsson entwickelt und in den Arbeiten [MM05] und [CT06] schrittweise verbessert. Dieses Programm ist in Personal-Java erstellt worden und verwendet die Kartendaten von SwissMap 50 Version 2.0 (verfügbare Massstäbe 1:50'000, 1:100'000, 1:200'000 und 1:400'000). Da neuere Smartphones kaum noch Personal-Java-Programme unterstützen, haben wir uns entschlossen, diese Basisvarianten nicht weiter zu entwickeln.

Die Entwicklung der beiden neuen Versionen des HikeTrackers (Midlet und .NET) begann im Jahr 2006 als vollständige Neuentwicklung und wurde wiederum in verschiedenen Arbeiten [CT06, G007] weiterentwickelt. Sie verwenden die Kartendaten von SwissMap 25 und SwissMap 50 Version 3. Mit der Arbeit [JR07] wurde schliesslich das neue in diesem Artikel beschriebene Konzept realisiert. Einzig die Verschlüsselung der Kartendaten ist im Augenblick noch nicht vollständig implementiert. Daher können die aktuellen Programme noch nicht der Öffentlichkeit zur Verfügung gestellt werden. Zudem ist der Funktionsumfang noch etwas geringer als bei der Personal-Java-Version. Bei der Funktion Road dots wird man dafür in Zukunft auch Routen aus anderen Programmen importieren und visualisieren können.

Referenzen

[And] Android, Open Handset Alliance Project. http://code.google.com/android/

[CT06] Casoni, Thalmann, HikeTracker. Diplomarbeit Studiengang Informatik, Fachhochschule Nordwestschweiz FHNW, 2006. http://www.hiketracker.org/downloads/HikeTracker_06.pdf

[EMTAC] EMTAC Trine: GPS Bluetooth Receiver einer Firma aus Taiwan. Das Gerät kann bei verschiedenen Firmen gekauft werden, eine Homepage der Firma EMTAC gibt es nicht mehr.

[G007] Gruntz, D., Olloz, Erfahrungen mit dem Location API (JSR 179). JavaSPEKTRUM 06, 2007. http://www.fhnw.ch/technik/imvs/publikationen/artikel-2007

[GdJ05] Gupta, A., de Jode, M., Extending the Reach of MIDlets: how MIDlets can access native services. Symbian Developer Network, 2005. http://www.arvindgupta.net/pdf/The_Bridge.pdf

[JR07] Jean-Richard, M., HikeTracker. Diplomarbeit Studiengang Informatik, Fachhochschule Nordwestschweiz FHNW, 2007. http://www.hiketracker.org

[MAS] MASCOT. Modular Approach to Software Construction Operation and Test http://en.wikipedia.org/wiki/

[MM05] Meier, Mühlebach, InhouseTracker. Diplomarbeit Studiengang Informatik, Fachhochschule Aargau, 2005. http://www.hiketracker.org/downloads/InhouseTracker_05.pdf

[NMEA] NMEA-0183 Datensätze http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm

[RW04] Rosenberg, Wichtermann, HikeTracker. Diplomarbeit Studiengang Informatik, Fachhochschule Aargau, 2004. http://www.hiketracker.org/downloads/HikeTracker_04.pdf

[Val] Valentina Database http://www.paradigmasoft.com/

Lancierung und Verwaltung von Open Source Projekten

Open Source Projekte sind ein wichtiger Beitrag öffentlicher Forschungseinrichtungen und der Privatwirtschaft zur allgemeinen Weiterentwicklung der Informatik. Sobald die Software der Öffentlichkeit zur Verfügung gestellt wird, stellen sich Fragen der passenden Lizenzierung und der Mitwirkung von externen Entwicklern. Aus diesem Grund stellen wir in diesem Artikel die wichtigsten Open Source Lizenzen vor und zeigen in einem Fallbeispiel wie häufig benutzte Kollaborationswerkzeuge in der Open Source Software-Entwicklung gewinnbringend eingesetzt werden können.

Andreas Hofmann, Matthias Krebs, Christoph Stamm, Ursula Sury[1] | christoph.stamm@fhnw.ch

Der Begriff Open Source ist entstanden, um sich des Begriffes Freie Software zu entledigen. *Freie Software* und Open Source unterscheiden sich in der eigentlichen Bedeutung nicht wesentlich. Die Entscheidung, den Terminus Open Source zu etablieren begründete sich zum Teil auf der möglichen Missinterpretation des Wortes frei. Die *Free Software Foundation (FSF)* [FSF] verstand das Wort im Sinne von Freiheit («free speech, not free beer»), jedoch wurde es oft fälschlicherweise mit kostenlos assoziiert, da der englische Begriff *free* beide Bedeutungen haben kann.

Bruce Perens und Tim O'Reilly beschlossen, dass die Freie Software-Gemeinde mehr und besseres Marketing benötigt, um diese Art von Software weniger ideologisch zu bewerten. Der Begriff Open Source wurde von da an flächendeckend im Marketing genutzt und war auch der Namensgeber für die von Raymond, Perens und O'Reilly gegründete *Open Source Initiative (OSI)* [OSI].

Die OSI wendet den Begriff Open Source auf Software an, deren Lizenzverträge den folgenden drei charakteristischen Merkmalen[2] entsprechen:

- Der Programmcode liegt in für Menschen lesbarer und verständlicher Form vor. In der Regel handelt es sich bei dieser Form um Quelltexte in einer höheren Programmiersprache. Vor dem eigentlichen Programmablauf ist es normalerweise notwendig, diesen Text durch einen so genannten Compiler in eine binäre Form (*Binärprogramm*) zu bringen, damit das Computerprogramm vom Rechner ausgeführt werden kann. Binärprogramme sind für den Menschen im semantischen Sinne praktisch nicht lesbar.
- Die Software darf beliebig kopiert, verbreitet und genutzt werden. Für Open Source Software gibt es keine Nutzungsbeschränkungen. Weder bezüglich der Anzahl der Benutzer, noch bezüglich der Anzahl der Installationen. Mit der Vervielfältigung und der Verbreitung von Open Source Software sind auch keine Zahlungsverpflichtungen gegen eine Lizenzgeberin verbunden.
- Die Software darf verändert und in der veränderten Form weitergegeben werden. Durch den offen gelegten Quelltext ist Verändern ohne weiteren Aufwand für jedermann möglich. Weitergabe der Software soll ohne Lizenzgebühren möglich sein.

Gerade der dritte Punkt zeigt deutlich, dass Open Source Software förmlich von der aktiven Beteiligung der Anwender an der Entwicklung lebt. So bietet sich Open Source Software zum Lernen, Mitmachen und Verbessern an. In den Abschnitten *Lancierung eines Open Source Projektes* und *Fallbeispiel* zeigen wir zuerst allgemein und dann anhand eines konkreten Beispiels von einer Programmbibliothek zur Bildkomprimierung auf, welche Schritte notwendig sind, um selber ein Open Source Projekt zu lancieren und zu verwalten.

Trotz den drei genannten charakteristischen Merkmalen scheint der Begriff Open Source nicht frei von Interpretationen zu sein, da selbst die FSF und die OSI sich nicht immer einig darüber sind, ob die Nutzungsrechte einer Software-Lizenz den von FSF bzw. OSI verlangten Kriterien von Open Source genügen. Der Begriff der Software-Lizenz und die Anwendung einer solchen sind hier sehr zentral und werden daher im nachfolgenden Abschnitt genauer dargestellt.

Software-Lizenzen

Eine Lizenz ist ein Gebrauchsrecht für etwas Nichtmaterielles. Im Falle einer Software-Lizenz wird die Art und Weise des erlaubten Gebrauchs der Software klar geregelt. Software-Lizenzen treten nicht nur im Open Source Umfeld auf, son-

1 Ursula Sury ist selbständige Rechtsanwältin und Dozentin an der Hochschule Luzern.

2 Diese drei Charakteristika sind in der Open Source Definition der Open Source Initiative detailliert festgelegt.

dern immer bei der Benutzung von Software. Wer Software einsetzt, akzeptiert global – entweder stillschweigend oder ausdrücklich durch Bestätigung die Lizenzvereinbarungen gelesen zu haben – die allgemeinen Geschäftsbedingungen zwischen Lizenzgeber (z. B. Software-Hersteller) und Lizenznehmer (Benutzerin). Da die Benutzer in den meisten Fällen blind vertrauen, sind die Benutzer gegen abstruse Geschäftsbedingungen grundsätzlich geschützt.

Mehr als blindes Vertrauen ist bei der Lancierung eines eigenen Open Source Projektes angebracht. Denn für das eigene Projekt sollte eine passende Software-Lizenz ausgewählt werden und dies setzt wiederum voraus, dass verschiedene Lizenzen bekannt sind. Aus diesem Grund listen wir in Abschnitt *Verschiedene Open Source Lizenzen* ein paar wichtige Open Source Software-Lizenzen auf und stellen sie in Abschnitt *Auswahl einer passenden Lizenz* einander gegenüber. Doch zuerst erklären wir die Begriffe Lizenz, Copyright und Copyleft.

Bei all diesen Ausführungen betonen wir, dass wir die Sachlage aus der Sicht von Informatikern betrachten und uns für einfachen Umgang mit Wissen einsetzen. Des weitern ist wichtig zu erwähnen, dass beim weltweiten Austausch von Software verschiedene Rechtssysteme aufeinander prallen. Lizenzbedingungen werden meistens vor dem Hintergrund des Rechtssystems desjenigen Staates formuliert, in dem die Software entwickelt wird. Die Anwendung dieser Lizenzbedingungen in einem fremden Rechtsystems ist dann eine ganz andere Sache. Bei unseren Betrachtungen gehen wir vom Schweizer Recht [CHLR] aus, wo nichts Gegenteiliges erwähnt ist, welches im Bereich des Urheberrechtes weltweit ähnlich ist.

Lizenz

Treu der Open Source Philosophie zitieren wir Wikipedia [Wiki] zum Begriff Lizenz, da der ganze Inhalt von Wikipedia unter der *GNU*-Lizenz für freie Dokumentation [GNU] steht, die das Kopieren von Texten für jegwelche Zwecke ausdrücklich erlaubt:

> *Allgemein ist eine Lizenz (v. lat.: licere = erlauben) eine Erlaubnis, Dinge zu tun, die ohne diese verboten sind. Dies können einerseits staatlich erteilte Sonderrechte sein, zum Beispiel in der ehemaligen DDR die Spielerlaubnis für Musiker oder die «Lizenz zum Töten» des fiktiven Agenten James Bond, andererseits, bei gewerblichen Schutzrechten wie dem Urheber- oder Patentrecht, von juristischen oder natürlichen Personen mittels Verträgen eingeräumte Rechte.*

Es geht also um eine Erlaubnis, um ein Recht, im Fall von Software-Lizenzen um ein Recht auf Verwendung von Software (ähnlich der Miete eines Produkts). Werden Computerprogramme in einem Arbeitsverhältnis in Erfüllung vertraglicher Pflichten geschaffen, so ist die Arbeitgeberin allein zur Ausübung der ausschliesslichen Verwendungsbefugnisse berechtigt. Ansonsten besitzen die Urheber das ausschliessliche Recht auf Gebrauch. Generell wird hier zwischen einfachen und ausschliesslichen Rechten unterschieden. Wenn eine Urheberin dem ausschliesslichen Gebrauch ihres Werkes durch jemand anderen zustimmt und sich keinerlei Gebrauchsrechte vorbehält, so kann der neue Rechtsbesitzer unter Ausschluss aller, einschliesslich der Urheberin selber, das Werk exklusiv nutzen. An der Urheberschaft ändert sich aber nichts. Man spricht hier von einem ausschliesslichen Recht. Der Besitzer eines solchen Rechtes kann einfache Gebrauchsrechte an Dritte vergeben, wobei die Urheberin damit einverstanden sein muss. Der Besitzer eines einfachen Gebrauchsrechtes ist nur dazu berechtigt, das Werk auf die vertraglich festgelegte Weise zu gebrauchen.

Bei freier Software werden einfache Gebrauchsrechte pauschal an jedermann eingeräumt. Die Zustimmung des Lizenznehmers wird üblicherweise stillschweigend durch sein korrektes Verhalten in der Anwendung der Lizenz signalisiert; es muss nichts unterschrieben werden. Insofern der Rechteinhaber keine Gegenleistungen fordert, kann die Lizenz kurz und simpel ausfallen. Ein bekanntes Beispiel lautet: «Do the fuck you want with it.» – ein wenig formlos, doch juristisch gültig. Problematisch ist jedoch bei solchen Lizenzen, dass veränderte Versionen der Computerprogramme nach geltendem Recht nicht automatisch ebenso freigiebig an jedermann lizenziert sind. So geniessen so genannte Werke zweiter Hand ihren eigenständigen Schutz, so lange nichts anderes vereinbart worden ist. Die *GNU*[3] *General Public License* [GPL] zum Beispiel versucht daher die Freiheiten zu bewahren und verlangt eine Gegenleistung für die eingeräumten Rechte:

- Dass das Programm nicht allein als Binärprogramm, sondern nur zusammen mit einer für Menschen verständlichen Version, dem Quelltext, weitergegeben wird.
- Dass veränderte Versionen nur dann verbreitet werden dürfen, wenn sie ebenfalls unter die GPL gestellt werden. Wer sich nicht daran hält, verliert seine eigenen Gebrauchsrechte wieder.

Diese Verfahrensweise, eine Freiheit zu bewahren, wird in Anlehnung an den Begriff Copyright Copyleft genannt.

Copyright

Das Copyright stammt aus dem angloamerikanischen Raum und ist dem Schweizerischen Urheberrecht [URG] ähnlich. Es unterscheidet

3 GNU ist ein rekursives Akronym von GNU's not Unix.

sich aber in der Ausrichtung insofern, dass das Copyright eher die wirtschaftlichen Aspekte (den Schutz von wirtschaftlichen Investitionen), hingegen das Urheberrecht in der Schweiz mehr die Urheberin eines Werkes als Schöpferin und deren ideellen Beziehungen zum Werk berücksichtigt. Das angloamerikanische Copyright wird nicht automatisch der Urheberin eines Werkes zugestanden, sondern es kann einem wirtschaftlichen Rechteverwerter (z. B. Verlag) übertragen werden. Im Gegensatz dazu wird das Schweizerische Urheberrecht der Schöpferin ohne weiteres Zutun automatisch zugesprochen. Die Urheberin ist jedoch frei ihr Urheberrecht an eine andere natürliche Person zu übertragen. Damit ein Werk urheberrechtlich geschützt sein kann, ist eine geistige Schöpfung mit individuellem Charakter notwendig. Zudem muss es eine gewisse Schöpfungshöhe aufweisen; triviale Werke fallen somit nicht unter das Urheberrecht. Computerprogramme sind im URG explizit als Werke aufgelistet.

