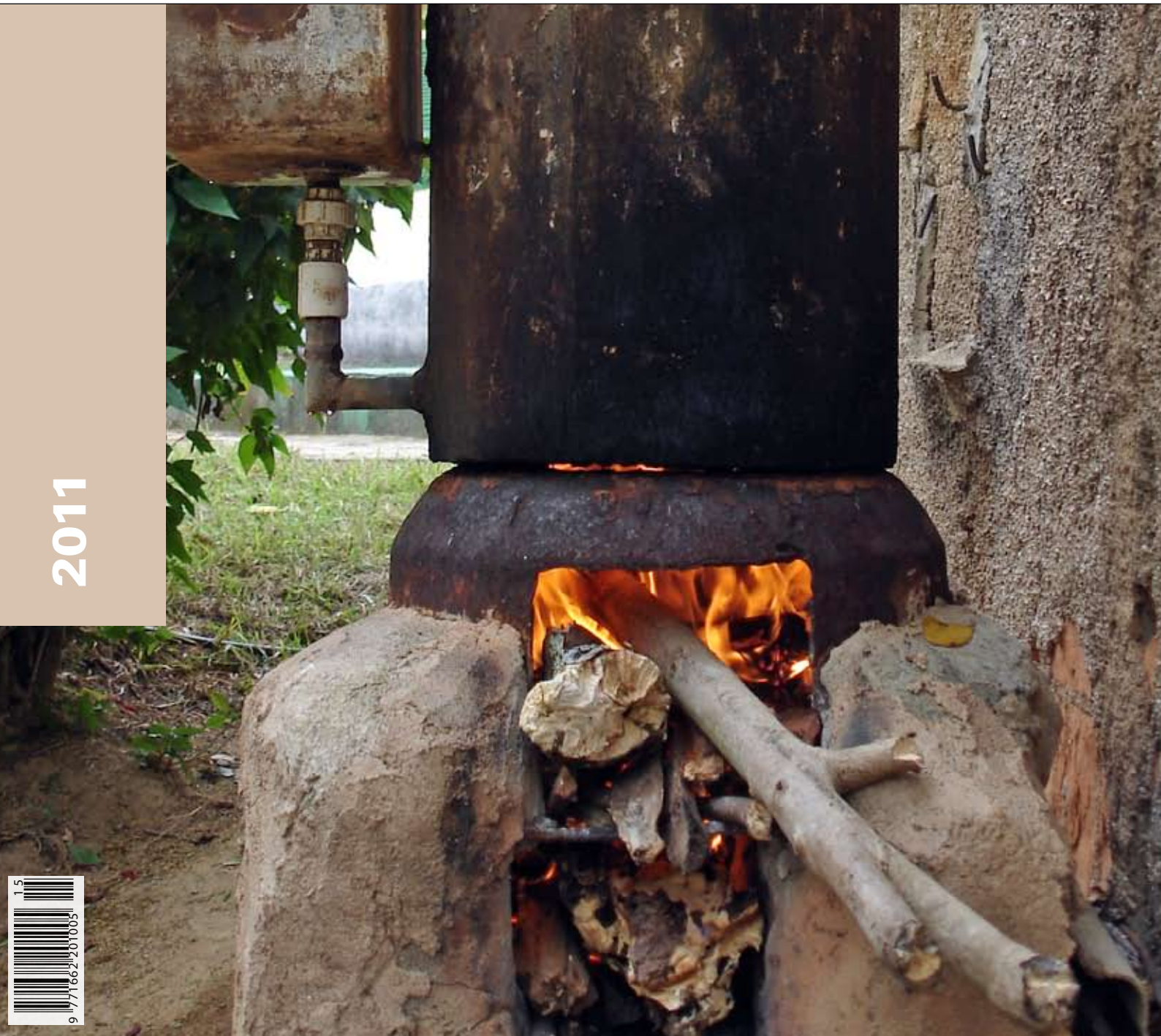
Fachhochschule Nordwestschweiz
Hochschule für Technik

Institut für Mobile und Verteilte Systeme

FOKUS REPORT



2011

9 771662 201005 15

Editorial

Der PC wird dreissig. Oft ist er totgesagt worden und trotz allen Hypes und Flops hat er sich über all die Jahre bewähren können. Fast alle Lebensbereiche hat er durcheinander gewirbelt und kaum einen Stein auf dem andern belassen. Das analoge Leben hat sich durch ihn und mit ihm zum digitalen verändert. Wenige Veränderungen in der Zeit waren so nachhaltig wie diese.

Bereits seit seiner Geburt steht er im Spannungsfeld zwischen Zentralisten und Individualisten. Ursprünglich als Ergänzung zu den hauseigenen Mainframes gedacht, hat er sich schnell eine eigenständige Daseinsberechtigung erwirkt und dies nicht nur als Ersatz des eleganten Kugelkopfes. Schon früh haben Tüftler, Entwickler und Visionäre ständig neue Anwendungsgebiete für den persönlichen Arbeitsrechner identifiziert und mit jeder neuen PC-Generation sind wieder weitere Visionen zur Realität geworden. Dieser Umbruch hält bis heute an und keiner weiss, wie lange es noch so weiter gehen wird.

In all diesen Jahren haben sich die Zentralisten immer wieder das Ende des PCs herbeigesehnt: das dumme Terminal, der Netzcomputer, Software as a Service sind nur ein paar der Versuche, welche sich die Server-Anbieter ausgedacht haben, um sich ein grösseres Stück des weltweiten Rechnerkuchens abzuschneiden. Natürlich werden gute Argumente für die zentralisierten Dienste hervorgebracht, wie reduzierte Plattformabhängigkeit, vereinfachte und billigere Wartung oder verbesserte Sicherheit. Dass dabei oft die Bedürfnisse der individuellen Endbenutzer auf der Strecke bleiben, ist leider mehr als eine Binsenwahrheit. Technologien der Individualisten, wie zum Beispiel Peer-to-Peer, vermaschte WLANs oder PDAs, sind den Zentralisten oft ungeheuer, erschweren ihre ICT-Aufgabe und werden primär als Sicherheitsrisiko und grossen Kostenfaktor betrachtet.

Mal sind die Individualisten am Zug und propagieren beispielsweise das Smartphone als persönlichen Computer, der die Daten dort verarbeiten kann, wo sie anfallen, dann sind die Zentralisten wieder dran und schwärmen von weissen Clouds am azurfarbenen Himmel, die endlich alle Sorgen der geplagten Datenverarbeiter im Nebel vergessen machen soll. All diese Trends haben die Hard- und Software des PCs beeinflusst und weitergebracht. Und so steht er heute immer noch so rüstig und vital da, wie vor drei dreissig Jahren.

Dass Individualisten und Zentralisten auf keinen fall disjunkt sein müssen, führt uns der Suchriese fast täglich vor. Mit dem Kauf eines Handy-Herstellers in den USA demonstriert er deutlich, dass er alles dransetzen will, um die desginverliebten i-Kunden zu einem Wechsel vom eleganten

Inhalt

Impressum

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

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

Redaktion: Prof. Dr. Christoph Stamm
Layout: Thekla Müller-Schenker
Erscheinungsweise: jährlich
Druck: jobfactory Basel
Auflage: 200

ISSN: 1662-2014

Apfel zum lächerlich anmutenden grünen Androiden zu bewegen. Nun, es gibt tatsächlich ein paar gute Argumente, die für den humanoiden Roboter sprechen. So zum Beispiel, dass Android eine offene Plattform ist, die es jedem erlaubt, eigene Programme in Java zu entwickeln und zu installieren, ohne dass sie vorgängig die Zensur überstehen müssen.