Copyleft

Für Fortentwicklungen gilt wie für Erstentwicklungen, dass unter Einbezug der Schöpfungshöhe ein eigenständiges Urheberrecht besteht. Ist das ursprüngliche Werk zum Beispiel für jeden frei kopierbar, verteilbar, veränderbar usw., so übertragen sich diese Freiheiten jedoch nicht automatisch auf die Fortentwicklungen des Werks. Dadurch kann freie Software, welche nicht speziell lizenziert wird, zum Ausgangsmaterial künstlich knapper *proprietärer Software*[4] werden. Da dies oft nicht erwünscht ist, beinhalten gewisse Lizenzen eine Copyleft-Klausel.

Gemäss Wikipedia ist Copyleft ein Schutzverfahren in bestimmten Lizenzen freier Software beziehungsweise freier Inhalte (z. B. Software), welches einen bestimmten Aspekt des Copyrights (beziehungsweise Urheberrechts) in sein Gegenteil zu verkehren versucht; daher auch der Name. Das heisst, mit einer Copyleft-Vereinbarung wird die Freiheit von Weiterbearbeitungen und Fortentwicklungen eines freien Werkes erzwungen, um dadurch die Vereinnahmung zu verhindern. So erlaubt die Urheberin einer Software beispielsweise die freie Verwendung und die Weiterentwicklung ihres Quelltextes nur dann, wenn der Urheber der Fortentwicklung sich damit einverstanden erklärt, seine Weiterentwicklung unter den gleichen Bedingungen wie das ursprüngliche Werk frei zu geben.

Nicht alle Entwickler freier Software sind jedoch mit den gängigen Copyleft-Vereinbarungen einverstanden. Durch den Zwang, Fortentwicklungen ebenfalls unter die gleiche Lizenz zu stellen, sehen sie die Freiheit unangemessen eingeschränkt und raten stattdessen zu Copyleft-freien Lizenzen.

Eine Copyleft-Klausel ist oft auch sinnvoll, weil sich die Urheberrechtsbestimmungen zwischen Werken zweiter Hand und Werken mit mehreren Miturhebern dahingehend unterscheiden, dass im Fall eines Werks zweiter Hand, die Urheberin des bestehenden Werks kein Mitsprache bei der Nutzung des Werks zweiter Hand erhält, dass aber bei Werken mit Miturhebern allen gemeinsam die Urheberschaft und somit ein Mitspracherecht zusteht.

Problematisch beim Copyleft ist, dass zwei verschiedene Copyleft-Lizenzen oft miteinander inkompatibel sind. Das heisst, es können keine zwei Werke mit verschiedenen Copyleft-Lizenzen zu einem einzigen kombiniert werden. Aufgrund der grossen Verbreitung der GPL in Zusammenhang mit Computerprogrammen ist der Fall zweier unterschiedlicher Copyleft-Lizenzen jedoch eher selten. Für freie Literatur, freie Musik usw. lauert hier aber eine grosse Gefahr. Solche Inkompatibilitäten können nur dann aufgelöst werden, wenn die Lizenzen selbst erlauben, dass das Werk unter eine andere Lizenz gestellt werden darf (z. B. GPL bei der LGPL Version 2.1) oder sich selbst als Ausnahmen einer bestehenden Lizenz definieren (z. B. LGPL Version 3 als Ausnahme der GPLv3). Die bekanntesten Lizenzmodelle, die die Open Source Bedingungen gemäss der OSI erfüllen, sind nachfolgend aufgeführt. Dabei sind die GPL und die LGPL die mit Abstand am weitesten verbreiteten Lizenzmodelle [OSL, IFROSS].

GNU General Public License (GPL)

1983 legte Richard Stallman, Computerpionier am MIT, den Grundstein des GNU-Projekts. GNU sollte das proprietäre UNIX-Betriebssystem (bestehend aus einem Kernel und verschiedenen Anwenderprogrammen) durch freie Software ersetzen. Es sollte jedem gestattet sein, den Code der Software frei zu verändern und weiterzugeben. Eine kommerzielle Nutzung war dennoch nicht ausgeschlossen; so verkaufte Richard Stallman selbst frühe Versionen der GNU-Software auf Datenträgern.

1989 entwickelte das GNU-Projekt für seine Software eine einheitliche Lizenz, zur Verankerung der folgenden Freiheiten:

- Freiheit, das Programm für jeden Zweck auszuführen;
- Freiheit, den Quelltext des Programms zu studieren und anzupassen;
- Freiheit, das Programm zu kopieren;
- Freiheit, das veränderte Programm zu kopieren.

Diese Freiheiten sollten auch langfristig sichergestellt werden. Zu diesem Zweck enthält die GPL zwei wesentliche Klauseln, welche das Programm selbst als auch vom Quelltext abgeleitete Varianten (Derivate) betrifft:

4 Unter proprietärer Software verstehen wir Software, die keine Open Source Software ist.

- Jedes Derivat muss ebenfalls vollständig unter der GPL lizenziert werden;
- Bei Weitergabe des Binärprogramms muss der Quelltext des gesamten Programms mitgeliefert oder auf Anfrage ausgehändigt werden.

Die Anwendung der GPL auf eine Programmbibliothek führt dazu, dass jedes Programm, welches diese Bibliothek nutzt (gleich ob statisch dazu gebunden oder dynamisch zur Laufzeit aufgerufen) ebenfalls unter der GPL stehen muss. Eine eigene Problematik ist die Zusammenstellung von GPL- und nicht GPL-lizenzierten Werken, auf welche wir hier nicht weiter eingehen.

Das GNU-Projekt erwies sich als recht erfolgreich. Heutige Linux-Systeme beruhen zu grossen Teilen auf GNU-Software (z. B. GNU C-Compiler, GNOME-Desktop). Schon aus dieser Tradition heraus verwenden viele neue Open Source Projekte die GPL. Die GPL garantiert, dass freie Software stets freie Software bleibt und dass Software-Firmen nur dann GPL-lizenzierte Software in eigenen Produkten einsetzen und verkaufen, wenn sie den Quelltext ihrer Software frei zugänglich machen.

GPLv3

Die dritte und neuste Version der GNU General Public License wurde am 29. Juni 2007 verabschiedet. Sie löst damit die seit 16 Jahren bestehende Version 2 ab. Mit den Anpassungen soll die GPLv3 [GPLv3] den heutigen Begebenheiten in der Softwarelizenzwelt besser Rechnung tragen.

Ein oft auftretendes Problem bei juristischen Formulierungen ist, dass gewisse Begriffe je nach Land unterschiedlich interpretiert werden, was dazu führen kann, dass die Lizenzbedingungen unterschiedlich ausgelegt werden. Diesem Umstand trägt die GPLv3 vermehrt Rechnung, indem sie alle Begriffe, welche missverständlich interpretiert werden können, genauer erklärt. So wird beispielsweise neu der Begriff «convey» anstatt dem bisherigen «distribute» für die Verbreitung einer Software verwendet [QG].

Die GPLv3 passt sich aber auch an neue Copyright-Begebenheiten an, welche in den letzten Jahren unter anderem im Rahmen des Digital Millennium Copyright Act (DMCA) Einzug gehalten haben. Der DMCA untersagt zum Beispiel die Umgehung technischer Schutzmassnahmen wie DRM (Digital Rights Management) in Werken. Anders als die GPLv2, welche DRM generell ablehnt, erlaubt die GPLv3 DRM-Funktionalität in GPL-lizenzierten Quelltext einzubauen. Gleichzeitig deklariert die GPL jedoch ein DRM, welches mit GPL-lizenziertem Quelltext implementiert wurde, als eine technisch nicht wirksame Schutzmassnahme, weshalb der DMCA in solch einem Fall ungültig und damit die Umgehung des DRM erlaubt ist.

Neben dem DMCA, welcher zu grosser Aufregung geführt hat, sind in letzter Zeit einige Probleme mit Software-Patentklagen in den USA aufgetreten. Um Benutzer und Entwickler von GPL-basierter Software vor solchen Patentklagen zu schützen, schreibt die GPLv3 vor, dass kein Lizenznehmer gegen einen Anderen eine Patentklage einreichen darf, so lange dieser den GPL-Code im Rahmen der Lizenz nutzt oder weiterentwickelt.

Des weitern bietet die GPLv3 das uneingeschränkte Recht auf die Nutzung modifizierter Software auf einem Gerät, welches mit GPL-lizenzierter Software ausgestattet ist. Eine sogenannte *Tivoization*[5] ist verboten. Grundsätzlich ist eine Verschlüsselung oder eine Integritätsprüfung zum Schutz des kompilierten Programms zulässig, allerdings muss auf Verlangen die Information, die zur Entschlüsselung nötig ist, herausgegeben werden.

Die dritte Version der GPL zeichnet sich zudem durch eine bessere Lizenzkompatibilität aus. Eine Fremdlizenz ist dann mit GPLv3 kompatibel, wenn die Fremdlizenz ausschliesslich Zusatzparagraphen enthält, welche die GPL nicht lockern, falls sie weggelassen werden. So kann zum Beispiel in einer Fremdlizenz die Verwendung geschützter Markenzeichen in einem zusätzlichen Paragraphen explizit untersagt sein. Eine solche Fremdlizenz ist dennoch zur GPLv3 kompatibel, denn die Verwendung geschützter Markenzeichen wäre auch ohne diesen Paragraphen unzulässig.

Mozilla Public License (MPL)

Als im Februar 1998 der Quelltext des Web-Browsers Mozilla freigegeben wurde, stellte das Mozilla-Projekt gleich auch eine neue Software-Lizenz [MPL] vor, die Mozilla Public License Version 1.0. Mittlerweile wird die revidierte Version 1.1 verwendet. Die MPL enthält einige neuartige Klauseln, ähnelt aber grundsätzlich der LGPL.

Unterschieden wird zwischen Dateiderivaten und Werkderivaten. Dateiderivate sind Änderungen an einzelnen MPL-lizenzierten Dateien, ihre Zusammenführung oder Inklusion in anderen Dateien. Werkderivate sind Werke, die Funktionen aus den MPL-lizenzierten Dateien aufrufen oder von ihnen aufgerufen werden. Der Quelltext von Dateiderivaten muss auf Anfrage ausgehändigt werden. Dateiderivate müssen ebenfalls unter der MPL lizenziert werden. Werkderivate können dagegen beliebig lizenziert werden.

Damit wird sichergestellt, dass die Schnittstellen der Applikation offen bleiben, einzelne Erweiterungen jedoch proprietär sein können. Diese Vorgehensweise ist sicherlich besonders

5 Benannt nach dem Hardware-Mediaplayer TiVo, welcher trotz GPL-lizenziertem Quelltext die Ausführung modifizierter Software mittels Integritätscheck verhinderte.

bei einem Web-Browser nachvollziehbar, aber auch für andere modulare Systeme geeignet.

Zwar schreibt die MPL für Dateiderivate die Aushändigung des Quelltexts vor, erlaubt aber für die Binärdateien beliebige Lizenzierung, sofern die Lizenzbedingungen nicht mit der MPL in Konflikt stehen. So kann z. B. die Verbreitung von bestimmten Binärdateien untersagt werden, die Herstellung und freie Verbreitung von Binärdateien durch Dritte jedoch nicht.

Die MPL enthält auch eine salvatoresche Klausel, was bedeutet, dass die verbleibenden Bedingungen auch rechtskräftig sind, wenn es einzelne Bedingungen aufgrund im Lande des Rechtsvorgangs geltender Bestimmungen nicht sind. So könnten z. B. Kryptographie-Exportverbote eine Verbreitung bestimmten Quelltexts verhindern, dies würde jedoch die übrigen MPL-Bedingungen nicht tangieren, sie müssen soweit möglich erfüllt werden.

Seit Version 1.1 erlaubt es die MPL, einzelne Dateien des Werkes unter mehreren Lizenzen zur Verfügung zu stellen, was bei Wahl einer GPL-kompatiblen Lizenz auch die Verbindung von GPL- mit MPL-lizenziertem Quelltext ermöglicht.

GNU Lesser General Public License (LGPL)

Möchte man den Open Source Gedanken so weit wie möglich unterstützen, sollte man eine Copyleft-Lizenz wählen, also z. B. GPL oder MPL. Möchte man dagegen nicht verhindern, dass proprietäre Programme den eigenen Quelltext nutzen und womöglich gewinnbringend vermarkten, wie dies beispielsweise für die Etablierung eines De-Facto-Standards mithilfe von Programmbibliotheken oft notwendig ist, so kann man sich für eine Copyleft-freie Lizenz entscheiden. Die grosse Verbreitung einer neuen Technologie wie z. B. des Bildformates PNG [PNG] hängt nicht unwesentlich davon ab, dass verschiedenste Software-Hersteller die Programmbibliothek libpng integrieren und dadurch die Bildformate der PNG-Familie in ihren Produkten unterstützen.

Für Programmbibliotheken bildet die LGPL einen speziellen Kompromiss, der den Copyleft-Effekt aufrechterhält, aber gleichzeitig die Nutzung innerhalb proprietärer Software nicht unnötig erschwert.

Ursprünglich hiess die LGPL «Library General Public License», sie wurde jedoch in «Lesser» umbenannt, da Richard Stallman befürchtete, sie würde sonst zu häufig für Programmbibliotheken verwendet. Der Begriff Lesser soll kenntlich machen, dass die Nutzungsfreiheit (im Sinne des GNU-Projekts) pragmatischen Motiven untergeordnet wird. Die LGPL unterscheidet sich von der GPL im Wesentlichen dadurch, dass sie sowohl die statische als auch die dynamische Bindung einer LGPL-lizenzierten Bibliothek mit beliebigen Programmen erlaubt, ohne dass automatisch das gesamte Programm unter den Bedingungen der Lizenz [LGPL] stehen müsste. Allerdings muss ein Reverse Engineering des gesamten Programms gestattet sein.

Die LGPL erzwingt gleichzeitig das «Copyleft» für den Code der eigentlichen Programmbibliothek: Alle Veränderungen an der Bibliothek selbst müssen wiederum frei verfügbar sein. Weiter wird von dem die Bibliothek verwendenden Programm verlangt, dass es auf die Benutzung der Bibliothek an prominenter Stelle deutlich aufmerksam macht.