Dominik Gruntz beschreibt in seinem Artikel „Android API Levels", worauf Programmier zu achten haben, wenn sie Vorwärts- und Rückwärtskompatibilität mit den verschiedenen API Levels beachten wollen. Er beschreibt dies anhand einer Anwendung, welche für den Datenaustausch auch NFC verwenden kann. In einem zweiten Artikel vertieft er das Thema NFC und zeigt auf, wie NFC mit entsprechend ausgerüsteten Geräten genutzt werden kann. Wie ernst es dem Suchriesen in Sachen NFC ist, wird sich schon sehr bald weisen, wenn die ersten hauseigenen NFC Smartphones auf den Markt kommen.

Der Trend zu immer mehr integrierten Sensoren zeigt sich nicht nur an NFC. Längst haben GPS, Beschleunigungs-, Temperatur- und Feuchtigkeitssensoren ihren Platz im Smartphone gefunden. Dass die damit gemessenen Daten auch in der Ferne ihren Wert haben, ist sofort ersichtlich, wenn man an verteilte Messgeräte denkt. Sobald also Sensoren mit Telekommunikationstechnik verknüpft werden, entstehen flexibel einsetzbare Messgeräte, die ihre Daten an einen Server übermitteln und von dort aus auch ferngewartet werden können. Obwohl diese Idee überhaupt nicht neu ist, gibt es auf dem Markt kaum allzwecktaugliche Geräte für den Endverbrauchermarkt.

Matthias Krebs hat während seiner Masterarbeit einen Prototyp eines solchen allzwecktauglichen und mobil einsetzbaren Fernmessgerätes entwickelt. In seinem Artikel „Active Data Logger" stellt er das Gesamtkonzept vor und beschreibt dann vor allem das Kommunikationsprotokoll und die dazu passende Software, um die übertragenen Messdaten in einer zentralen Datenbank abzulegen. Sein Konzept erlaubt es, Filter auf dem Messgerät zu installieren, welche die Datenflut bereits vor der Fernübertragung ausdünnen oder konzentrieren. Trotzdem werden solche Messgeräte das Sammeln von Daten noch zusätzlich anregen.

Nicht nur für Fernmessgeräte, sondern auch für den Betrieb von Clouds sind grosse und vernetzte Datenzentren notwendig. Während früher oft von Rechenzentren die Rede war, weil man aus den weniger grossen Datenbeständen durch angewandte Mathematik Informationen gewinnen wollte, so steht heute meist das Sammeln und die Verwaltung der riesigen Datenmengen im Zentrum. An vielen Orten wachsen mittlerweile die Datenbestände schneller an, als sie wirklich verarbeitet und somit genutzt werden können. Die Datensammler verkommen zu den digitalen Messies. Dass dieses Sammeln und Verwalten auch mit einem erheblichen Energiebedarf verbunden ist, darf wohl kaum erstaunen.

Im Artikel „Energieoptimiertes Data Center" modellieren die Autoren Cyrill Grüter, Peter Gysel und Christoph Meier den Energieverbrauch von Datenzentren und beschreiben ein selber entwickeltes Werkzeug für Planer von solchen Zentren. Der aufmerksame Leser des Artikels wird bemerken, dass das Data Center nicht nur ein zentraler Pfeiler einer virtuellen Cloud sein kann, sondern dass die darin produzierte Wärme üblicherweise durch einen Kaltwassersatz abgeführt wird, welcher ausserhalb des Gebäudes zu einer geringen Dunstbildung führt und somit zur realen Wolkenbildung beiträgt. In den meisten Datenzentren, wo die Server primär mit Luft und erst sekundär die warme Luft mit Kaltwasser gekühlt wird, erhöht sich das kalte Wasser lediglich um ca. 5°C. Dieses lauwarme Wasser eignet sich nicht zum Heizen und müsste daher durch zusätzlichen Energiebedarf mittels einer Wärmepumpe auf eine höhere Temperatur gebracht werden. Erst wenn die elektronischen Geräte direkt mit Wasser gekühlt werden, lässt sich dieses Heisswasser sinnvoll in Heizsysteme und Fernwärmenetze einspeisen.

Wie Systemtests der Durchflussmesstechnik, welche oft in Heizsystemen zum Einsatz kommt, automatisiert werden können, berichten Anja Kellner und Martin Kropp im Beitrag „Testautomatisierung von Systemtests mit Continuous Integration & Co". Sie weisen darauf hin, dass sich Continuous Integration Umgebungen auch für die Realisierung von vollständig automatisierten Testinfrastrukturen für System- und Akzeptanztests im industriellen Umfeld eignen.

Die Vitalität der heutigen PCs beschreiben die Autoren Martin Schindler und Christoph Stamm im letzten Beitrag dieses Heftes, wo sie auf die Erkennung von fehlerhaften Abläufen in automatisierten, industriellen Prozessen eingehen. Mittels Hochgeschwindigkeitskameras werden mehrere hundert Bilder des Ablaufs pro Sekunde aufgenommen, an einen Rechner übertragen und dort von neu entwickelten, parallelen Algorithmen analysiert. Dabei spielen die hochparallelen GPUs eine zentrale Rolle. Mit speziellen APIs können sie für eigene Zwecke verwendet werden. Wie eine Parallelisierung sinnvollerweise gemacht wird und worauf bei der GPU-Programmierung sonst noch zu achten ist, wird ausführlich im Artikel „Highspeed Videoverarbeitung" dargelegt.

Wie sich unschwer feststellen lässt, bietet auch die diesjährige Ausgabe eine grosse thematische Breite, welche das eingangs beschriebene Spannungsfeld zwischen Zentralisten und Individualisten auf eindrückliche Art widerspiegelt.

Prof. Dr. Christoph Stamm
Forschungsleiter IMVS

NFC mit Android

Seit der Android Version 2.3 (Gingerbread) unterstützt Google neu auch NFC (Near Field Communication) und verleiht damit dem Mobile Payment und Mobile Ticketing neuen Schub. NFC ist eine kontaktlose Schnittstellentechnologie, welche die einfache und schnelle Kommunikation über ca. 2 cm zwischen RFID-Tags und einem speziell ausgerüsteten Mobiltelefon ermöglicht. Mit Release 2.3.3 wird auch das Schreiben von Tags und die Kommunikation zwischen zwei Mobiltelefonen über NFC unterstützt. In diesem Artikel geben wir einen Überblick über NFC und zeigen, wie diese Technologie über die Android APIs verwendet werden kann.

Dominik Gruntz | dominik.gruntz@fhnw.ch

Die Einsatzgebiete von NFC sind vielfältig. Daten, die auf einem NFC-Tag abgespeichert sind, können durch Berührung mit einem Mobiltelefon ausgelesen werden. Bei den ausgelesenen Daten kann es sich um eine URL, eine Telefonnummer, ein Bild, eine Geo-Koordinate, eine Visitenkarte (vCard) oder um applikationsspezifische Daten handeln. Die Daten lassen sich dann direkt auf dem Gerät weiterverarbeiten, ein mühsames Abtippen von URLs oder Fotografieren von QR-Tags entfällt. So wird es beispielsweise möglich sein, alleine durch Berührung mit dem Mobiltelefon die Bedienungsanleitung für den Geschirrspüler abzurufen und auf dem Display anzuzeigen. Tags an den Regalen im Supermarkt informieren über Inhalte von Lebensmitteln, und mit Hilfe der ausgelesenen Produktnummer könnten nützliche Zusatzinformationen aus unabhängigen Quellen angezeigt werden, zum Beispiel eine persönliche Allergiewarnung. Auch das Selfscanning (wie zum Beispiel passabene von Coop [Coop08] oder subito von Migros [Migr11]) könnte damit realisiert werden. NFC-Mobiltelefone wirken damit als Browser für das Internet der Dinge [MF10], d.h., es wird möglich, jedes Objekt, jede Person und jeden Ort mit Online-Dokumenten auf dem Web zu verknüpfen.