BSD License

Die BSD-Lizenz hat eine andere historische Tradition als die GPL bzw. LGPL. Der Name stammt von der Berkeley Software Distribution, einer seit den 1970er Jahren entwickelten UNIX-Distribution (BSD-Lite), die Anfang der 1990er als Open Source Software zur Verfügung gestellt wurde. Daraus entwickelten sich verschiedene freie UNIX: OpenBSD, NetBSD, FreeBSD usw. Die Lizenzbedingungen [BSD] sind einfach:

- Das Programm darf in jeder Form, auch in Binärform, weitergegeben werden. Eine Pflicht zur Überlassung des Quelltexts besteht nicht;
- Bei der Weitergabe in Binärform muss die Lizenz den Dateien beigefügt werden;
- Bei Derivaten darf der Name der Autoren bzw. des Herstellers nicht ohne Erlaubnis für Werbezwecke verwendet werden.

Zusätzlich enthielt die erste Version der BSD-Lizenz eine so genannte Werbeklausel, die es erforderlich machte, in Werbematerialien auf die Universität Berkeley hinzuweisen. Diese Klausel wurde 1999 aufgrund von Beschwerden aus den originalen Quellen von BSD-Lite entfernt. Das FreeBSD übernahm diese Lizenzänderung grösstenteils für die eigenen Erweiterungen, NetBSD und OpenBSD folgten dem nicht. Die erste Version der BSD-Lizenz wird dennoch von vielen Open Source Projekten verwendet. So finden sich in allen BSD-Versionen (und bei anderen, die BSD-lizenzierte Software vertreiben) etliche Dutzend dieser Klauseln, mit dem Hinweis auf den jeweiligen Verfasser.

Apache Software License

Die Apache Software Foundation, die von namhaften IT-Unternehmen unterstützt wird, hat für ihre Open Source Serverprodukte (am bekanntesten ist der Apache Webserver) eine eigene Lizenz entwickelt. Diese Lizenz erlaubt Veränderungen am Quelltext und die ausschliessliche Weitergabe in Binärform, sofern die Lizenzbedingungen [Apache] beigefügt werden. Sie enthält eine BSD-ähnliche Werbeklausel, die in Version 1.1 auf Dokumentation beschränkt wurde. Sie verbietet ausserdem den Gebrauch des Namens Apache für Derivate.

Die Apache Software-Lizenz ist sehr stark auf Apache-Produkte spezialisiert und empfiehlt sich kaum für allgemeine Open Source Applikationen.

Auswahl einer passenden Lizenz

Die zuvor aufgelisteten Software-Lizenzen sind nur eine kleine, bekannte Auswahl. Selbstverständlich steht es jedem Entwicklungsteam offen, eine eigene Software-Lizenz zu verfassen. Während grosse Anbieter diesen Aufwand nicht scheuen, so greifen kleinere Entwicklungsteams üblicherweise auf bestehende Lizenzen zurück. Dabei erfordert die grosse Verschiedenheit dieser Lizenzen, dass bei der Wahl einer passenden Lizenz für ein eigenes Open Source Projekt, mehrere Aspekte beachten werden.

So sollte man sich zunächst überlegen, welche Fremdbibliotheken, -module oder welchen fremden Quelltext man in der eigenen Software verwendet und unter welchen Bedingungen diese publiziert worden sind. Wird fremder Open Source eingesetzt, so ist die Wahl der eigenen Lizenz stark eingeschränkt, da die Lizenzen untereinander kompatibel sein müssen. Verwendet man hingegen keine anderen Open Source Projekte, so ist man frei in der Wahl und muss sich lediglich über die angestrebte Nutzung der eigenen Software im Klaren sein.

Die zuvor besprochenen Lizenzen unterscheiden sich hauptsächlich im Copyleft-Effekt, d.h. in der Pflicht, Erweiterungen und Verbindungen zu anderer Software ebenfalls unter eine Open Source Lizenz zu stellen. Die BSD-Lizenz ist in dieser Hinsicht die liberalste, die GPL die restriktivste. In Tabelle 1 fassen wir diesen Sachverhalt zusammen.

Die GPL, die beinahe schon als Standardlizenz gilt, macht meist am wenigsten Probleme.

Lizenz	Freiheits-grad	Copyleft-Effekt		Urheber-Hinweis
		Quelltext verwenden	Bibliothek einbinden	
BSD	hoch	beliebige Lizenz	beliebige Lizenz	in Derivaten keine Werbung mit dem Urheber
MPL	mittel	MPL	beliebige Lizenz	in Derivaten Hinweis auf die Lizenz
LGPL	mittel	LGPL	beliebige Lizenz	in Derivaten Hinweis auf die Lizenz an prominenter Stelle
GPL	gering	GPL	GPL	Hinweis auf Lizenz in allfällig vorhandenem Lizenzdialog

Tabelle 1: Vergleich der wichtigsten Lizenzen bezüglich Freiheitsgrad, Copyleft-Effekt und notwendiger Hinweise auf die Urheberschaft.

Der wesentliche Unterschied zu vielen anderen Lizenzen ist das Verbot der Weitergabe in der Binärform ohne die Bereitstellung des Quelltextes. Als Urheber hat man stets die Möglichkeit, die Lizenz einer Software zu ändern, was im Fall von freigegebenen Inhalten jedoch nicht unproblematisch ist, da die veränderten Lizenzbedingungen nur für neue Lizenznehmer gelten. Faktisch führt dies dazu, dass Rechte an bestehendem Quelltext nicht eingeschränkt werden können. Es ist aber durchaus möglich, ab einem bestimmten Stand des Open Source Projektes, den neu entwickelten Quelltext unter neue Lizenzbedingungen zu stellen. Wird von einem Urheber diese Möglichkeit wahrgenommen, mag sich jedoch der eine oder andere aktive Open Source Mitentwickler vor den Kopf gestossen fühlen.

Anwendung von Lizenzen

Eine Lizenz anwenden ist einfach: Die Lizenz sollte als Datei dem Programm beigefügt werden – in der Unix-Welt hat sich der Dateiname COPYING eingebürgert. Jede Quelltext-Datei sollte in kurzer Form auf den oder die Urheber, den Namen der Lizenz, die Version und die Datei verweisen, also z. B.:

```
Diese Datei ist Bestandteil von
MegaCalc, 1994-2003 entwickelt von
Werner Muster, Klaus Beispiel und
anderen, siehe die Datei CREDITS für
Details. MegaCalc ist freie Software
unter den Bedingungen der GNU
General Public License Version 2.
Die Haftung ist ausgeschlossen.
Diesem Programm sollte die Datei
COPYING beiliegen, welche die
Lizenzbedingungen enthält. Sollte
dies nicht der Fall sein, können Sie
die Lizenz bei der Free
Software Foundation (www.fsf.org)
anfordern: Free Software Foundation,
Inc., 59 Temple Place, Suite 330,
Boston, MA 02111-1307 USA.
```

Zusätzlich empfiehlt es sich und bei bestimmten Lizenzen ist es sogar erforderlich, die Lizenzbedingungen beim Programmstart oder bei der Installation mindestens einmal einzublenden.

Bei mehrsprachiger Software sollte der Hinweis in englischer Sprache eingefügt werden. Anwendungsbeispiele für die einzelnen Lizenzen finden sich in einer Liste von Open Source Lizenzen der OSI [OSIa].

Lancierung eines Open Source Projektes

Was bedeutet es, ein Open Source Projekt zu lancieren und es zu unterhalten? Welche Verantwortungen und Aufgaben gegenüber der grossen Internetgemeinde nimmt man dadurch auf sich? In diesem Abschnitt wollen wir versuchen, diese Fragen zu klären und zu zeigen, dass die Initiierung eines Open Source Projektes kein Hexenwerk ist und dass sich der Unterhalt des

Projektes in Grenzen hält. Durch Verwendung von vorhandenen Open Source Plattformen kann einerseits viel Eigenarbeit gespart und anderseits der Projektfortschritt beschleunigt werden.

Loslassen

Einer der schwierigsten Punkte bei der Lancierung eines Open Source Projektes ist vermutlich der, dass der eigene Programmcode aus der Hand gegeben werden muss. Das erfordert eine gewisse Bereitschaft und Offenheit, denn nach der Veröffentlichung kann nicht mehr selbst bestimmt werden, was mit dem Quelltext alles gemacht wird. Es bestehen aber immer noch genügend Möglichkeiten, beim lancierten Projekt die wichtigen Dinge zu entscheiden. Die Kontrolle darüber, welcher Quelltext und welcher nicht zum Projekt beigesteuert wird, wird nicht abgegeben.

Hat man sich erst einmal damit abgefunden, dass der Quelltext ab jetzt allen zugänglich ist, so sollen auch die Möglichkeiten genutzt werden, die ein Open Source Projekt bietet. Primär geht es um die potentiell grosse Zahl an möglichen «Mitarbeitern». Sehr bald werden einzelne, speziell interessierte Benutzer aus der Masse hervorstechen. Diese gilt es einzubinden. Meist haben solche Benutzer oder eben auch Entwickler am Inhalt des Projektes ein eigenes, grosses Interesse und sind selbst daran interessiert, Verbesserungen oder Erweiterungen am Projekt vorzunehmen. Damit diese Entwickler das nicht im stillen Kämmerlein für sich selbst tun und das Projekt nicht davon profitiert, muss man dafür sorgen, dass engagierte Entwickler ihre Leistungen ins Projekt einbringen können. Verlässlichen und interessierten Entwicklern soll daher Zugriff auf die Quelltext-Verwaltung gegeben werden. Einerseits können diese so ihre Entwicklungen dem Projekt beisteuern, andererseits können sie einem auch Arbeit abnehmen, indem sie z.B. rapportierte Fehler beheben. Es darf von der Nutzergemeinde durchaus auch etwas erwartet werden, und die Bereitstellung des eigenen Quelltexts in einem Open Source Projekt muss nicht als rein selbstloser Akt betrachtet werden.

Vorabklärungen

Ist die Projektidee für ein Open Source Projekt vorhanden, oder hat man sich entschieden, ein bestehendes Projekt als Open Source frei zu geben, so muss eine geeignete Software-Lizenz ausgewählt und eine Plattform für die Entwicklung, Pflege und Verwaltung des Open Source Projektes bestimmt werden. Bei der Wahl der Plattform stehen grundsätzlich zwei Möglichkeiten zur Verfügung: Aufsetzen einer eigenen oder Auswahl einer bestehenden.

Der Aufwand für die Wahl einer passenden Lizenz und einer komfortablen Plattform hängt stark von den Vorkenntnissen und den gesammelten Erfahrungen mit Open Source Projekten ab. Ist überhaupt kein Vorwissen vorhanden, und muss man sich zuerst über Lizenzen, Plattformen usw. einlesen, so kann dieser Arbeitsschritt bis zu einer Arbeitswoche beanspruchen.

Eine geeignete Plattform beinhaltet Client-Server-Werkzeuge zur verteilten Software-Entwicklung und zur Koordination der Abläufe zwischen den beteiligten Personen. Dazu gehören üblicherweise auch Funktionen zur Verwaltung der Projektmitglieder.

Neben diesen einzelnen Werkzeugen existieren auch komplette Open Source Plattformen, welche die zuvor genannten Dienste allesamt anbieten. Solche Open Source Plattformen bieten darüber hinaus eine erhöhte Publizität, da das eigene Projekt vom Bekanntheitsgrad der Plattform selber profitieren kann. Als weiterer Vorteil kommt hinzu, dass man sich nicht selbst um die Hardware- und Softwareinfrastruktur kümmern muss.

Eine Auswahl an Plattformen

Zur Führung von Open Source Projekten stehen einige komplette Plattformen zur Auswahl. Nicht alle sind gleich populär und nicht alle bieten die genau gleichen Dienste an.

Die Entwicklung des GNU-Projektes läuft auf *Savannah* [Sav]. Derzeit werden dort etwa 2800 Projekte verwaltet, an denen mehr als 50'000 registrierte Benutzer teilnehmen. Neben den GNU-eigenen Projekten sind unter den 2800 Projekten etliche GNU-fremde angesiedelt.

BerliOS [Berli] ist eine Plattform, die hauptsächlich von Mitgliedern des *Fraunhofer Institutes für Offene Kommunikationssysteme (FOKUS)* [FOKUS] betrieben wird. Gemäss eigenen Angaben hat BerliOS sich zum Ziel gesetzt, die unterschiedlichen Interessengruppen im Umfeld der Open Source Software zu unterstützen und dabei eine neutrale Vermittlerfunktion anzubieten. Die Zielgruppen von BerliOS sind einerseits die Entwickler und Anwender von Open Source Software und andererseits kommerzielle Hersteller von Open Source Betriebssystemen und Anwendungen sowie Support-Firmen.

Die bekannteste und mit über 100'000 unterhaltenen Projekten auch grösste Plattform ist *SourceForge* [SrcF]. Sie wird von der *Open Source Technology Group* [OSTG] betrieben und besitzt über eine Million registrierte Benutzer. Die Nutzung der Plattform ist kostenlos. SourceForge bietet eine zentralisierte Projekt-, Versions- und Release-Verwaltung mittels Subversion an. Zur Kommunikation zwischen Entwicklern oder Benutzern stehen den Projekten Mailinglisten und Foren zur Verfügung. So genannte Tracker realisieren das Fehlermelde- und Supportsystem, sowie die Mechanismen zur Überbringung von Benutzerwünschen (*feature requests*).

Mittels webbasierter Werkzeuge können die Projektmitglieder verwaltet und deren Rechte im Projekt konfiguriert werden. Es besteht die Möglichkeit, auf der Projektseite Neuigkeiten zum Projekt zu verkünden und neben dem Quelltext als solches, können vorkompilierte und zur sofortigen Nutzung bereite Programmversionen oder Installationsprogramme der Software zur Verfügung gestellt werden.

Die Projekte können mit Schlüsselwörtern versehen und bestimmten Kategorien zugeordnet werden. So lassen sich Projekte mit der Suchfunktion von SourceForge einfach finden. Ein Projekt wird von SourceForge automatisch mit Statistiken versehen, die Auskunft darüber geben, wie aktiv ein Projekt ist. Für Personen auf der Suche nach einem geeigneten Projekt ist das ein hilfreiches Kriterium für die Wahl eines Projektes. Projekte, die sehr aktiv sind, werden auf der Einstiegsseite von SourceForge prominent beworben.