Die Anwendung von NFC ist vergleichbar mit dem Einlesen von 2D-Barcodes, ausser dass die Handhabung viel einfacher ist. Es muss nicht speziell eine Applikation wie der Barcode-Scanner gestartet werden, und die Kamera muss auch nicht auf ein Tag ausgerichtet werden. Es genügt, den NFC-Tag mit dem Mobiltelefon zu berühren. Dieses erkennt den Tag automatisch und startet eine Applikation zur Verarbeitung der übertragenen Daten. Mit einem NFC-Mobiltelefon kann im Prinzip auch ein Tag emuliert werden, welches von einem Lesegerät über die NFC-Schnittstelle ausgelesen oder beschrieben werden kann. Häufig handelt es sich dabei um Tags mit erweiterter Funktionalität, sogenannte Smartcards. Damit kann kontaktloses Bezahlen realisiert werden, wie es heute in vielen Ländern bereits in Form kontaktloser Kreditkarten genutzt wird.

All diese Anwendungsfälle haben wir im Rahmen des Projektes touch'n'pay (www.touchnpay.ch) umgesetzt und einen Hofladen so ausgerüstet, dass das Einkaufen mit NFC-fähigen Mobiltelefonen möglich wird. Die Regale haben wir mit NFC-Produktetiketten ausgezeichnet. Wenn der Kunde sein Mobiltelefon an ein solches Produktetikett hält, wird das Produkt automatisch auf seinem elektronischen Kassenzettel auf dem Mobiltelefon eingetragen. Sobald alle gewünschten Produkte in der Einkaufstasche liegen, muss nur noch ein Checkout-Tag berührt oder das Zahlen-Menü auf dem Mobiltelefon gewählt werden, um den Bezahlvorgang auszulösen. Das Mobiltelefon haben wir ausserdem verwendet, um auch ausserhalb der Öffnungszeiten den Zugang zum Hofladen zu ermöglichen. Das Türschloss greift dabei über NFC auf im Telefon abgespeicherte Daten zu [BG10].

In diesem Artikel geben wir eine kurze Einführung in NFC und zeigen dann, wie mit Android über NFC kommuniziert werden kann. Einen guten Überblick über die NFC Technologie findet man im kürzlich erschienenen Buch von Josef Langer und Michael Roland [LR10].

Betriebsarten

NFC unterscheidet folgende Betriebsarten:

- *Reader/Writer-Modus:* In dieser Betriebsart wird das NFC-Mobiltelefon zum Leser und kann passive NFC-Tags auslesen und mit Daten beschreiben.
- *Card-Emulation-Modus:* In dieser Betriebsart ist das NFC-Mobiltelefon passiv und emuliert ein Tag, typischerweise eine Smartcard. Ein RFID-Leser, z.B. ein Kassensystem oder ein Türschloss, greift auf das im NFC-Mobiltelefon emulierte Smartcard-Tag zu.
- *Peer-to-peer-Modus:* Die peer-to-peer Betriebsart ermöglicht es, Informationen zwischen zwei (aktiven) Geräten auszutauschen. Es

kann sich dabei um einen Gutschein handeln oder um gegenseitige Identifikationen, damit grössere Datenmengen danach einfach über Bluetooth oder über das Internet ausgetauscht werden können.

Android unterstützt ab der Version 2.3.4 den Reader/Writer-Modus und (eingeschränkt) den Peer-to-peer-Modus. Die Peer-to-peer Kommunikation erlaubt nur den Austausch einzelner Meldungen, so wie sie auch auf Tags gespeichert sind. Es ist jedoch nicht möglich, über NFC eine Socketverbindung zwischen zwei Android-Geräten aufzubauen.

Der Card-Emulation-Mode wird von den Android APIs nicht unterstützt. Der Zugriff auf das sogenannte Secure-Element (SE), auf welchem (wie in einer Smartcard) Daten und Programme sicher abgelegt werden können, ist damit aus einer Android-Applikation nicht möglich.

Die Hardware

NFC wird aktuell von den Smartphones *Nexus S, Galaxy SII* und *Galaxy Nexus* von Samsung unterstützt. Im Nexus S ist der PN544 NFC-Controller von NXP eingebaut [PN544]. Dieser Controller unterstützt alle Betriebsarten und enthält ein integriertes SE (SmartMX Security Chip), unterstützt jedoch über das Single Wire Protocol (SWP) auch den Zugriff auf ein auf einer erweiterten SIM Karte (UICC) abgelegtes SE. Das Galaxy SII enthält kein integriertes SE, sondern sieht nur den Zugriff via SWP auf ein SE auf der SIM Karte vor. Im Galaxy Nexus ist der PN65N NFC-Controller von NXP verbaut. Dieser Controller ist von der Funktionalität her identisch mit jenem im Nexus S.

Es ist also nur eine Frage der Zeit, bis der Zugriff auf das SE auch über das Android-API möglich ist. Für diesen Zugriff ist jedoch ein Schlüssel nötig und diesen kennen aktuell nur Google und NXP. Die Frage, wie dieser Schlüssel an die Entwickler verteilt wird, ist hingegen unbeantwortet.

Die Software

Konzentrieren wir uns daher auf das, was mit den heutigen Geräten möglich ist: das Lesen und Schreiben von NFC-Tags, sowie der Austausch von Meldungen zwischen Geräten. Die NFC Data Exchange Format (NDEF) Spezifikation [NDEF] legt die Formate für den Austausch von Informationen zwischen NFC-Geräten und NFC-Tags fest. Anwendungsdaten sind (zusammen mit Metainformationen) in einem oder mehreren NDEF-Records abgelegt, welche zusammen eine NDEF-Meldung bilden. Die Records ermöglichen die Repräsentation von Datenpaketen in verschiedenen Datenformaten. Ein Tag kann in einer NDEF-Meldung z.B. eine URI, ein Text und ein Icon enthalten. Wie Daten in den Records repräsentiert und auf den Geräten interpretiert werden, legt die *NFC Record Type Definition* (RTD) Spezifikation fest.

In Android sind die nötigen Datentypen im Paket *andorid.nfc* definiert. Die Klasse *NdefMessage* repräsentiert eine NDEF-Meldung, welche einen oder mehrere NDEF-Records enthält. In diesen Records sind die Nutzdaten abgelegt. Ein NDEF-Record besteht aus einem Header und einem Datenteil. Im Header sind neben Flags Informationen zum Typ der Nutzdaten, die Länge der Nutzdaten sowie optional eine eindeutige Kennung des Records abgelegt. Android bildet einen NDEF-Record in der Klasse *NdefRecord* ab.

Für die Beschreibung des Typs eines Records sind verschiedene Formate vorgesehen. Der Wert des Feldes *Type Name Format* (TNF) gibt an, in welchem Format der Typ codiert ist. Die TNF-Werte sind in der NDEF-Spezifikation (und als Konstante in der Klasse *NdefRecord* definiert. In Tabelle 1 ist die Bedeutung der verschiedenen TNF-Werte erklärt.

Neben den beiden Klassen *NdefMessage* und *NdefRecord* enthält das Paket *android.nfc* auch noch die Klassen *NfcManager* und *NfcAdapter*. Mit dem Manager kann auf den Adapter zugegrif-

TNF Wert		Bedeutung
TNF_EMPTY	0	Dieses Format zeigt an, dass der Record leer ist, d.h., er hat weder Typ, Kennung noch Nutzdaten. Die entsprechenden Getter-Methoden auf dem Record geben leere Arrays zurück.
TNF_WELL_KNOWN	1	Dieses Format zeig an, dass der Typ als NFC Forum Well-known Type codiert ist (entsprechend der Record Type Definition). Beispiel: „T“ steht für einen Text-Record, „U“ für eine URI oder „Sp“ für einen Smart-Poster-Record. Die wichtigsten Typen sind in den Konstanten NdefRecord.RTD_TEXT, RTD_URI und RTD_SMART_POSTER abgelegt.
TNF_MIME_MEDIA	2	Records mit diesem TNF enthalten einen Typ-String im MIME-Format nach RFC 2046, z.B. „image/png“ für ein Bild oder „text/x-vCard“ für eine Visitenkarte.
TNF_ABSOLUTE_URI	3	Wenn diese Konstante gesetzt ist, dann enthält das Typfeld eine absolute URI nach RFC 3986. Dieser URI definiert dann das Format der Nutzdaten.
TNF_EXTERNAL_TYPE	4	Dieses Format gibt an, dass das Typfeld ein NFC Forum External Type ist. Firmen können damit anwendungsspezifische Record-Typen definieren.
TNF_UNKNOWN	5	Wenn dieses Flag gesetzt ist, dann enthält der Record Daten in einem unbekannten Format. Der Typ-String ist in diesem Fall leer.
TNF_UNCHANGED	6	Ein TNF mit Wert 6 gibt an, dass die Nutzdaten auf mehrere Records verteilt sind (analog zu chunked encoding von HTTP-Payloads). Der Typ der Nutzdaten wird auf dem ersten Record spezifiziert, auf den nachfolgenden Records ist TNF=6 gesetzt und das Typfeld ist leer.