Eigene Infrastruktur

Wird nicht auf eine vorhandene Plattform zurückgegriffen, so kann ein Projekt auch auf einem eigenen, über das Internet erreichbaren Server unterhalten werden. Auf diesem Server sollten folgende Dienste (mit Beispielanwendungen) laufen:

- Webserver: für die Projektwebseite;
- CVS oder Subversion: Versions- und Release-Verwaltung [CVS];
- Mailman [MLM]: für Mailinglisten;
- phpBB [BB]: Forensoftware.

Um einen derartigen Server einzurichten, muss mit 2 bis 4 Arbeitstagen gerechnet werden, je nach Erfahrung im Umgang mit Internet-Server und je nach Funktionsumfang des Servers. Einmal sauber und korrekt eingerichtet, sollte eine solche Infrastruktur keinen grösseren Aufwand zur Pflege benötigen. Zu den laufend anfallenden Arbeiten gehören z. B. Software-Updates und Backups.

Veröffentlichung

Sobald eine geeignete Open Source Plattform zur Verfügung steht, kann die Veröffentlichung des vorhandenen Projektes unter der gewählten Lizenz angegangen werden. Dazu gehört das Einrichten einer Projekt-Webseite, das allfällige Einspielen des bereits vorhandenen Quelltextes in die Versionsverwaltung, die Erstellung eines ersten Releases und die Einrichtung der Kommunikationskanäle. Der Aufwand für diesen Schritt hängt stark von der verwendeten Open Source Plattform ab. Bei SourceForge nimmt die gesamte Einrichtung etwa einen Arbeitstag in Anspruch.

Da sich die Nutzer eines Projektes grundsätzlich in den beiden Rollen Endbenutzer und Entwickler unterscheiden, sollten für diese verschiedenen Gruppen auch unterschiedliche Informationskanäle eingerichtet werden. Folgende Bezeichnungen haben sich für die Benennung von Mailinglisten resp. von Foren etabliert: *<projectname>-devel und <projectname>-user.*

Nicht nur bei der Benennung von Kommunikationskanälen, sondern auch bei der Strukturierung des Quelltextes, der Unterstützung von Entwicklerwerkzeugen und nicht zuletzt bei der Programmierung des Quelltextes selbst haben sich in der Open Source Community einige Gepflogenheiten durchgesetzt. Manchmal sind die Erwartungen der Open Source Gemeinde derart deutlich, dass die Gemeinde nicht immer sehr *open minded* erscheint. So werden in Foren Neulinge mit scheinbar normalen Fragen mit unfreundlichen Antworten abgekanzelt, sollten sie in der Fragestellung eine der vielen ungeschriebenen Regeln nicht beachtet haben. Die Antworten werden dann oft in einem typischen Abkürzungskauderwelsch gegeben, so dass der verwunderte Neuling nicht mal den Grund der seltsam abweisenden Antwort erahnen kann; oder würden Sie erwarten, dass *RTFM* so viel wie «read the fucking manual» bedeutet? Nicht dass wir selbst auf diese Regeln Wert legen würden, aber ist man am Erfolg des eigenen Open Source Projektes interessiert, lohnt es sich auf einige, zum Teil sogar sinnvolle, Erwartungen der Open Source Gemeinde einzugehen.

Das Open Source Mantra

Als das Open Source Mantra[6] wird oft folgende Regel angesehen: Release early and *release often*. Offenbar wird es in der Open Source Gemeinde wenig geschätzt, wenn ein Projekt fixfertig als Open Source freigegeben wird. Doch noch wichtiger scheint zu sein, dass man regelmässig die Änderungen publik macht und nicht nur grosse Meilensteine veröffentlicht.

Ein Projekt sollte wenn immer möglich auch für die verschiedenen Entwicklungsumgebungen vorbereitet sein. Sehr positiv aufgefasst wird, wenn neben den Microsoft Visual Studio Projektdateien auch die notwendigen GNU Autotools-Dateien (autoconf, automake und libtool) [AuTo] für die Kompilierung des Projektes mit der *GNU Compiler Collection* [GCC] beiliegen.

Damit die Entwicklung auf verschiedenen Plattformen möglichst reibungslos klappt, sollte schon bei der Organisation des Quelltextes auf eine gewisse Verzeichnisstruktur Wert gelegt werden. Folgende Struktur hat sich z. B. für C++-Programme etabliert und wird auch so erwartet:

```
<projektname>-
  |
  +-doc/
  +-include/
  |  +-<projektname>/
  |  |--*.h
  +-src/
  |--*.cpp
```

6 Eine formelhafte Wortfolge, die repetitiv rezitiert wird.

Bei einem Open Source Projekt werden auch bestimmte Dateien erwartet, die unter anderem die Lizenz enthalten (COPYING), Neuigkeiten bereit stellen (NEWS), die Autoren erwähnen (AUTHORS) oder häufig gestellte Fragen beantworten (FAQ).

Über die eingerichteten Kommunikationskanäle wie Mailinglisten und Foren unterhält man sich mit den Interessenten des Projektes. Zu Beginn muss man selbst stark mit Rat und Tat zur Seite stehen, doch je schneller die Zahl der Benutzer steigt, desto rascher ist die Community selbst im Stande, sich mit Hilfestellung zu unterstützen. Es wird zwar stark geschätzt, wenn die Projekt-initiatoren die Kommunikationskanäle weiter verfolgen und wertvollen Rat beisteuern, doch wird das nicht erwartet. Je nach Aktivität in den Mailinglisten und den Foren reicht ein wöchentlicher Besuch in den Mailinglisten und Foren, um das Projekt genügend aktiv zu begleiten.

Wenn das Projekt und die verwendeten Techniken es zulassen, macht es Sinn, ein Projekt möglichst plattformunabhängig zu programmieren. Diese Möglichkeit sollte man auch aus Eigeninteresse nutzen. Das ist nicht nur der grösseren «Mitarbeiterzahl» wegen interessant, sondern auch deswegen, dass das eigene Projekt auf verschiedenen Plattformen betrieben werden kann.

Verwendet man z. B. C++ als Programmiersprache, so lässt diese Sprache die plattformunabhängige Entwicklung grundsätzlich zu. Eine Portierung auf eine andere Betriebsystem-Plattform ist aber nur dann einfach zu bewerkstelligen, wenn die im Projekt verwendeten Programmbibliotheken auch auf der Zielplattform vorhanden sind. Bei C++ Programmen sollte aus diesem Grund möglichst auf die Standardbibliotheken, wie z. B. die *C++ Standard Library* [C++Lib] oder die *Standard Template Library (STL)* [STL], zurückgegriffen werden. Diese beinhalten einige wichtige Datentypen und Funktionen, die in praktisch jedem Programm verwendet werden: Zeichenketten, numerische Berechnungen, Ein- und Ausgabe, Fehlerbehandlung, Laufzeit-Typerkennung, Nationale Besonderheiten, Speicherverwaltung usw.

Überhaupt stellt schon die Wahl der Programmiersprache ein wichtiges Kriterium für die Portabilität dar. C/C++-Programme lassen sich auf fast allen Plattformen kompilieren. Bei Java ist die Plattformunabhängigkeit Konzept. C#-Programme hingegen lassen sich praktisch auf allen Windows-System zuverlässig ausführen. Für Unix-Plattformen kann auf die .NET-kompatible Laufzeitumgebung *Mono* zurückgegriffen werden [Mono].

Fallbeispiel: libPGF

Das vorliegende Fallbeispiel soll den Übergang von proprietärer zu Open Source Software-Entwicklung am Beispiel von *libPGF* [libPGF] illustrieren. *Progressive Graphic File (PGF)* [PGF] ist ein von uns in C++ selbst entwickeltes Bildformat, basierend auf diskreter Wavelettransformation.

Ein Projekt, gerade auch ein Open Source Projekt, braucht einen guten Namen, der nicht bereits von vielen anderen ähnlichen Bedeutungen belegt ist. Bei Dateiformaten fällt die Namenswahl nicht leicht, da unter gewissen Betriebsystemen Dateiformate über eine kurze Dateiendung gekennzeichnet werden. Obwohl gemäss einschlägigen Webseiten [FILExt, FExt] die Dateiendung PGF bereits mehrfach belegt ist («GPS Pathfinder Office Geoid Grid File», «PowerGREP File Selection», «Portfolio Graphics Compressed Bitmap», «Portfolio Graphics image format»), entscheiden wir uns dennoch für das prägnante Kürzel PGF. Den Projektnamen des Open Source Projektes lehnen wir an ähnliche Projekte an, wie zum Beispiel *libpng* [PNG], und wählen libPGF.

Im Zusammenhang mit dem Projektnamen steht eine starke Webpräsenz im Vordergrund. Dabei stellt ein einfaches Auffinden durch Suchmaschinen ein nicht unwesentlicher Erfolgsfaktor für die Bekanntmachung dar. Neben dem eigenen Webauftritt hilft ein Eintrag in Wikipedia [PGFa] für gute Platzierungen bei bekannten Suchmaschinen. Oft verwendete Abkürzungen oder Abkürzungen mit rufschädigenden Assoziationen sollten tunlichst vermieden werden.

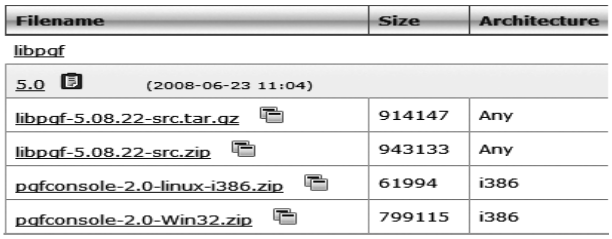
Lizenz

Für Programmbibliotheken ist die Wahl der geeigneten Lizenz entscheidend. Wird eine strenge Lizenz gewählt (z. B. GPL), steht der ideologische Aspekt der Aufrechterhaltung des Open Source Gedankens durch Copyleft im Vordergrund. Wird andererseits eine sehr liberale Lizenz angewendet (z. B. BSD), so steht einer grösstmöglichen Verbreitung nichts im Wege, da auch Hersteller proprietärer Software sich dieser Bibliotheken bedienen können.

Für uns als Entwickler von libPGF spielt der Open Source Gedanke eine wichtige Rolle, doch überwiegt das Interesse einer möglichst grossen Verbreitung. Aus diesem Grund kommt für uns die reine GPL nicht in Frage und LGPL bietet sich als pragmatische Lösung an. Nicht zuletzt da es sich bei libPGF um ein Bibliotheksprojekt handelt, passt die LGPL bestens.

Plattform und Internetpräsenz

Open Source Projekte leben davon, dass der Quelltext und andere Projektdateien der Entwicklergemeinde und den Anwendern zur Verfügung gestellt werden. Grundsätzlich wäre es ausreichend, den Quelltext auf einer eigenen Webseite anzubieten. Typische Abläufe der Softwareentwicklung wären damit aber nicht ausreichend unterstützt. Somit drängt sich die Verwendung



Filename	Size	Architecture
libpgf		
5.0 (2008-06-23 11:04)		
libpgf-5.08.22-src.tar.gz	914147	Any
libpgf-5.08.22-src.zip	943133	Any
pgfconsole-2.0-linux-i386.zip	61994	i386
pgfconsole-2.0-Win32.zip	799115	i386

Abbildung 1: Übersicht der angebotenen Inhalte des Release 5.0.

einer Open Source Plattform auf. Wir haben uns für SourceForge entschieden, da diese Plattform die Erstellung eines Projektes ohne grossen Zusatzaufwand ermöglicht und die mit Abstand bekannteste Plattform von allen ist.

In den nächsten Abschnitten beschreiben wir die von uns vollzogenen Schritte von der Erstellung eines SourceForge-Projektes bis zur Einrichtung einer unabhängigen Projektwebseite.

Projekt erstellen

Die Erstellung eines SourceForge-Projektes ist ziemlich einfach und besteht aus dem Anlegen eines SourceForge-Accounts und der offiziellen Registrierung des Projekts. SourceForge leitet einen sehr gut durch diesen Prozess und macht im Voraus schon darauf aufmerksam, welche Informationen vorhanden sein müssen. So sind unter anderem eine volle Beschreibung des Projektes, die gewünschte Open Source Lizenz und der Name des Projekts nötig. Darüber hinaus können Angaben zu den folgenden Punkten gemacht werden:

- SourceForge Software-Kategorie, in welche das Projekt passt;
- unter welchen Plattformen die Software funktioniert;
- in welcher Programmiersprache die Software geschrieben wurde;
- wie der Entwicklungsstand ist (Alpha, Beta, Version).

Dieser Projektantrag wird dann von Menschenhand kontrolliert und spätestens nach zwei Arbeitstagen erhält man einen Entscheid über Annahme oder Ablehnung des Projektes.

Projekt einrichten

Vorausgesetzt der Antrag ist akzeptiert worden, so steht anschliessend ein Projektplatz [SrcFo] zur Verfügung. Dadurch können die Kommunikationskanäle wie geplant eingerichtet, die Projektangaben detaillierter beschrieben (z. B. Link auf die Projektwebseite), die Versionsverwaltung mit dem Quelltext des aktuellen Entwicklungsstandes eingespeist und vielleicht ein erstes Release bereitgestellt werden.

Im libPGF-Projekt haben wir die folgenden zwei Foren und zwei Mailinglisten als Kommunikationskanäle eingerichtet:

- Das Announce-Forum dient zur Ankündigungen von grösseren Anpassungen unsererseits und ermöglicht das Führen von sachbezogenen Diskussionen.
- Im Discussion-Forum können sich Nutzer und Entwickler zu allen Themen rund um libPGF unterhalten. Das Forum ist auch dafür gedacht, dass sich die Nutzergemeinde hierin selbst hilft.
- Die libpgf-devel-Mailingliste ist für Diskussionen unter Entwicklern der libPGF selbst gedacht.
- Die libpgf-user-Mailingliste ist für Entwickler gedacht, die libPGF für andere Projekte benutzen wollen.

Quelltexte und Releases bereitstellen

In vielen Open Source Projekten werden neben den Quelltexten auch stabile Momentaufnahmen (*Releases*) bereitgestellt. Von diesen stabilen Momentaufnahmen werden oft vorkompilierte Binary-Releases (Binärprogramme) angeboten, um die doch zum Teil mühsamen Kompilationen den Benutzern zu ersparen. Entsprechend diesen Gepflogenheiten bieten wir neben dem aktuellen Entwicklungsstand, stabile Momentaufnahmen des Quelltextes und kompilierte Versionen an. Für die Releases muss ein geeignetes Versionierungsschema her. Die Versionierung bei der libPGF ist so zu lesen: libPGF-5.0 ist der fünfte Haupt-Release der Bibliothek. Eine kleinere (minor) Änderung, würde den Minor-Release libPGF-5.1 ergeben. Eine Fehlerbehebung für diese Version würde das Patchfix-Release libPGF-5.1-1 ergeben.