Tabelle 1: Bedeutung der TNF-Werte gemäss NDEF-Spezifikation

```
   <intent-filter>
      <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
      <category android:name="android.intent.category.DEFAULT"/>
      <data android:mimeType="application/vnd.fhnw.coupon" />
   </intent-filter>
```

Listing 1: Intent-Filter für das Lesen von NDEF-Meldungen

```
byte[] getCodeFromNdefMessages(Intent intent) {
   byte[] payload = null;
   if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())) {
      Parcelable[] rawMsgs = intent.getParcelableArrayExtra(
                                 NfcAdapter.EXTRA_NDEF_MESSAGES );
      if (rawMsgs != null && rawMsgs.length > 0) {
         NdefMessage msg = (NdefMessage)rawMsgs[0];
         NdefRecord[] recs = msg.getRecords();
         if(recs.length > 0) payload = recs[0].getPayload();
      }
   }
   return payload;
}
```

Listing 2: Auslesen der Daten aus einem Intent

fen werden, und der Adapter enthält Methoden für den Peer-to-peer-Modus sowie eine Methode *isEnabled* mit welcher geprüft werden kann, ob NFC in den Einstellungen für Drahtlosnetzwerke aktiviert ist.

Lesen von NFC-Tags

Sobald Android ein NFC-Tag erkennt, löst es einen entsprechenden Intent aus, der die eingelesenen Daten enthält. Falls es sich um einen Tag mit einer NDEF-Meldung handelt, dann wird ein Intent mit der Aktion *NDEF_DISCOVERED* verschickt. Falls der Tag keine Meldung im NDEF-Format enthält (oder falls der entsprechende Intent von keiner Activity behandelt worden ist), so wird ein Intent mit der Aktion *TECH_DISCOVERED* verschickt. Über das im Intent enthaltene Tag-Objekt kann geprüft werden, welche Protokolle dieser Tag unterstützt. Alle Android NFC-Geräte müssend die folgenden Technologien unterstützen:

- NfcA (ISO/IEC 14443-3 Typ A)
- NfcB (ISO/IEC 14443-3 Typ B)
- NfcF (JIS 6319-4, auch bekannt unter dem Namen FeliCa)
- NfcV (ISO 15693, auch bekannt unter dem Namen Vincinity)
- IsoDep (ISO 14443-4, APDU basierte Smart-Cards)
- Ndef (Tags, welche einem der vom NFC Forum definierten Tag-Typen entsprechen)

Das Nexus-S unterstützt daneben noch spezielle Protokolle für Mifare Karten. Falls der Tag keines dieser Protokolle unterstützt (oder falls der entsprechende Intent von keiner Activity behandelt worden ist), so wird ein Intent mit der Aktion *TAG_DISCOVERED* verschickt. Falls mehrere Aktivitäten auf einen Intent reagieren können, so muss der Benutzer wählen, welche Aktivität er verwenden will.

Falls eine Activity auf einen dieser Intents reagieren will, so muss ein passender Intent-Filter definiert werden. In Listing 1 ist angegeben, welchen Filter eine Activity deklarieren muss, um NFC-Tags lesen zu können, welche eine NDEF-Meldung mit einem speziellen MIME-Typ enthalten (hier *application/vnd.fhnw.coupon*).

Auf dem Intent, den die Activity erhält, können Zusatzinformationen ausgelesen werden:

- EXTRA_ID: Identifikationsnummer des Tags
- EXTRA_TAG: ein Objekt, welches den Tag repräsentiert (und zusätzliche Operationen anbietet)
- EXTRA_NDEF_MESSAGES: NDEF-Meldungen (falls es sich um einen NDEF-Tag handelt)

In Listing 2 wird der Inhalt (Payload) des ersten NDEF-Records der ersten NDEF-Meldung eines Intents ausgelesen und im Binärformat zurückgegeben.

```
   <intent-filter>
      <action android:name="android.nfc.action.TECH_DISCOVERED" />
   </intent-filter>

   <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
      android:resource="@xml/filter_nfc"
   />
```

Listing 3: Intent-Filter für die Kommunikation mit NDEF-Tags

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
   <tech-list>
      <tech>android.nfc.tech.Ndef</tech>
   </tech-list>

   <tech-list>
      <tech>android.nfc.tech.NdefFormatable</tech>
   </tech-list>
</resources>
```

Listing 4: Definition einer Tag-Technologie Liste

Schreiben von NFC-Tags

Um einen Tag mit dem Handy zu beschreiben, muss eine Verbindung zu diesem aufgebaut werden. Typischerweise wird dazu ein Intent-Filter vom Typ *TECH_DISCOVERED* definiert (Listing 3). Die Technologien, die vom Programm verarbeitet werden können, müssen in einer separaten Datei spezifiziert werden, auf die mit einem *meta-data* Tag verwiesen wird. Die Liste in Listing 4 spezifiziert, dass sie Tags vom Typ *Ndef* oder *NdefFormatable* beschreiben kann.

Listing 5 zeigt, wie ein Tag, der mit dem Intent-Filter in Listing 3 und 4 erkannt wurde, beschrieben wird. Dazu wird die Tag-Instanz aus dem Intent in den Typ *Ndef* oder *NdefFormatable* konvertiert, damit mit der Methode *writeNdefMessage* bzw. *format* eine NDEF-Meldung auf den Tag geschrieben werden kann. Die Methode *createRtdUriRecord* haben wir bereitgestellt. Diese Hilfsfunktion erzeugt einen NDEF-Record mit TNF=1 (WELL-KNOWN) und Typ=„U" (URI) und codiert den Prefix der URL gemäss Spezifikation.

In der Applikation weist man den Benutzer darauf hin, dass der Tag, der als nächstes berührt wird, beschrieben wird. Damit nun nicht andere Applikationen ebenfalls auf den Intent reagieren, der beim Berühren dieses Tags verschickt wird, kann man mit der Methode *enableForegroundDispatch* auf dem NFC-Adapter sicherstellen, dass nur die eigene Activity den Intent erhält.

Peer-To-Peer

Seit der Version 2.3.3 unterstützt Android auch die Peer-to-peer-Kommunikation zwischen zwei Geräten. Damit kann eine NDEF-Meldungen von Gerät zu Gerät übertragen werden. Ab der Version *Ice Cream Sandwich* von Android ist auch das Live-Pushing zwischen zwei Geräten möglich (Android Beam). Wenn man z.B. auf einem Gerät mit YouTube einen Film betrachtet und dann ein anderes NFC-Gerät berührt, so läuft der Film auf dem anderen Gerät an derselben Stelle weiter. Was jedoch auch mit *Ice Cream Sandwich* noch nicht unterstützt wird, ist die direkte Kommunikation über LLCP (NFC Logical Link Control Protocol).

Mit der Methode *enableForegroundNdefPush* kann eine Activity eine NDEF-Meldung via Peer-to-peer publizieren und mit *disableForegroundNdefPush* die Publikation wieder deaktivieren. Listing 6 zeigt, wie in den Methoden *onResume* und *onPause* die Meldung *pushMessage* publiziert wird. In einem zweiten Handy wird diese NDEF-Meldung über normale Intents publiziert.

```
void writeUrlToTag(Intent intent, String url) throws IOException,
      FormatException {
   String action = intent.getAction();
   if (NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)) {
      NdefRecord rec = NdefRecordRtdUri.createRtdUriRecord(url);
      NdefMessage msg = new NdefMessage(new NdefRecord[] { rec });

      Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
      Ndef ndef = Ndef.get(tag);
      if (ndef != null) {
         ndef.connect();
         ndef.writeNdefMessage(msg);
         ndef.close();
      } else {
         NdefFormatable ndeff = NdefFormatable.get(tag);
         if (ndeff != null) {
            ndeff.connect();
            ndeff.format(msg);
            ndeff.close();
         }
      }
   }
}
```

Listing 5: Methode welche eine URL auf einen Tag schreibt

```
public void onResume() {
   super.onResume();
   if (nfcAdapter != null)
      nfcAdapter.enableForegroundNdefPush(this, pushMessage);
}
public void onPause() {
   super.onPause();
   if (nfcAdapter != null)
      nfcAdapter.disableForegroundNdefPush(this);
}
```

Listing 6: Aktivierung und Deaktivierung von NDEF-Push

Anwendungen

Die auf dem Nexus S vorinstallierte *Tags*-Applikation unterstützt die Verwaltung von Tags. Wird z.B. ein mit dem Programm in Listing 5 beschriebener Tag berührt und vom Gerät erkannt (vorausgesetzt, NFC ist in den Einstellungen für Drahtlosnetzwerke aktiviert worden), so wird der darauf gespeicherte Link angezeigt (siehe Abb. 1) und kann mit einem Fingerklick weiterverarbeitet werden. Auf der Seite *Mein Tag* kann eingestellt werden, welcher Inhalt mit NDEF-Push publiziert werden soll (siehe Abb. 2).

Im Android Markt findet man weitere Anwendungen, welche NFC verwenden. Mit der Applikation *NFC Tag Info* können detaillierte Informationen zu Tags abgefragt werden. Abbildung 3 zeigt, wie diese Applikation die mit Listing 5 geschriebene NDEF-Meldung darstellt. Eine Applikation, welche das Beschreiben von Tags unterstützt, ist der *NXP Tag Writer*.

Mit der Applikation *WiFiTap* können WIFI-Konfigurationen auf einen Tag gespeichert und geladen werden (mit Netzname, Typ und Passwort). Die Applikation *NFC TaskLauncher* erlaubt es, Aktionen auf einem Tag abzuspeichern, die dann bei Berührung des Tags ausgeführt werden. Als Aktionen können z.B. Konfigurationseinstellungen (Lautstärken und Klingeltöne) vorgenommen werden, Alarme gestellt werden, Applikationen gestartet werden, etc. Ein solcher Tag könnte z.B. im Auto, am Arbeitsplatz oder im Sitzungszimmer deponiert werden, um entsprechende Einstellungen vornehmen zu können.

Taglet ist eine Applikation die es erlaubt, im Internet Informationen zu einem Tag zu hinterlegen. Diese Information wird dann angezeigt, wenn der Tag mit einem anderen Gerät ausgelesen wird. Auf diese Weise lassen sich Tags annotieren. Die Applikation *Enable Table* erlaubt es, Besuchern eines Restaurants Rabatt-Gutscheine abzugeben. Der NFC-Tag ist dabei im Rechnungsetui eingebaut. Aus der Beschreibung im Android Markt und im Internet geht jedoch nicht eindeutig hervor, wie diese Gutscheine dann eingelöst werden können.

Damit der Android Markt eine Applikation nur dann anzeigt, wenn das Gerät NFC unterstützt, muss dem Manifest das folgende uses-feature hinzugefügt werden: *<uses-feature android:name=„android.hardware.nfc" android: required=„true">*.

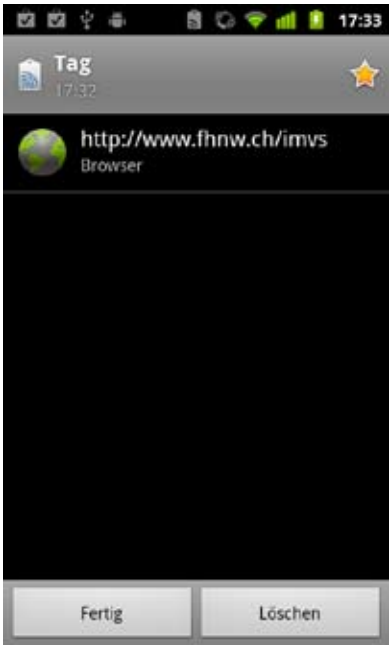


Abbildung 1: Nexus S Tags Applikation

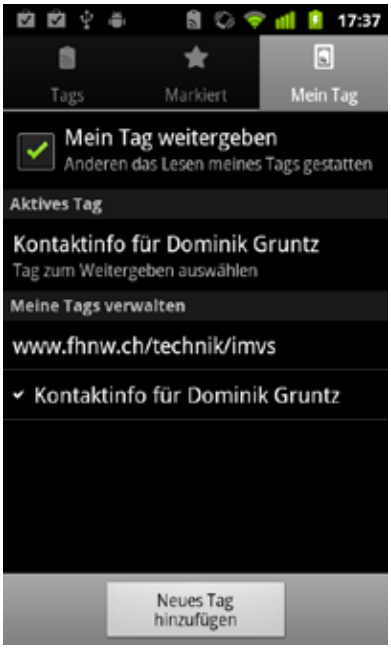


Abbildung 2: Weitergeben von Tags

Abbildung 3: NFC Tag Info App

Zusammenfassung

Die NFC-APIs und das Nexus S sind ein erster Schritt in Richtung Mobile Payment und Mobile Ticketing. Mit den vorliegenden Android-Versionen lässt sich diese Vision jedoch noch nicht realisieren, da dazu wichtige Funktionen noch fehlen. Der im Nexus S und Galaxy Nexus eingebaute NFC-Controller unterstützt jedoch die volle Funktionalität, daher ist es nur eine Frage der Zeit, bis Google auch die Bibliotheken auffrischt. Unklar ist auch, wie auf das SE zugegriffen werden kann, d.h., wie man die dazu nötigen kryptographischen Schlüssel erhält. Google Wallet zeigt, dass dies grundsätzlich möglich ist.

Die im Paket *android.nfc* definierten Klassen sind sehr spartanisch gehalten. Es wäre wünschenswert, wenn für die in den NDEF- und RTD-Spezifikationen definierten Typen eigene Klassen definiert wären. Es ist nicht einsehbar, warum jeder Programmierer die Tags erneut parsen soll. Aber auch in diesem Punkt darf damit gerechnet werden, dass hier in einer zukünftigen Version von Android nachgebessert wird.

Eines steht jedoch fest: NFC ist eine wichtige Technologie, deren Verbreitung und Akzeptanz durch die Initiative von Google nun rasch zunehmen wird. Ob 2011 als NFC Jahr in die Geschichte eingehen wird? Man hat in der Vergangenheit schon oft ein NFC Jahr ausgerufen, ohne dass Taten gefolgt sind. Google hat inzwischen erreicht, dass NFC in aller Munde ist, aber leider noch nicht, dass es auch in allen Mobiltelefonen verfügbar ist. Wenn Apple dereinst doch noch ein iPhone 5 mit NFC angekünden wird, dann ist NFC nicht mehr aufzuhalten.

Referenzen

[BG10] Ingo Bauersachs, Dominik Gruntz; Touch'n pay: ein NFC-Feldversuch, IMVS Fokus-Report, p. 37-42, 2010.

[Coop08] Coop Supermarkt, passabene für das praktische Einkaufen, 2008, http://tinyurl.com/passabene.

[LR10] Josef Langer, Michael Roland, Anwendungen und Technik von Near Field Communication (NFC), Springer Verlag, Sept 2010.

[MF10] Friedemann Mattern, Christian Flörkemeier, Vom Internet der Computer zum Internet der Dinge, Informatik Spektrum, Vol. 33, No. 2, S. 107-121, 2010.

[Migr11] Subito – einfach und schnell einkaufen, 2011, http://www.migros.ch/subito.

[NDEF] NFC Forum, NFC Data Exchange Format (NDEF), Rev. 1.0. Technical Specification, Jul. 2006, http://www.nfc-forum.org/specs/spec_list/

[PN544] NXP NFC Controller PN544 http://www.nxp.com/acrobat_download2/literature/9397/75016890.pdf

Android API Levels

Mit jeder neuen Android Version werden auch immer neue Features unterstützt. Mit Android 2.0 (Eclair) wurde Multi-Touch eingeführt, seit Android 2.3 (Gingerbread) ist neu die Kommunikation über NFC (Near Field Communication) möglich und Android 3.0 (Honeycomb) kennt neben Activities auch Fragments, um die UI-Möglichkeiten von Tablets besser unterstützen zu können. Der Programmierer muss bei jeder Applikation entscheiden, auf welcher Version er seine Programme entwickelt, und dabei Vorwärts- und Rückwärtskompatibilität beachten. Wir diskutieren dies in diesem Artikel am Beispiel einer Applikation, welche (optional) für den Austausch von Daten auch NFC verwenden soll.

Dominik Gruntz | dominik.gruntz@fhnw.ch

Android Entwickler werden beim Erzeugen eines neuen Projektes in Eclipse gefragt, für welche Version (welches *Build Target*) die Applikation vorbereitet werden soll. Diese Entscheidung legt fest, welche Bibliotheken bereitgestellt werden. Wählt man hier die neueste Android Version, so können alle neuen Features verwendet werden, aber die Applikation läuft nicht mehr auf Geräten, auf welchen eine Vorgängerversion des Systems installiert ist. Wählt man hingegen eine ältere Version, so sind die neuen Features nicht verfügbar.

Tabelle 1 zeigt die Android Plattformen, die bis heute publiziert worden sind, zusammen mit ihrer Verteilung im Markt. Implementiert man auf der Basis von Android 2.1 (Eclair), dann erreicht man 97.2% aller Android Nutzer. Falls man jedoch Android 2.3 (Gingerbread) voraussetzt, dann erreicht man nur noch einen Drittel der Android Nutzer. Eine historische Entwicklung der Verteilung der verschiedenen Android-Versionen findet man unter [And11a].

Vorwärtskompatibilität

Da wir möglichst viele Kunden mit unserer Applikation erreichen wollen, haben wir unser Projekt auf API Level 7 (Android 2.1.x) entwickelt. Google garantiert, dass eine Applikation, die für ein bestimmtes API Level entwickelt worden ist, auch auf allen höheren API Levels ausgeführt werden kann (ohne diese neu zu kompilieren). Dies wird erreicht, in dem bei neuen Android Versionen keine Änderungen vorgenommen werden, welche zu Inkompatibilitäten führen können. Konkret heisst dies, dass nur folgende Änderungen möglich sind:

- Hinzufügen von neuen Paketen, neuen Klassen und neuen Methoden
- Entfernen von Exceptions aus *throws*-Deklarationen in Methoden und Konstruktoren
- Hinzufügen von Runtime-Exceptions in *throws*-Deklarationen
- Verstärkung des Resultattyps bei Methoden (z.B. bei der *clone*-Methode)
- Erweiterung der Sichtbarkeit von Klassen, Methoden, Konstruktoren oder Feldern
- Markierung von Klassen, Konstruktoren, Methoden oder Feldern als *deprecated*

In der Vergangenheit konnten diese Bedingungen grösstenteils eingehalten werden. Mit jeder neuen Version wird auch ein *Android API Differences Report* bereitgestellt der zeigt, welche Änderungen am API vorgenommen worden sind (z.B. in [And11b] für die Änderungen zwischen den Versionen Froyo und Gingerbread).

Vorwärtskompatibilität ist enorm wichtig, denn so wird sichergestellt, dass die installierten Applikationen auch nach einem System-Update noch funktionieren.

Platform Version	API Level	VERSION_CODE	Verteilung
Android 1.0	1	BASE	
Android 1.1	2	BASE_1_1	
Android 1.5	3	CUPCAKE	1.0%
Android 1.6	4	DONUT	1.8%
Android 2.0	5	ECLAIR	
Android 2.0.1	6	ECLAIR_0_1	
Android 2.1.x	7	ECLAIR_MR1	13.3%
Android 2.2.x	8	FROYO	51.2%
Android 2.3, 2.3.1, 2.3.2	9	GINGERBREAD	0.6%
Android 2.3.3, 2.3.4	10	GINGERBREAD_MR1	30.7%
Android 3.0.x	11	HONEYCOMB	0.2%
Android 3.1.x	12	HONEYCOMB_MR1	0.7%
Android 3.2	13	HONEYCOMB_MR2	0.5%

Tabelle 1: Plattform Versionen und deren Verteilung [And11a]

Reflection

Unsere Applikation verwaltet Coupons und Vergünstigungen, die via QR-Codes[1] zwischen Nutzern ausgetauscht werden können. Falls die Applikation auf einem Gerät ausgeführt wird, welches Near Field Communication (NFC) unterstützt, so soll auch der Austausch von Coupons via NFC

1 QR-Codes sind zweidimensionale Codes, QR steht für Quick Response.