Um den Einsatz der Bibliothek zu illustrieren, stellen wir der Bibliothek ein prägnantes Demoprogramm im Quelltext und in kompilierter Version zur Seite (siehe auch Abbildung 1). Damit sollten die ersten Bedürfnisse von Anwendern der Bibliothek und jener, die einfach mal einen Blick auf die Fähigkeiten der Bibliothek werfen wollen, befriedigt sein. Die beiden Quelltextpakete (libpgf-5.08.22-src.tar.gz und libpgf-5.08.22-src.zip) unterscheiden sich nur in der Kompressionsart. Entpackt man diese bei sich auf dem Rechner, so erhält man ein Verzeichnis mit den typischen Dateien für ein Open Source Projekt:

```
libpgf/
  +-doc/
  +-include/
  |  +-libpgf/
  +-libpgf.xcodeproj/
  +-src/
  |--AUTHORS
  |--autogen.sh
  |--config.h.in
  |--configure.ac
  |--COPYING
  |--FAQ
  |--INSTALL
  |--libpgf.pc.in
  |--libpgf.spec.in
  |--Makefile.am
  |--NEWS
  |--PGFCodec.vcproj
  |--README
```

Für aktive Entwickler steht der Zugriff auf den aktuellen Quelltext mittels einer Versionsverwaltung zur Verfügung. Bei SourceForge kommt Subversion hierfür zum Einsatz. Der Lesezugriff auf die Dateien in der Versionsverwaltung ist aber auch jedem anderen Benutzer gestattet. Einen schreibenden Zugriff bekommen aber nur diejenigen SourceForge-Benutzer, denen wir als Projektverantwortliche Zugriff gestatten. Der Zugriff auf die Versionsverwaltung Subversion geschieht mittels dem Protokoll https und kann praktisch in jeder Entwicklungsumgebung komfortabel eingebunden werden[7].

Projektwebseite

Eine Projektwebseite dient der Verbreitung von allgemeinen Informationen zum Projekt. Es wird beschrieben, worum es sich beim Projekt handelt, welche Features angeboten werden, welche Einsatzgebiete geeignet sind, wer dahinter steht und welche Hintergrundinformationen zur Verfügung stehen. Nicht zuletzt dient eine solche Seite Marketingzwecken. Marketingtechnisch ist eine Projektwebseite, die unter einem eigenen, aussagekräftigen Domainnamen erreichbar ist, wertvoller als eine Seite, die unter einer Subdomain oder in einem Unterverzeichnis zu finden ist.

Ein Projektplatz bei SourceForge ist hauptsächlich als Kollaborationswerkzeug für die Entwicklung gedacht. Trotzdem bietet SourceForge pro Projekt 100 MByte Speicherplatz, PHP und eine MySQL-Datenbank an, um unter einer Subdomain von SourceForge.net eine Projektwebseite zu betreiben. Dabei setzt sich die URL folgendermassen zusammen: http://<projektname>.sourceforge.net.

Trotz dieser Möglichkeit haben wir uns für eine eigene Domain entschieden und betreiben die Projektwebseite bei einem selbst gewählten Hoster[8]. Dort bieten wir weitere Informationen zur libPGF an und verweisen für die Downloads und die Supportkanäle auf die entsprechenden SourceForge-Projektseiten. Vom Sourceforge-Projektplatz wiederum verweisen wir zur eigenen Projektwebseite www.libpgf.org.

Fazit und Schlussbemerkungen

Unsere bisherigen Erfahrungen mit dem Open Source Projekt libPGF sind durchwegs gut. Gleich zu Beginn haben wir zwei sehr interessierte Personen über die Kommunikationskanäle kennen gelernt. Während die erste Person uns bei der Erstellung der Releases geholfen hat (GNU Autotools und Vorschlag wie die Verzeichnisstruktur geändert werden sollte), zeigte die andere Person primär Interesse am Bildformat. Auf uns nicht bekanntem Weg fand das Bildformat denn auch schon Einzug in die Open Source Game-Engine OGRE [OGRE]. Das ist natürlich eine sehr schöne Sache.

Der weitere Aufwand für den Unterhalt des Projektes geht gegen null. Dann und wann sehen wir wieder mal in den Foren nach, ob da plötzlich eine rege Diskussion statt findet (was nicht der Fall ist) und bei den eingerichteten Mailinglisten habe wir uns natürlich auch angemeldet. Sendet jemand eine Meldung an diese Listen, so erhalten wir diese per E-Mail und können darauf antworten.

Genau zwei Jahre nach Veröffentlichung des Projektes sind der Quelltext der Bibliothek libPGF und das Demoprogramm gesamthaft 1180mal bezogen worden. Diese Zahlen sind nicht berauschend, was sicherlich auch damit zusammenhängt, dass sich unsere Aktivitäten auf ein Minimum beschränkten. Gerade was das Marketing anbelangt, hätten wir einiges mehr tun können. Zum Beispiel wäre ein Pressecommuniqué an die führenden Online-News-Dienste und technischen Verlage der Aufmerksamkeit dienlich gewesen.

7 Für Microsoft Visual Studio existiert z. B. Ankhsvn: http://ankhsvn.tigris.org/

8 In unserem Fall ist die Projektwebseite bei METANET GmbH, Switzerland, gehostet.

Referenzen

[Apache] Apache Software Lizenztext: http://www.apache.org/licenses/LICENSE-2.0.html

[AuTo] GNU Software: http://www.gnu.org/software/

[BB] PHP Bulletin Board: http://www.phpbb.com

[Berli] BerliOS: http://www.berlios.de

[BSD] BSD-Lizenztext: http://www.opensource.org/licenses/bsd-license.php

[C++Lib] GNU C++ Standard Library: http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html

[CHLR] Landesrechte: http://www.admin.ch/ch/d/sr/index.html

[CVS] Concurrent Versions System: http://www.nongnu.org/cvs/

[FExt] File Extensions: http://www.file-extensions.org

[FILExt] FILExt: http://filext.com

[FOKUS] FOKUS Institut: http://www.fokus.fraunhofer.de/home/

[FSF] Free Software Foundation: http://www.fsf.org

[GCC] GCC, the GNU Compiler Collection: http://www.gnu.org/software/gcc/

[GNU] GNU Free Documentation License: http://de.wikipedia.org/wiki/Wikipedia

[GPL] GNU General Public License: http://www.gnu.org/copyleft/gpl.html

[GPLv3] Offizieller GPLv3-Text: http://www.gnu.org/licenses/gpl.html

[IFROSS] Institut für Rechtsfragen der Freien und Open Source Software: http://www.ifross.de/ifross_html/lizenzcenter.html

[LGPL] LGPL-Text: http://www.gnu.org/licenses/lgpl.html

[libPGF] libPGF Open Source Image Codec: http://www.libpgf.org/

[MLM] GNU Mailing List Manager: http://www.gnu.org/software/mailman/

[Mono] Mono Project: http://www.mono-project.com/Main_Page

[MPL] MPL-Text: http://www.mozilla.org/MPL/MPL-1.1.html

[OGRE] Object-Oriented Graphics Rendering Engine; OGRE: http://www.ogre3d.org

[OSI] Open Source Initiative: http://www.opensource.org

[OSIa] Licenses by Name: http://opensource.org/licenses/alphabetical

[OSL] Open-Source-Lizenzen: http://openfacts.berlios.de/index.phtml?title=Open-Source-Lizenzen

[OSTG] OSTG: http://www.ostg.org

[PGF] xeraina PGF: http://www.xeraina.ch/pgf/

[PGFa] Progressive Graphics File: http://de.wikipedia.org/wiki/Progressive_Graphics_File

[PNG] Portable Network Graphics, lizenziert unter der zlib/libpng-Lizenz: http://www.libpng.org

[QG] GPLv3 Quick-Guide: http://www.gnu.org/licenses/quick-guide-gplv3.html

[Sav] Savannah: http://savannah.gnu.org

[SrcF] SourceForge: http://www.sourceforge.net

[SrcFo] SourceForge libPGF: https://sourceforge.net/projects/libpgf/

[STL] STL Programmer's Guide: http://www.sgi.com/tech/stl/

[SVN] Subversion: http://subversion.tigris.org/

[URG] Bundesgesetz über das Urheberrecht und verwandte Schutzrechte: http://www.admin.ch/ch/d/sr/c231_1.html

[Wiki] Wikipedia – Die freie Enzyklopädie: http://de.wikipedia.org

Fit for Business Process Testing

While low level unit testing has been quite well explored and is now established, the automated testing of complex business processes is still a challenge. One reason for this is that developers often just do not understand or speak the language of business people and vice versa. This makes the specification of test cases for real business processes more difficult. Another reason is that unit testing frameworks, which are very well suited for the automated testing of low-level classes, were not designed to test high-level processes needing extra infrastructure. In this article we will show how the open source framework Fit, which was designed for high-level acceptance testing, can be used for automated business process testing and how these tests can be added to legacy code.

Cecilia Garcia Garcia, Martin Kropp, Wolfgang Schwaiger | martin.kropp@fhnw.ch

With the increasing popularity of agile software development methods and of xUnit testing frameworks, unit testing has become very popular and can be seen as an accepted technique for automated low level component testing (units and methods). Unfortunately, this cannot (yet) be said for high-level testing on the acceptance, service, and especially, the business process level. The reasons for this lack of automated testing techniques are manifold [Fit, Mug05]:

- Business people and developers speak different languages;
- Business processes are complex and have more dependencies on external resources than low level components;
- The number of available automated testing tools is limited.

Different languages: Business people and developers are experts in their own application domains, each with their own domain language and terms. However, these different domains often do not understand one another either because of the different terms, or even worse, because of having the same expressions but with different meanings, which can cause fundamental misunderstandings.

Complexity: Business processes usually consist of a sequence of interactions between the user and the system under development, often with alternative control flows. Furthermore, business processes often require additional infrastructure, such as servers, databases, and communication channels, which at least can cause extra effort when implementing tests.

Automated Testing: Various tools for automated acceptance testing, the so-called Capture-Replay tools, are available (e. g. [jRapl]). However, they are typically bound to GUI controls and sometimes even to their absolute position in the window, which makes them prone to changes. Thus, a good support tool for the testing of high-level services and business processes is still missing [Mens07].

In this paper we will describe how the Framework for Integrated Testing (Fit), can be used to specify and implement automated business process tests for existing code and to help solve at least parts of the above mentioned problems [Fit].

Fit is an open source acceptance testing framework, which was developed by Ward Cunningham et al. Its goal is to bridge the gap between users and developers and to support automated testing on the acceptance level, thus addressing the above mentioned problems.

Case Study

We will demonstrate the testing of business processes by using the Java Pet Store web application [SunPet]. This application was chosen as a demonstrator, because (1) it contains the typical business processes of shopping applications, (2) it has no implemented test cases, in fact, it was not even designed for testing, (3) it shows the typical characteristics of a complex software product, such as the usage of different technologies for development (AJAX, J2EE, JSF), a complex structure, a database connection, legacy code, and (4) it is easy to understand from a user's point of view.

Figure 1: The «Pet Store» application

We will use the common and well-known «search-select-buy» shopping process of an online store.

The Shopping Business Process

When users enter the Java Pet Store application, (see Figure 1), they can choose to search for a pet. Then the search criteria is entered, which can include the kind of pet, parts of a description, and additional criteria (tags). Once the search has been completed, the results are displayed on a catalogue screen.

The user then continues the selection of the desired pet. In the catalogue, the user chooses the buy option using the external PayPal [PayPal] payment service system. The individual steps in the process can be seen below:

1.	Select search menu option
2.	Enter search criteria
3.	Check/Uncheck "Also Search Tags"
4.	Execute search
5.	Present search results
6.	Select desired pet
7.	Buy pet with the PayPal option

Table 1. The pet shopping process

Writing Test Cases

With Fit, once the business process has been specified, the user continues to write the test cases. As mentioned before, Fit tries to bridge the gap between users and developers by allowing the user to write the tests in his or her own domain language and from his or her own perspective.

To do this, Fit uses the concepts of *tables* and *fixtures* to specify the tests for *the system under test* (SUT) (see Figure 2). Fit reads the specified test cases from one or several HTML tables, interprets this data according to the underlying test type (fixture type), and then executes the tests on the SUT [Mug05].

This concept allows an automated, early and immediate feedback loop between users and developers, thus preventing the problem of late acceptance and business process testing.

Tables

Tables represent the user's view of the tests. Assuming that users and business people know how to deal with tabular representations in general, a tabular form for test case specification makes it easy for them to write tests.

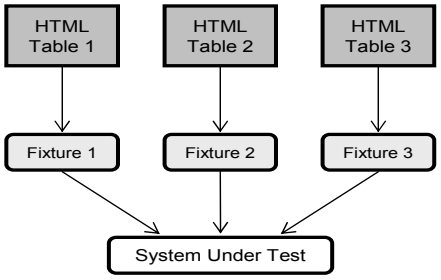


Figure 2: Fit concepts

Fit uses HTML tables to specify the tests and to report the test results. To indicate the success or failure of the tests, Fit uses a traffic light metaphor, which is well-known from other unit-testing frameworks [JUnit]. Moreover, Fit not only allows specifying expected data but also to specify commands and expected behavior and thus simulates user interaction and complete business processes. An example for the latter is given in Table 2, where the search-process of our case study with its input data and its expected behavior is specified.

To make it even more convenient for users and business people, tables can be written using other tools like Microsoft© Office (the generated files, however, have to be stored in *.html format). Furthermore, the wiki-based platform FitNesse [FitNesse] even allows the specification and execution of acceptance tests in distributed project teams over the web.

fit.ActionFixture		
start	Test search submit	
enter	search string	cat
enter	search tags	true
press	submit	
check	number of results	2

Table 2. Test cat search process

In general, the first row of a table always specifies the type of test to be executed, which is called a *fixture type* in Fit terminology (see table 2). The content of the subsequent rows depends on the fixture type and ranges from pure input data and expected data to commands and instructions as shown above. In the following, we will give an overview of the basic fixture types (a complete description can be found under [Mug05]).

ColumnFixture tables are used to test pure calculations and are the simplest type of table. Table 3 shows an example of a discount calculation test, which checks the correct discount percentage and the correct discount price calculation. It uses two input parameters (amount and customer category) and two expected values for comparison (percentage, discount price).