```
if(getPackageManager().hasSystemFeature("android.hardware.nfc"))  {
  try {
    Class<?> c = Class.forName("android.nfc.NfcAdapter");
    Method getDefaultAdapter = c.getMethod("getDefaultAdapter",
                                  android.content.Context.class);
    nfcAdapter = getDefaultAdapter.invoke(null, this);

    Class<?> clsNdefMessage = Class.forName("android.nfc.NdefMessage");
    Class<?> clsNdefRecord = Class.forName("android.nfc.NdefRecord");

    Object codeRecord = clsNdefRecord.getConstructor(Short.TYPE,
            byte[].class, byte[].class, byte[].class).newInstance(
      (short)2,    // NdefRecord.TNF_MIME_MEDIA
      "application/vnd.fhnw.coupon".getBytes(Charset.forName("US-ASCII")),
      new byte[0],
      itemData
    );

    Object arr = Array.newInstance(clsNdefRecord, 1);
    Array.set(arr, 0, codeRecord);
    ndefPushMessage = clsNdefMessage.getConstructor(arr.getClass()).newInstance(arr);
  } catch(Exception e) {
    nfcAdapter = null;
  }
}
```

Listing 1: Zugriff auf NFC Funktionalität via Reflection

möglich sein. Die auf diesen Geräten vorhandenen NFC Klassen sind jedoch nicht Teil des API Level 7. Der Zugriff auf diese Klassen muss daher mit Reflection gelöst werden.

Das API Level des Gerätes, auf dem die Applikation ausgeführt wird, kann (seit API Level 4) über den Wert von *Build.VERSION.SDK_INT* ausgelesen werden. Mit dieser Information weiss man, welche zusätzlichen Klassen via Reflection verfügbar sind. Im Fall von NFC kann auch abgefragt werden, ob der entsprechende Service vorhanden ist (denn die NFC Funktionalität ist in Gingerbread optional, d.h. die Klassen sind im API vorhanden, aber es ist nicht zwingend auch ein NFC-Adapter verbaut). In unserem Code prüfen wir daher in der *onCreate* Methode mit dem Aufruf `getPackageManager().hasSystemFeature("android.hardware.nfc")`, ob NFC unterstützt wird.

Listing 1 zeigt den Zugriff auf die NFC-Funktionalität via Reflection. Das Feld *nfcAdapter* ist dabei in der Activity als Feld vom Typ *Object* deklariert und und wird mit dem Default-NFC-Adapter initialisiert. Zusätzlich erzeugen wir noch die NDEF-Meldung, die wir dann via NDEF-Push bereitstellen wollen und legen diese im Feld *ndefPushMessage* vom Typ *Object* ab. In den Methoden *onResume* und *onPause* (in Listing 1 nicht gezeigt) prüfen wir dann, ob dieses Feld definiert ist, bevor wir versuchen, NDEF-Push mit den Methoden *enableForegroundNdefPush* bzw. *disableForegroundNdefPush* zu aktivieren bzw. zu deaktivieren.

Im Manifest haben wir die für NFC nötigen Erweiterungen vorgenommen. Android ignoriert

```
<uses-permission android:name="android.permission.NFC"/>

<application android:icon="@drawable/icon"
             android:label="@string/app_name">
  <activity android:name=".ItemListActivity"
            android:label="@string/app_name"
            android:launchMode="singleTask">

    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
      <action android:name="android.nfc.action.NDEF_DISCOVERED" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:mimeType="application/vnd.fhnw.coupon" />
    </intent-filter>
  </activity>
</application>
```

Listing 2: Manifest mit NFC Deklarationen

Permissions und Intent-Filter, die nicht bekannt sind. Das Manifest in Listing 2 funktioniert auf jeden Fall auch unter Android 2.1 (Eclair MR1). Im LogCat erscheint lediglich die Warnung

```
WARN/PackageManager(59):
 Unknown permission android.permission.NFC
  in package ch.fhnw.imvs.android.coupons
```

Rückwärtskompatibilität

Programmierung mit Reflection macht nicht wirklich Spass, daher empfiehlt Google, jeweils gegen das neueste API zu programmieren, dabei jedoch auch ältere APIs zu unterstützen. Dies kann erreicht werden, indem im Manifest mit der *uses-sdk*-Angabe sowohl ein minimales als auch ein Ziel-API spezifiziert wird:

```
<uses-sdk
  android:minSdkVersion="7"
  android:targetSdkVersion="10" />
```

Damit stehen die Bibliotheken des Ziel-APIs (hier Version 10) zur Verfügung, aber die Applikation kann auch bereits ab der Min-SDK-Version (hier Version 7) ausgeführt werden. Auf einem Gerät mit einem älteren Android System kann die Applikation jedoch nicht mehr installiert werden.

Klassen und Methoden, die in einer Version > 7 eingeführt worden sind, dürfen nur verwendet werden, wenn die Applikation auf einem Gerät mit entsprechender Android-Version ausgeführt wird. Wird diese Bedingung verletzt, dann wirft die Dalivk-VM einen *NoClassDefFoundError* und der Benutzer sieht eine Maske wie in Abbildung 1 und ist gezwungen, den Task zu schliessen.

Da jedoch mit den Klassen aus dem Ziel-API gearbeitet werden kann, vereinfacht sich der Code gegenüber jenem aus Listing 1 gewaltig. Im Gegensatz zur Version mit Reflection kann der Compiler nun prüfen, ob die Aufrufe semantisch korrekt sind. Das Resultat findet man in Listing 3. Dieser Code funktioniert auf allen Plattformen, da nur dann auf die NFC Funktionalität zugegriffen wird, falls diese vorhanden ist, und da die Dalvik-VM Klassen erst dann lädt, wenn sie auch benötigt werden. Eine Klasse wird geladen, gelinkt und initialisiert, unmittelbar bevor

- eine Instanz dieser Klasse erzeugt wird,
- eine statische Methode dieser Klasse aufgerufen wird oder
- auf ein statisches Feld dieser Klasse zugegriffen wird (ausser der Zugriff auf Konstanten, d.h. statische Felder vom Typ String oder mit einem primitiven Typ welche final deklariert sind).

Insbesondere dürfen Felder vom Typ *NfcAdapter* oder *NdefMessage* in der Activity deklariert werden, auch wenn diese Klassen erst mit API Level 9 eingeführt worden sind. Die Initialisierung dieser Felder mit *null* oder der Vergleich mit *null* impliziert kein Laden der entsprechenden Klassen und kann damit auch auf einem Gerät mit API Level < 9 ausgeführt werden. Auch die Konstante *PackageManager.FEATURE_NFC* (auch erst seit API Level 9) darf im Code verwendet werden, denn diese Konstante (mit dem Wert "android.hardware.nfc") wird direkt vom Compiler zur Übersetzungszeit aufgelöst.

Der Nachteil dieses Ansatzes ist, dass der Entwickler auch unbeabsichtigt Features aufrufen kann, welche auf der *minSdkVersion* gar noch nicht vorhanden waren. Eclipse zeigt zwar (durch Einblenden der JavaDoc) an, ab welchem API Level eine Methode gültig ist, aber es liegt in der Verantwortung des Programmierers, darauf zu achten. Insbesondere muss die Applikation auf allen unterstützen API-Levels getestet werden. In den Run- bzw. Debug-Konfigurationen von Eclipse kann angegeben werden, welches virtuelle Device beim Start verwendet werden soll. Hier ist die Option «manuell» zu wählen, damit man dann beim Start der Applikation angeben kann, auf welchem API Level die Applikation ausgeführt werden soll.

Um verschiedene Andorid Versionen zu unterstützen wird typischerweise mit dem State-Pattern gearbeitet, d.h. es werden unterschiedliche Klassen implementiert, welche die Funktionalität je nach Build-Version bereitstellen. In einer Factory wird dann beim Programmstart die richtige Strategie erzeugt. Ein Beispiel dazu findet man in [Bray10].

Library-Projekte: leider nein…

Man könnte dieses Problem entschärfen, indem man das Projekt gegenüber einem tieferen Ziel-API entwickelt, und die Zusatzfunktionalität in ein separates Projekt auslagert. Android bietet dazu spezielle Library-Projekte an. Library-Projekte enthalten Code und Ressourcen, die aus mehreren Projekten referenziert werden können. Diese Projekte werden nicht in ein APK-File übersetzt und können entsprechend nicht auf einem Gerät installiert werden, sondern die Klassen dieser Projekte werden in das referenzierende Projekt übernommen. Leider werden sie dabei auch unter dem API Level dieses Projektes übersetzt. Es ist also nicht möglich, ein Library-Projekt mit API Level 10 in ein Projekt mit API Level 7 zu importieren, falls das Library-Projekt Features von

Abbildung 1: NoClassDefFoundError

```
private NfcAdapter nfcAdapter;
private NdefMessage ndefPushMessage;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // ...
  if(getPackageManager().hasSystemFeature(PackageManager.FEATURE_NFC)) {
    try {
      nfcAdapter = NfcAdapter.getDefaultAdapter(this);
      assert nfcAdapter != null; // guaranteed by spec

      NdefRecord codeRecord = new NdefRecord(
        NdefRecord.TNF_MIME_MEDIA,
        "application/vnd.fhnw.coupon".getBytes(Charset.forName("US-ASCII")),
        new byte[0],
        itemData
      );
      ndefPushMessage = new NdefMessage(new NdefRecord[]{codeRecord});
    } catch(Exception e) {
      e.printStackTrace();
      nfcAdapter = null;
    }
  }
}

@Override
protected void onResume() {
   super.onResume();
   if(nfcAdapter != null) {
     nfcAdapter.enableForegroundNdefPush(this, ndefPushMessage);
   }
}