The first row specifies the name of the fixture, the second row the names of the input fields (amount, customer category) and the calculated values (percentage(), discount price()). Each following row of the table represents an independent test case.

Calculate discount			
amount	customer category	percentage()	discount price()
0.00	Gold	30%	0.00
120.00	Gold	30%	84.00
120.00	Silver	20%	96.00
999.00	Regular	0%	999.00

Table 3. ColumnFixture sample

Test search for available cats			
name	description	tags	price
Friendly Cat	This black and white colored cat is super friendly. Anyone passing by your front yard will find him puring at their feet and trying to make a new friend. His name is Anthony, but I call him Ant as a nickname since he loves to eat ants and other insects.	awesome interesting cool	307.10
Fluffy Cat	A great pet for a hair stylist! Have fun combing Bailey's silver mane. Maybe trim his whiskers? He is very patient and loves to be pampered.	awesome interesting cool	527.00

Table 4. Test database query result with RowFixture

RowFixture tables are a special type of ColumnFixtures. While the rows of the ColumnFixture table are executed independently of each other, rows of a RowFixture table build an entity and can represent a group, a list, a sequence, or a collection of things. As an example, Table 4 shows our «search cat» case study test.

ActionFixture tables are typically used to simulate actions, performed by the user on a screen, for example. Moreover, ActionFixture types are very well suited for testing even large, sequential processes. An example of an ActionFixture can be seen in Table 2. The left column contains the actions to be executed. Fit offers the following predefined action commands: *start*, *enter*, *press*, and *check*. The second column describes the action to be executed in the user's language, and the third column contains optional input parameters for the action.

Testing a Complete Business Process

We have seen how individual test cases can be specified very easily with tables, now we are going to specify the test for the complete shopping process as shown in Table 1. Fit addresses these requirements by combining several tables to a single HTML page.

We want to test the complete shopping process with the following tests and we will use the specified fixture types. The complete test process can now be easily written by putting all the different test tables into a single webpage as shown in Figure 3.

	Test	Description	Fixture Type
1.	Test Search Process	Test if the specified test criteria deliver the correct amount of data	ActionFixture
2.	Test Search Result	Test the returned result set for correctness	RowFixture
3.	Test Selected Product	Test if the correct product is read based on the product's ID	ActionFixture
4.	Test Payment	Check if the payment process works correctly	RowFixture
5.	Test Catalogue after selling a pet	Test if the sold cat is removed from the catalogue	RowFixture

Table 5. Test cases for the shopping process

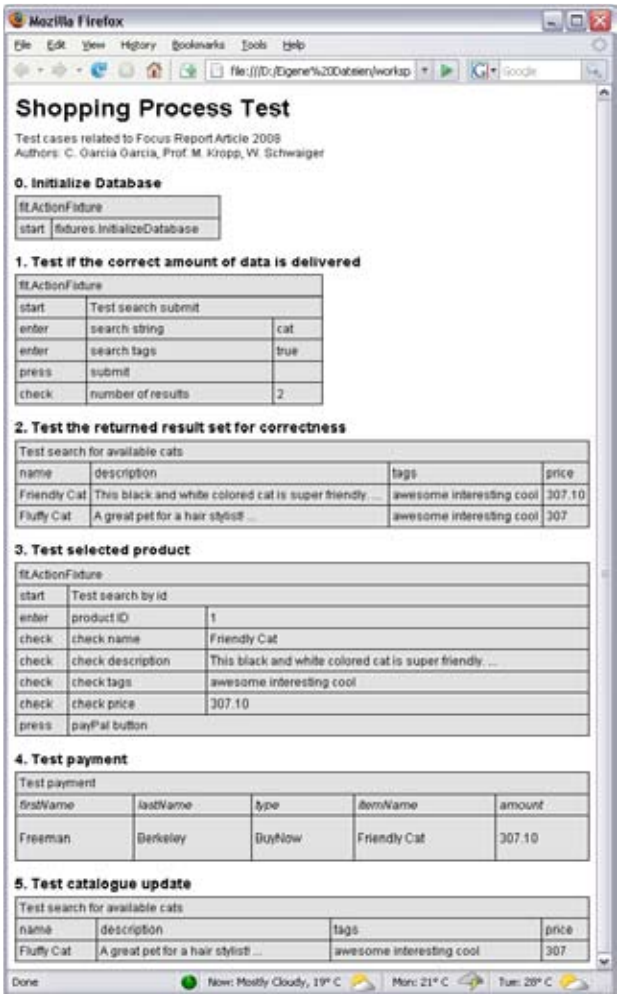


Figure 3: Complete shopping process test in HTML

In this special test case we expect that the user has found the pet of his or her choice and wants to buy it. A pet has been selected from the search result and then the user is redirected to the store's catalogue where the selected pet is pictured. On the catalogue screen the «PayPal» icon is selected to pay for the new mate. After payment has been made the purchased pet must be removed from the catalogue, which is then tested as seen in Figure 3.

If the Fit test cases are simple, then the table will be stand-alone and self-contained. This is not the case for multi-table test cases. In a test process, subsequent tables usually depend on the preceding ones. This requires that the test case planner and the test case implementer work closely together, to clearly communicate the dependencies which cannot be seen from the tables themselves.

Writing the Test Code / Binding Tables to Code

After the test cases have been specified in the user's own domain language, the developer can implement the fixtures. This glue code differs according to the implemented fixture type. Listing 1 shows the fixture code for second table from Figure 3.

The relevant statements have been highlighted. The first row in the table specifies that an action fixture will be used, i.e. that the first column of the

```
public class TestSearchSubmit extends Fixture {

    private ArrayList<Item> hitlist;

    public int numberOfResults() { return hitlist.size(); }

    public void searchString(String searchString)  {
        fixtures.InitializeDatabase.dataBaseMock.setSearchString(searchString);
    }
    public void submit() {
        hitlist = fixtures.InitializeDatabase.dataBaseMock.searchAction();
    }
    public void searchTags(boolean searchTags) {
        fixtures.InitializeDatabase.dataBaseMock.setSearchTags(searchTags);
    }
}
```

Listing 1: From test table to code

following rows contains actions to be executed. The second row specifies the name of the concrete test class (which is derived from the class Fixture). All subsequent rows contain the concrete actions to be executed and the necessary data. While the first column of these rows specifies the action type, the second column corresponds to the appropriate method in the test class, and the third (and, optionally, all the following columns) specify its input data. Fit allows a very readable form for the names in the table and automatically converts them into the appropriate classes, methods and variables in the fixture by removing punctuation and blanks and by using camel casing [FitGrace].

The third table is a RowFixture and its important implementation details are shown in Listing 2. A RowFixture contains two important methods; *getTargetClass()* and *query()*. The getTargetClass() method allows Fit to retrieve the concrete instance of the RowFixture class, which is then used to call the query()-Method. The query()-method assembles the elements of the actual list from the result list returned by the database and forwards it to Fit for comparison. In the PetObject class in Listing 2, it can be seen that the number of defined attributes does not exactly match the column count of the third table in Figure 3. Fit does not directly compare the fields of the returned query to a table row; it really adheres to the specified headings and attributes. The implementation of the remaining tables is equivalent to the just described code listings.

Conclusion

Writing automated business process tests poses additional challenges for the test specification and implementation.

In this article, we have shown how the Fit testing framework can be used to test business processes. Its approach of using tables to specify tests, independently of the actual GUI, makes it easy for business people and users to write test cases in their own domain language and thus it can indeed help to bridge the gap between developers and users. This makes Fit a perfect complement to xUnit testing frameworks used to test low level classes and methods.

In our case study we concentrated on the Fit core library. This library makes it very easy to write and run tests and to demonstrate the basic principles of business process testing. For more complex processes, however, the data setup and the test tables become cumbersome and hard to read [Mug05]. Moreover, to share a common context between multiple tables, the Fit core library uses the not terribly OO-like approach of public static members.

To avoid these problems, additional fixture types have been developed for the Fit Library [FitNesse, FitLib], which offers, for example, the fixture types SetUpFixture and DoFixture to handle setups, workflows and to share a common context between tables.

The open architecture of Fit enables the customization and the extension of its functionality to meet individual needs. The existing command set of ActionFixtures can be easily extended through the subclassing and the addition of a new method for each new command. To define a custom command set in a separate class, a new action class can be directly derived from the class Fixture and new commands implemented as new methods.

Moreover, developers can define their own fixture types by just deriving from any corresponding base fixture class. This allows the development of domain specific fixture types such as test, database, GUI and service [FitFix]. To enable fully automated regressions tests, Fit can be run from a command line.

Although there is room for improvements in Fit, e. g. usability and ease of use, but also the refactoring of existing test cases [Otting08], plug-in support and extensions [FitPro], Fit does well when acceptance and business process test-ing are considered, especially when it is used from the very beginning of a software project.

```
public class TestSearchForAvailableCats extends RowFixture {

    public Class<?> getTargetClass() { return PetObject.class;  }

    public Object[] query() {
        ArrayList<Item> resultList = fixtures.
            InitializeDatabase.dataBaseMock.searchAction();

        PetObject array[] = new PetObject[resultList.size()];

        for (int i=0; i < resultList.size(); i++) {
            array[i] = new PetObject(
                resultList.get(i).getProductID(),
                resultList.get(i).getName(),
                resultList.get(i).getDescription(),
                resultList.get(i).tagsAsString(),
                resultList.get(i).getPrice().toString(),
                resultList.get(i).getImageURL(),
                resultList.get(i).getImageThumbURL());
        }
        return array;
    }
}

// the Pet business object
public class PetObject {
    public String productID;
    public String price;
    public String name;
    public String description;
    public String tags;
    public String imageURL;
    public String imageURLThumb;

    public PetObject(String productID, String name, String description,
        String tags, String price, String imageURL, String imageURLThumb) {
        this.productID = productID;
        this.price = price;
        this.name = name;
        this.description = description;
        this.tags = tags;
        this.imageURL = imageURL;
        this.imageURLThumb = imageURLThumb;
    }
}
```

Listing 2: From test table to code 2

References

[Ambler08] S. W. Ambler, April 2008, Introduction to Test Driven Design (TDD). http://www.agiledata.org/essays/tdd.html.

[Beck08] K. Beck, April 2008, XProgramming.com - an Agile Software Development Resource. http://www.xprogramming.com.

[Fit] W. Cunningham, et al., April 2008, Fit: Framework for Integrated Test. http://fit.c2.com.

[FitClips] IBM, Vishnu Vettrivel, April 2008, FIT and Eclipse: Testing with the Extended FIT Eclipse plug-in. http://www.ibm.com/developerworks/aix/library/au-fiteclipse/index.html.

[FitFix] FitNess Useful Fixtures, April 2008, http://fitnesse.org/UsefulFixtures

[FitGrace] Improving Readability using Graceful Names, April 2008, http://fitnesse.org/FitNesse.GracefulName.

[FitLib] FitLibrary, April 2008, http://fitnesse.org/FitLibrary.

[FitNesse] FitNesse, April 2008, www.fitnesse.org.

[FitPro] SourceForge.net, April 2008, FITpro - Acceptance Test Solutions. http://sourceforge.net/projects/fitpro/.

[jRap] John Steven et al. jRapture: A Capture/Replay Tool for Observation-Based Testing. Intl. Symp. on Software Testing and Analysis, Portland, Oregon, August 2000, pp. 158-167.

[JUnit] JUnit.org Resource for Test Driven Development: JUnit.org, April 2008, http://www.junit.org.

[Mens07] Tom Mens, Serge Demeyer: Software Evolution (pp173), Springer 2007.

[Mock08] Mock Object Authors, April 2008, Mock Objects. http://www.mockobjects.com.

[Mug05] R. Mugridge, W. Cunningham: FIT for Developing Software. Framework for Integrated Tests. Prentice Hall International, 2005

[Otting08] Tim Ottinger's Blog on FitNesse, April 2008, http://tottinge.blogsome.com/2008/02/01/fitnesse-frustration/.

[PayPal] PayPal: Online Payment, Merchant Account - PayPal. April 2008, https://www.paypal.com.

[SunPet] Sun Microsystems, Inc., April 2008, Java™ Pet Store Demo. https://blueprints.dev.java.net/petstore.

Beweisen statt Testen – Höhere Zuverlässigkeit durch die Java Modeling Language

Unser tägliches Leben wird immer stärker von Software durchdrungen; daher ist die Korrektheit von Software von fundamentaler Bedeutung. Gleichzeitig wird Software immer komplizierter; daher sind mächtige Methoden zum Nachweis dieser Korrektheit erforderlich. Der wichtigste Ansatz in dieser Sache – der mathematische Beweis der Korrektheit – gewinnt zusehends an Bedeutung für die industrielle Praxis. Im Folgenden werden die grundlegenden Ideen zu diesem Ansatz erläutert sowie eine aktuelle Sprache, die Java Modeling Language, zusammen mit ihren wichtigsten Werkzeugen zur Realisierung dieses Ansatzes vorgestellt.

Edgar Lederer | edgar.lederer@fhnw.ch

Das vielleicht vordringlichste Problem der Informatik ist die Korrektheit von Software – was nützt die benutzerfreundlichste, effizienteste und innovativste Software, wenn sie nicht richtig funktioniert? Die meisten Anwenderinnen und Anwender haben sich schon oft über die Eigentümlichkeiten ihres PCs oder ihrer Büroprogramme die Haare gerauft; das einzige, worauf man sich wirklich verlassen kann, ist, etwas überspitzt formuliert, dass sie im entscheidenden Moment ihren Dienst versagen. Das ist ärgerlich, aber immerhin nicht tödlich. Anders sieht die Situation aus, wenn Software zur Steuerung von Flugzeugen, Eisenbahnen und Autos, oder aber Atomkraftwerken, militärischen Einrichtungen und medizinischen Geräten eingesetzt wird. Hier kann ein Fehlverhalten leicht zur Gefahr für Leib und Leben werden (ein berühmtes Beispiel ist das medizinische Bestrahlungsgerät Therac-25, dessen Fehlverhalten zwischen 1985 und 1987 zu sechs Unfällen führte, von denen drei tödlich endeten); solche Software bezeichnet man daher auch als *sicherheitskritisch* (safety-critical). Ebenfalls äusserst heikel ist es, wenn Software Finanztransaktionen oder unbemannte Raketen steuert. Ein Fehlverhalten solcher Software kann zum Scheitern der Mission führen, für deren Durchführung eben diese Software benötigt wird (ein berühmtes Beispiel ist der fehlgeschlagene Jungfernflug der Trägerrakete Ariane 5 von 1996); solche Software bezeichnet man daher auch als *missionskritisch* (mission-critical).

Korrekte Software

Was bedeutet aber genau, dass Software *korrekt* ist? Ein Programm beispielsweise, das den Sinus einer gegebenen reellen Zahl perfekt ausrechnet, ist dennoch nicht korrekt, wenn die Aufgabe des Programms darin besteht, die Quadratwurzel aus der Zahl zu ziehen. Wir sehen also, dass Korrektheit kein absoluter, sondern nur ein relativer Begriff ist: Eine Software kann nur gegenüber den Anforderungen, die an sie gestellt werden, korrekt sein – nicht aber *per se*.

Um bei der Entwicklung der Software stets ein klares Ziel vor Augen zu haben, erweist es sich als äusserst zweckmässig, diese Anforderungen schriftlich festzuhalten, und zwar *bevor* mit dem Schreiben des eigentlichen Programms begonnen wird. Oft wird das so entstandene Dokument die *Spezifikation* des Programms genannt. Leider ist das Aufschreiben der Spezifikation in der Regel auch kein Kinderspiel: Haben wir wirklich alle Anforderungen aufgelistet (Vollständigkeit)? Widersprechen sich unsere Anforderungen auch nicht (Widerspruchsfreiheit)?

Mit der Spezifikation zerfällt die Frage nach der Korrektheit eines Programms also in zwei Teile:

- Erfüllt die Spezifikation unsere Anforderungen?
- Erfüllt das Programm unsere Spezifikation?

Die Überprüfung der ersten Frage wird als *Validierung* bezeichnet und oft durch die Frage «Bauen wir das richtige Produkt?» apostrophiert, während die Überprüfung der zweiten Frage als *Verifikation* bezeichnet wird und durch die Frage «Bauen wir das Produkt richtig?» charakterisiert wird.

Im Idealfall erfolgt die Validierung, bevor mit dem Schreiben des eigentlichen Programms begonnen wird. Wie kann dies geschehen? Möglich und gängig sind das sorgfältige Review der Spezifikation im Team oder die Entwicklung und Anwendung eines Prototypen. Besonders attraktiv ist es auch, die Spezifikation in einer sehr hohen, meist deklarativen (funktionalen oder logischen) Programmiersprache zu schreiben. Dann ist die Spezifikation nämlich direkt ausführbar. Dies ist für die Endanwendung meist nicht effizient

genug, dafür ist aber das Schreiben der ausführbaren Spezifikation vergleichsweise schnell erledigt. Nun können praktische Erfahrungen beim Ausführen der Spezifikation gesammelt werden, durch die die Spezifikation schrittweise verbessert werden kann, bevor noch eine einzige Zeile in der endgültigen Programmiersprache geschrieben ist. (Es ist aber bei diesem Ansatz unbedingt zu beachten, dass die Ausführbarkeit der Spezifikation einen Verlust an Abstraktion nach sich ziehen kann, was der Idee einer Spezifikation genau zuwiderläuft.)

Software-Verifikation

Schwerpunkt dieses Artikels ist aber nicht die Validierung, sondern die Verifikation. In der industriellen Praxis erfolgt die Verifikation in den meisten Fällen mittels Testens: Das zu verifizierende Programm wird für geeignet ausgewählte Testfälle ausgeführt, und die dabei erhaltenen Ergebnisse werden mit den erwarteten Ergebnissen verglichen; welche Ergebnisse erwartet werden, sagt die Spezifikation. (Die Instanz, welche die richtigen Ergebnisse voraussagt, nennt man allgemein auch *Test-Orakel*; hier ist also die Spezifikation das Test-Orakel.) Welche Schlüsse kann man nun aus den Tests ziehen? Wenn man bedenkt, dass ein Programm dann und nur dann korrekt ist, wenn es *für alle denkbaren* Eingaben richtig funktioniert, die folgenden: Weichen erhaltenes und erwartetes Ergebnis für irgendeinen der ausgewählten Testfälle voneinander ab, so weiss man definitiv, dass das Programm fehlerhaft ist; stimmen sie hingegen für alle ausgewählten Testfälle überein, so weiss man zwar, dass das Programm für eben diese Testfälle korrekt funktioniert, aber über alle übrigen, nicht getesteten Fälle (und dies sind «astronomisch» viel mehr als die getesteten), weiss man leider absolut nichts. Es gibt eben keinen logischen Schluss von dem Speziellen auf das Allgemeine («Weil ich fünf weisse Schwäne gesehen habe, sind alle Schwäne weiss.»), und das Stetigkeitsargument aus den empirischen Wissenschaften wie der Physik («Weil die Brücke sowohl unbelastet als auch stark belastet hält, hält sie auch bei mittlerer Belastung.») ist bei Programmen nicht anwendbar, da Programme eben nicht auf stetigen, sondern auf diskreten Strukturen basieren. Diese Überlegungen führen zu der etwas ernüchternden und bereits 1969 von Edsger W. Dijkstra formulierten Erkenntnis, dass man durch Testen immer nur die Anwesenheit, niemals aber die Abwesenheit von Fehlern in Programmen zeigen kann. (Es ist in diesem Zusammenhang interessant festzuhalten, dass diese Erkenntnis konsistent ist mit der *Falsifikationstheorie* des Philosophen Sir Karl Popper, derzufolge sich wissenschaftliche Theorien immer nur falsifizieren, niemals aber verifizieren lassen; solange eine Theorie noch nicht falsifiziert ist, kann man lediglich sagen, dass sie sich bisher *bewährt* hat.) Die in der Industrie häufig durchgeführten *Akzeptanztests* zeigen auch nicht etwa, dass ein Programm korrekt ist, sondern lediglich – wie der Name auch treffend ausdrückt – dass es vom Auftraggeber *akzeptiert* wird, er also bereit ist, die Rechnung des Auftragnehmers zu begleichen. Auch das Unwort «ausgetestet» ist mehr als gefährlich, da es durch seine Vorsilbe «aus» suggeriert, dass alle denkbaren Fälle getestet wurden.

Glücklicherweise gibt es eine weit aussagekräftigere Methode als die des Testens, Programme gegenüber ihrer Spezifikation zu verifizieren, nämlich die *mathematische* oder *deduktive* Methode. Möchten wir in der Mathematik beispielsweise wissen, dass die binomische Formel $(a + b)^2 = a^2 + 2ab + b^2$ für alle Zahlen a und b gilt, so verschaffen wir uns dieses Wissen ja auch nicht dadurch, dass wir einige konkrete Zahlen für a und b einsetzen (z. B. $a = 2$, $b = 3$ und $a = 17$, $b = -5$), sondern indem wir mathematische Gesetze anwenden:

$$
\begin{aligned}
&(a + b)^2 \\
&= \text{// Definition von Quadrat} \\
&(a + b) \cdot (a + b) \\
&= \text{// Distributivgesetz} \\
&a \cdot (a + b) + b \cdot (a + b) \\
&= \text{// Distributivgesetz zweimal} \\
&a \cdot a + a \cdot b + b \cdot a + b \cdot b \\
&= \text{// Kommutativgesetz der Multiplikation} \\
&a \cdot a + a \cdot b + a \cdot b + b \cdot b \\
&= \text{// neutrales Element 1 der Multiplikation} \\
&a \cdot a + 1 \cdot (a \cdot b) + 1 \cdot (a \cdot b) + b \cdot b \\
&= \text{// Distributivgesetz} \\
&a \cdot a + (1 + 1) \cdot (a \cdot b) + b \cdot b \\
&= \text{// Definition von 2 und Definition von Quadrat} \\
&a^2 + 2ab + b^2
\end{aligned}
$$

Nun wissen wir, zweifelsfrei und nach nur endlich vielen Schritten, dass unsere binomische Formel für alle, also unendlich viele, Zahlen gilt. Es ist eine der faszinierendsten Eigenschaften der Mathematik, dass man Aussagen über unendlich viele Gegenstände nach nur endlich vielen Überlegungen erhält, und das mit (bis auf Rechenfehler) absoluter Sicherheit. Warum sollten wir dieses mächtige Werkzeug nicht auch für Programme einsetzen?

Dies setzt natürlich voraus, dass Programme und Spezifikationen mit mathematischer Präzision behandelt werden können. Insbesondere reicht es dann nicht mehr aus, Spezifikationen nur umgangssprachlich zu beschreiben; eine formale Beschreibung ist stattdessen erforderlich, und dafür wiederum werden formale Spezifikationssprachen benötigt. Des Weiteren ist es erforderlich, den Programmiersprachen und den Spezifikationssprachen mathematisch exakt definierte Bedeutungen zu geben. Schliesslich – und das erahnt man schon bei unserem trivialen Beispiel mit der binomischen Formel – sind mathematische Korrektheitsbeweise für kom-

plexe Programme extrem aufwändig und damit selbst fehleranfällig. Daher muss die Entwicklung solcher Beweise wiederum durch geeignete (fehlerfreie!) Programme unterstützt werden. All diese Anforderungen sind aber inzwischen durch jahrzehntelange weltweite Forschungsanstrengungen prinzipiell und für relevante Fallstudien erfüllt worden – das Hauptaugenmerk der Forschung liegt aktuell auf der Umsetzung dieser Methoden in die industrielle Praxis. Nicht zuletzt Microsoft Research arbeitet intensiv an der Umsetzung dieser Ideen; sowohl Microsofts *Spec#*, eine Sprache sehr ähnlich wie das hier vorgestellte JML, als auch Microsofts *Static Driver Verifier*, ein Programm zur statischen Verifikation bestimmter Eigenschaften von insbesondere fremd entwickelten Gerätetreibern, geben ein beredtes Zeugnis davon ab.

Zusicherungen, Hoare-Tripel und Hoare-Logik

Aber was sind nun eigentlich die grundlegenden Ideen, um mathematische Beweise auch auf Programme anwenden zu können? Diese Ideen wurden erstmals 1969 von C.A.R. (nun Sir Tony) Hoare in einem Artikel [H69] vorgestellt (seinerseits inspiriert durch einen Artikel [F67] aus dem Jahre 1967 von Robert W. Floyd) und basieren auf Zusicherungen, Hoare-Tripeln und Hoare-Logik, wie im Folgenden kurz skizziert wird.

Eine *Zusicherung* ist ein boolescher Ausdruck, der in der Regel einige der deklarierten Variablen des Programms enthält. In einem gegebenen Zustand der das Programm ausführenden Maschine hat eine Zusicherung also immer einen der beiden Werte *wahr* oder *falsch*. Damit lässt sich eine Zusicherung aber auch auffassen als Beschreibung einer Menge von Zuständen, nämlich der Menge aller derjenigen Zustände, in denen die Zusicherung den Wert *wahr* hat.

Ein *Hoare-Tripel* ist nun ein Ausdruck der Form «{P} A {Q}» wobei *P* und *Q* Zusicherungen sind, und *A* eine Anweisung ist. *P* und *Q* nennt man dabei auch *Vorbedingung* bzw. *Nachbedingung* des Hoare-Tripels. Ein Hoare-Tripel heisst *gültig*, wenn die folgende zentrale Bedingung erfüllt ist:

> *Wird die Anweisung* A *in einem Zustand ausgeführt, in dem die Vorbedingung* P *wahr ist, und wird die Ausführung in endlicher Zeit beendet (geht also nicht in eine Endlosschleife), so ist die Nachbedingung* Q *im resultierenden Zustand erfüllt.*

Ein Beispiel für ein gültiges Hoare-Tripel ist also «{ $x = 5$ } $x := x + 1$ { $x > 5$ }»; ein Beispiel für ein ungültiges Hoare-Tripel hingegen «{ $x = 5$ } $x := x + 1$ { $x > 6$ }». Hier ein weiteres Beispiel für ein Hoare-Tripel:

> { *p* ist ein syntaktisch korrektes Java-Programm }
> *c* := Java-Compiler(*p*)
> { *c* ist der korrekte Bytecode für *p* }

Es ist offensichtlich, dass die Entwickler des Java-Compilers daran interessiert sind, dass dieses Hoare-Tripel gültig ist. Aber wie lässt sich entscheiden, ob ein Hoare-Tripel gültig ist oder nicht? Dies ist möglich mittels der *Hoare-Logik*, einem Satz von Regeln, die es gestatten, aus der Gültigkeit von Hoare-Tripeln mit gegebenen Anweisungen auf die Gültigkeit von Hoare-Tripeln mit daraus zusammengesetzten Anweisungen zu schliessen. Betrachten wir als Beispiel die wichtigste dieser Regeln, nämlich die für Schleifen:

> *Ist das Hoare-Tripel* «{ I and B } A { I }» *gültig, so ist auch das Hoare-Tripel* «{ I } while B do A end { I and not B }» *gültig.*

Diese Regel ist unmittelbar einleuchtend: Wenn die Ausführung des Schleifenrumpfes *A* in einem Zustand, der die Zusicherung *I* und die Schleifenbedingung *B* erfüllt, zu einem resultierenden Zustand führt, der *I* immer noch erfüllt, so bleibt *I* auch noch nach beliebig oft wiederholter Ausführung des Schleifenrumpfes erfüllt – insbesondere auch noch nach der allerletzten. Nach dieser allerletzten Ausführung ist aber gleichzeitig die Schleifenbedingung nicht mehr erfüllt – ansonsten wäre der Schleifenrumpf ja nochmals ausgeführt worden.

Die Zusicherung *I* wird also durch Ausführung des Schleifenrumpfes nicht zerstört; man bezeichnet sie daher auch als *Invariante* der Schleife. Invarianten beschreiben die eigentliche Idee, die hinter einer Schleife verborgen liegt, und stellen den kreativen Teil bei der Entwicklung einer Schleife dar. Entsprechend ist es auch nicht verwunderlich, dass Invarianten im Allgemeinen nicht automatisch hergeleitet werden können (für einfachere Fälle lassen sich inzwischen jedoch wahrscheinliche Invarianten automatisch bestimmen; siehe dazu weiter unten). Von Seiten des Unterrichts ist noch folgendes anzumerken: Das Konzept der Invarianten gehört zu den für das grundlegende Verständnis der Informatik mächtigsten und daher auch wichtigsten Konzepten; umso trauriger ist es, dass dieses Konzept auch heute noch an vielen Universitäten im Unterricht nur stiefmütterlich behandelt wird [G06]; an unserer Schule findet es jedoch im Modul «Algorithmen und Datenstrukturen» einen breiteren Raum.

Design by Contract

Diese grundlegenden Ideen wurden bald auch auf die objektorientierte Programmierung angewendet; besonders einflussreich ist dabei die juristische Metapher des *Design by Contract* von Bertrand Meyer [M97]. Im Design by Contract gehört zu jeder Methode einer Klasse eine Vorbedingung und eine Nachbedingung; diese bilden einen Kontrakt zwischen dem Aufrufer der Methode und der Methode selbst:

- Der Aufrufer *verpflichtet* sich, die Methode nur in solchen Zuständen aufzurufen, die die

Vorbedingung erfüllen; entsprechend ist die Methode *berechtigt* davon auszugehen, dass die Vorbedingung erfüllt wird.
- Die Methode *verpflichtet* sich (in den Fällen, in denen die Vorbedingung erfüllt ist), einen Zustand zu erzeugen, der die Nachbedingung erfüllt; entsprechend ist der Aufrufer *berechtigt* (in den Fällen, in denen die Vorbedingung erfüllt ist) davon auszugehen, dass die Nachbedingung erfüllt wird.

Zusätzlich zu diesen Vor- und Nachbedingungen gibt es im Design by Contract Konsistenzbedingungen, die während der gesamten Lebensdauer eines Objektes einer Klasse erfüllt sein müssen; diese werden entsprechend als *Klasseninvarianten* bezeichnet.

Leider reichen diese grundlegenden Ideen noch nicht aus, um voll ausgebildete objektorientierte Programme zu spezifizieren und zu verifizieren. Weitere wesentliche Punkte sind die folgenden: (1) *Exceptions*: Das Verhalten bei Ausnahmen muss spezifiziert werden können. (2) *Frame problem*: Es muss spezifiziert werden können, welche Speicherzellen durch Ausführung des Programms *nicht* geändert werden dürfen, und das unter Berücksichtigung von Sichtbarkeit und Subtyping. (3) *Call-back problem*: Klasseninvarianten gelten stets zu Beginn und am Ende einer Methode, aber zunächst keineswegs notwendigerweise auch dazwischen. Ruft nun eine Methode eine Methode derselben Klasse auf (*call back*), so wird diese aufgerufene Methode im Allgemeinen in einem Zustand aufgerufen, in dem die Invariante verletzt ist, womit sich natürlich keinerlei Voraussagen mehr über das Verhalten der aufgerufenen Methode machen lassen. Stellt man aber sicher, dass die Invariante auch vor und nach jedem Methodenaufruf gilt, so ist das Problem gelöst; leider ist diese Lösung sehr restriktiv – andere Ansätze sind Gegenstand aktueller Forschung. (4) *Behavioral subtyping*: Das Verhalten eines Subtyps muss stets mit dem Verhalten seines Supertyps übereinstimmen (*behavioral subtyping*), so dass anstelle eines Objektes des Supertyps gefahrlos auch immer ein Objekt des Subtyps eingesetzt werden kann (*Liskov's substitution principle*). Dies lässt sich dadurch erreichen, dass ein Subtyp alle Spezifikationen seines Supertyps erbt (*specification inheritance*), also alle Klasseninvarianten des Supertyps und die Kontrakte aller überschriebenen Methoden. (5) *Data abstraction*: Besonders wesentlich ist die Spezifikation des Verhaltens aller Klassen, die ein gegebenes Interface implementieren; diese Spezifikation muss also im Interface selbst stehen. Dort aber sind keine Felder deklariert, so dass sich eigentlich gar keine Kontrakte und Klasseninvarianten formulieren lassen. Dieses Problem lässt sich aber lösen, indem man durch Deklaration von *Modell-Feldern* im Interface einen abstrakten Zustand schafft; mit diesen lassen sich dann Kontrakte und Invarianten formulieren. Dass eine Klasse ein Interface dann korrekt implementiert, lässt sich durch Relationen in der Klasse ausdrücken, die den konkreten Zustand der Klasse mit dem abstrakten Zustand des Interfaces in Beziehung setzen.

Java Modeling Language

Eine konkrete Spezifikationssprache, die alle diese Probleme löst (und zwar in der eben angedeuteten Art und Weise), ist die Java Modeling Language (JML[1]) [BC05, CKLP06, LBR06]. JML dient der Spezifikation des Verhaltens von voll ausgebildeten objektorientierten Programmen in Java. Wesentlich dabei ist, dass JML für den industriellen Einsatz gedacht ist, dass JML-Programmierer keine Experten für mathematische Methoden sein müssen, und dass für JML eine Reihe verschiedenartiger Werkzeuge zur Verfügung steht. Von diesen werden wir später noch einige etwas näher ansehen; vorerst aber betrachten wir ein kleines Beispiel einer JML-annotierten Java-Methode: Abbildung 1 zeigt die bekannte binäre Suche. Dieses Beispiel ist besonders reizvoll, da einerseits die operationelle Idee hinter der binären Suche sofort einleuchtet («Ein von links nach rechts aufsteigend sortiertes Feld wird in der Mitte geteilt: ist das gesuchte Element grösser als das Element in der Mitte, kommen sämtliche Elemente links der Mitte und die Mitte selbst nicht mehr in Frage; ansonsten fallen die Elemente rechts der Mitte flach. Die verbleibenden Elemente werden nach demselben Verfahren durchsucht, bis kein Element mehr verbleibt.»), es andererseits aber ziemlich schwierig ist, das Verfahren wirklich korrekt zu implementieren (die Leserin oder der Leser mögen es einmal selbst versuchen!). Unsere Implementierung basiert auf den Überlegungen in [B03].

Formuliert in JML sind die Vorbedingung, die Nachbedingung und die Schleifeninvariante. (Zusätzlich ist die Methode als *pure* deklariert, was bedeutet, dass sie keine Seiteneffekte hat; ausserdem enthält sie noch eine Zusicherung (Schlüsselwort *assert*) am Ende der Schleife.) Die *Vorbedingung* (Schlüsselwort *requires*) verlangt, dass das Feld nicht null ist, dass seine Länge mindestens 0 ist (dient lediglich der Verdeutlichung, dass auch die Länge 0 erlaubt ist), und dass das Feld aufsteigend sortiert ist, wobei auch gleiche Werte vorkommen dürfen. Die *Nachbedingung* (Schlüsselwort *ensures*) stellt sicher (jedenfalls solange die Vorbedingung erfüllt ist), dass der gesuchte Index zwischen 0 und der Länge des Feldes (jeweils einschliesslich) liegt, und dass alle Elemente des Feldes links vom gesuchten Index (ausschliesslich des Index) kleiner sind als der gesuchte Wert, und alle Elemente rechts davon (einschliesslich des In-

1 http://www.cs.ucf.edu/~leavens/JML/

```
/*@ requires a != null && a.length >= 0
  @   && (\forall int i; 0 <= i && i < a.length-1; a[i] <= a[i+1]);
  @ ensures 0 <= \result && \result <= a.length
  @   && (\forall int i; 0 <= i && i < \result; a[i] < x)
  @   && (\forall int i; \result <= i && i < a.length; a[i] >= x);
  @*/
static /*@ pure @*/ int binarySearch(int[] a, int x) {
    int l = 0;
    int r = a.length - 1;
    /*@ loop_invariant 0 <= l && l <= r+1 && r <= a.length - 1
      @   && (\forall int i; 0 <= i && i < l; a[i] < x)
      @   && (\forall int i; r < i && i < a.length; a[i] >= x);
      @*/
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (a[m] < x) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    //@ assert l == r+1;
    return l;
}
```

Abbildung 1: JML-annotierte Java-Methode zur bekannten binären Suche

dex) grösser oder gleich dem Wert. Die *Schleifeninvariante* (Schlüsselwort *loop_invariant*) sagt, dass vor und nach jedem Schleifendurchlauf alle Elemente des Feldes links von einem Index *l* kleiner sind als der gesuchte Wert, und alle Elemente rechts von einem Index *r* grösser oder gleich dem Wert sind, und (hauptsächlich) dass der linke Index den rechten Index „überholen" kann, aber nur um maximal eine Position. An der Schleifeninvariante ist besonders gut zu erkennen, dass das Feld insgesamt in drei Teile unterteilt wird (Teil 1: Elemente, von denen man schon weiss, dass sie kleiner sind als der gesuchte Wert; Teil 2: Elemente, von denen man noch nichts weiss; und Teil 3: Elemente, von denen man schon weiss, dass sie grösser oder gleich dem gesuchten Wert sind). Relevant sind also in Wirklichkeit *drei* Teile, obwohl das Verfahren *binäre* Suche heisst! Am Anfang des Verfahrens ist noch kein Wissen vorhanden: Teile 1 und 3 sind noch «leer», während Teil 2 das gesamte Feld abdeckt. Am Ende des Verfahrens hingegen, also wenn der linke den rechten Index überholt hat, ist das gesamte gewünschte Wissen vorhanden: Teil 2 ist nun leer, während die Teile 1 und 3 zusammen das gesamte Feld abdecken – womit die Suche abgeschlossen ist.

JML-Werkzeuge

Nun aber zu den Werkzeugen für JML: Das Werkzeug *jmlc* ist ein Compiler, der JML-annotierte Java-Programme in Bytecode übersetzt; die JML-Zusicherungen werden dabei in Code zu ihrer Überprüfung zur Laufzeit übersetzt. Wird eine Zusicherung zur Laufzeit des Programms verletzt, so wird eine aussagekräftige Fehlermeldung erzeugt; andernfalls bleibt die Zusicherung ohne Wirkung.

Das Werkzeug *jmlunit* verwendet die JML-Spezifikationen als Test-Orakel; die Testfälle selbst müssen dabei weiterhin von Hand festgelegt werden.

Das Werkzeug *ESC/Java2* ist ein *Extended Static Checker*, also ein Werkzeug zur statischen Analyse des Programmtextes; das Programm braucht dazu also nicht ausgeführt zu werden. ESC/Java2 generiert aus einem JML-annotierten Java-Programm automatisch eine Liste möglicher Schwachstellen des Programms. Dabei ist es weder das Ziel, dass die Liste alle Fehler des Programms enthält, noch dass alle Einträge in der Liste tatsächlich Fehler sind, sondern lediglich, dass die Liste möglichst viele der Fehler des Programms enthält, und dass möglichst viele der Einträge in der Liste tatsächlich Fehler sind, so dass sich der Aufwand für das sorgfältige Studium der Liste unter dem Strich auszahlt.

Das Werkzeug *JACK* ist ein Verifizierer und kommt damit ebenfalls mit dem Programmtext alleine aus. Dieser erzeugt aus einem JML-annotierten Java-Programm einen Satz von Verifikationsbedingungen; sind diese allesamt gültig, ist das Java-Programm korrekt. Diese Verifikationsbedingungen werden dann weitergeleitet an einen automatischen Theorembeweiser, der versucht, diese Bedingungen zu beweisen. Wie aus der theoretischen Informatik bzw. der Logik bekannt, ist es im Allgemeinen aber gar nicht möglich, beliebige Aussagen automatisch zu beweisen. In der Praxis stellt es sich aber heraus, dass doch bis zu 90% der Verifikationsbedingungen automatisch bewiesen werden können. Die übrigen, nicht bewiesenen Bedingungen werden dann dem Anwender zur weiteren Untersuchung vorgelegt; dieser kann sich dann beispielsweise um Beweise mit Hilfe eines interaktiven Theorembeweisers oder

um klassische mathematische Beweise mit Papier und Bleistift bemühen. JACK ist als Plug-in für Eclipse realisiert.

Das Werkzeug *Daikon* führt Java-Programme auf ausgewählten Eingaben aus und erzeugt aus den dabei entstandenen Ausführungsspuren wahrscheinliche Invarianten in Form von JML-Spezifikationen. Damit lässt sich in vielen der einfacheren Fälle der Aufwand zur Entwicklung der JML-Spezifikationen verringern; auch ist das Werkzeug nützlich für die nachträgliche Spezifikation von Legacy-Software, und insbesondere für das Refactoring von Legacy-Software, da beim Refactoring die Spezifikationen weiterhin eingehalten werden müssen – aber dazu ist es praktisch, die Spezifikationen erst einmal zu kennen.

Speziell in unserem Forschungsprojekt *ProMedServices*, bei dem es unter anderem gerade um das Refactoring von Teilen eines existierenden medizinischen Informationssystems geht, wollen wir den Einsatz von JML an einem praktischen Beispiel untersuchen. Des weiteren ist zu bemerken, dass ein grosser Anteil des Interesses an JML und verwandten Technologien aus dem Bereich Java-Smart-Card kommt, da darauf häufig Finanzanwendungen für den Masseneinsatz, also missionskritische Anwendungen, realisiert werden. Könnte man mit JML und verwandten Technologien hohe Zuverlässigkeit auch für andere mobile Geräte sicherstellen, so könnte man auch diese für einen solchen Einsatz in Betracht ziehen.

Seit Anbeginn der Informatik wird der mathematische Nachweis der Korrektheit von Software systematisch erforscht. Inzwischen sind viele der grundlegenden Fragen geklärt und viele leistungsfähige Werkzeuge entwickelt, so dass wir nun am Beginn des grossflächigen industriellen Einsatzes dieser Methoden stehen. Bis dahin sind noch etliche Jahre intensivster Arbeit in Theorie, Praxis und Ausbildung erforderlich – das Ziel, dass man sich auf Software auch im entscheidenden Moment verlassen kann, ist diese Arbeit aber allemal wert.

Referenzen

[B03] Roland Backhouse. Program Construction: Calculating Implementations from Specifications. John Wiley & Sons, 2003.

[F67] Robert W. Floyd. Assigning Meanings to Programs. Mathematical Aspects of Computer Science (Proceedings of Symposia in Applied Mathematics, Vol. 19), American Mathematical Society, 1967.

[G06] David Gries. What Have We Not Learned about Teaching Programming? Computer, Volume 39, Number 10, 2006.

[H69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming, Communications of the ACM, Volume 12, Number 10, 1969.

[M97] Bertrand Meyer. Object-Oriented Software Construction. Second Edition, Prentice Hall PTR, 1997.

[BC05] Lilian Burdy, Yoonsik Cheon et al. An overview of JML tools and applications. Int. Journal on Software Tools for Technology Transfer, Volume 7, Number 3, 2005.

[CKLP06] Patrice Chalin, Joseph R. Kiniry et al. Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2. Formal Methods for Components and Objects: 4th International Symposium, FMCO 2005, 2006.

[LBR06] Gary T. Leavens, Albert L. Baker, Clyde Ruby. Preliminary Design of JML: A Behavioral Interface Specification Language for Java. ACM SIGSOFT Software Engineering Notes, Volume 31, Number 3, 2006.

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

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