@Override
protected void onPause() {
  super.onPause();
  if(nfcAdapter != null) {
    nfcAdapter.disableForegroundNdefPush(this);
  }
}
```

Listing 3: Rückwärtskompatibler Zugriff auf NFC

API Level 10 nutzt die im API Level 7 noch nicht vorhanden waren.

Zusammenfassung

Android unterstützt Vorwärtskompatibilität, in dem es bei neuen Android Versionen keine Änderungen vornimmt, welche zu Inkompatibilitäten führen. Um auch ältere Versionen unterstützen zu können, ist im Manifest neben dem Ziel-API-Level auch ein Minimum-API-Level anzugeben. Die Übersetzung erfolgt dann gegenüber dem Ziel-API und es liegt in der Verantwortung des Programmierers sicherzustellen, dass Features nur dann verwendet werden, wenn sie auf dem Gerät, auf dem die Applikation läuft, auch vorhanden sind. Es wäre nützlich, wenn die IDE den Entwickler dabei unterstützen würde, z.B. indem alle Methodenaufrufe und Feldzugriffe, welche erst nach dem Minimum-API-Level eingeführt worden sind, entsprechend markiert würden.

Referenzen

[And11a] http://developer.android.com/resources/dashboard/platform-versions.html (2. September 2011)

[And11b] Android API Differences Report, From Level 8 to Level 9: http://www.devdiv.com/android/docs/sdk/api_diff/9/changes.html

[Bray10] Tim Bray, How to have your (Cup)cake and eat it too: http://android-developers.blogspot.com/2010/07/how-to-have-your-cupcake-and-eat-it-too.html

Active Data Logger

In the project[1] „Active Data Logger“ we investigate the design of embedded data acquisition systems when focusing on mobility, efficiency and coverage of as many use cases as possible. Mobile data acquisition systems are needed in many fields of research and productive environments that require mobility, and they have different requirements than non-mobile data acquisition systems. After all, data collection and analysis is a very important task, as it provides a basis for gaining knowledge and taking decisions based on the results. The feasibility of a universal mobile data acquisition system is demonstrated with a proof-of-concept design and implementation which covers multiple use cases.

Matthias Krebs, Christoph Stamm | matthias.krebs@fhnw.ch

Mobile data acquisition is an integral part of today's research, development and productive processes that take place in a mobile environment. Collecting data from different sources is important in order to perform tasks such as verifying a hypothesis or collecting statistical data.

Imagine a cycle racing team as an example. For the team staff, it is important to know technical and medical data of the cyclists, such as position, speed, cadence and heartbeat rate in real time. The problem is that during a race the team's cyclists can be spread over a large area, therefore, the team's supporting car cannot always be close to the cyclists. *Mobility* is one key factor in this case. In order to get data from the cyclists, the team needs a data acquisition system which is small, light and also *energy-efficient*, because the bicycles should carry as little extra weight as possible. The data acquisition system also has to be capable of communicating with the supporting car or the headquarters in real time, because the staff needs to know changes of data immediately. Because the supporting car can be several kilometers away from the cyclist, a wireless connection is required. Remote access to the data acquisition device might also be considered in case it has to be reconfigured or its status has to be checked.

In order to collect the required data, a data acquisition system that fits the use case is needed. Choosing the best product depends on the requirements of the use case, the data to be collected and, if predetermined, the sensors that are used for measurements. Many commercial data acquisition systems available on the market are tailored to a specific use case and do not perform well when trying to use them for a different purpose. These products have a well-defined set of features and supported input sources, which allows them to perform very well within the boundaries of their designated use case. However, this limited feature set can be a disadvantage if the product should be used in a different context.

In contrast to data acquisition systems intended for specific use cases, universal data acquisition systems are on the market as well. While the former often provide specific inputs for analog or digital sensors, depending on the intended purpose, the latter generally feature a number of simple analog or digital inputs. The reason for this is that analog sensors just need an analog voltage to be measured and digital inputs can be used for binary measurements such as limit switches or photoelectric barriers. The result is a universal data acquisition system that can be used with virtually any analog sensor and any binary output. However, the situation becomes different as soon as advanced digital sensors are taken into consideration. Digital sensors that feature digital communication interfaces such as SPI or I^2C are a lot more complex to access than analog sensors or digital inputs. They do not only require specific interfaces, but also complex communication protocols. This makes using digital sensors much more difficult to use in universal data acquisition systems.

Choosing a suitable communication technology is also important for the versatility, as the data acquisition system requires an internet connection to communicate with distant remote stations. Wired or wireless LAN could be integrated easily but requires local infrastructure. GSM-based networking is available in most places, but has limited bandwidth, operational costs for each device and makes remote access difficult due to private IPs generally being used. Such a data acquisition system cannot be accessed remotely unless a relay or push services like SMS are used.

This article describes parts of a project that has been conducted at the Institute of Mobile and Distributed Systems, in cooperation with the Institute of Micro-Electronics and the Institute of Aerosol and Sensor Technologies [GW10], which

1 Parts of this project have been financially supported by Scintilla AG and Förderverein Fachhochschule Nordwestschweiz Solothurn FVFS.

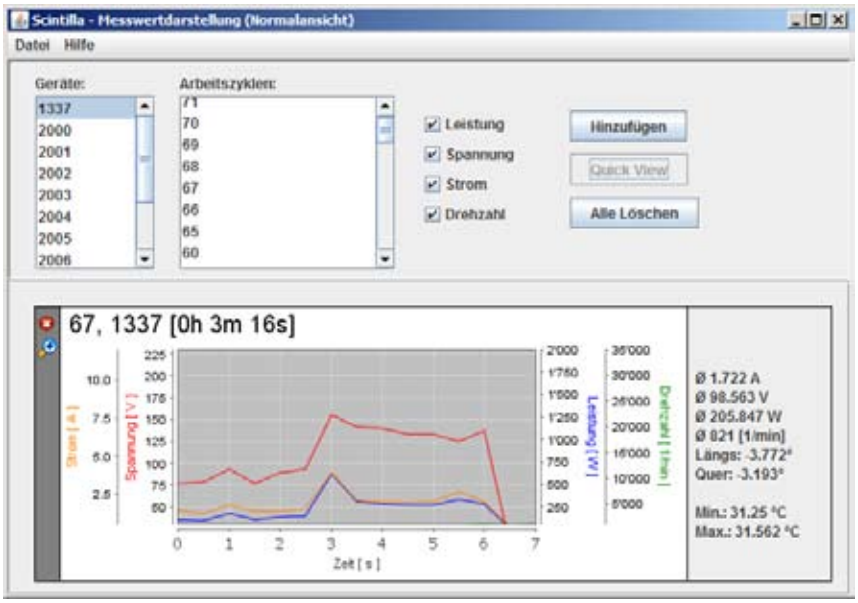


Figure 1: The modified data analysis application

both focus on embedded technology. Our goal is to create a basic framework for universal data acquisition systems that are capable of handling digital sensors and are usable in a mobile context. We take a look at a functional prototype implementing the design of a mobile universal data acquisition system that incorporates digital sensors and mobile communication technologies. The functional prototype includes software as well as embedded hardware components. We only take a brief look at the hardware components and focus on the software components instead.

Use Cases

In order to create a prototype, we need at least one but better several concrete use cases the design will be tested against. Two use cases are covered in this project, demonstrating the flexibility of the design. The first use case is the integration of sensors into portable power tools, which allows the manufacturer to analyze their usage profile. This part of the project is carried out in cooperation with Scintilla AG, a sub-organization of Bosch, and is a successor to the project [SC09]. In the project of 2009, the primary focus was to create the data analysis application *Scintilla Messwertdarstellung*, a Java-based desktop application that accesses a data collection database and displays data captured from power tools graphically. This application, as seen in Figure 1, is also used as part of our new functional prototype and is improved in the process.

The second use case is MiniDISC [MDIS], a device being developed at the Institute of Aerosol and Sensor Technologies, which is actually a mobile data acquisition device, because it is portable and measures fine dust particles. Both use cases share the same data transfer protocol [GW10].

System Design

The design of the data acquisition system is modular, so it can be adapted to different use cases more easily, with as little need to modify components as possible.

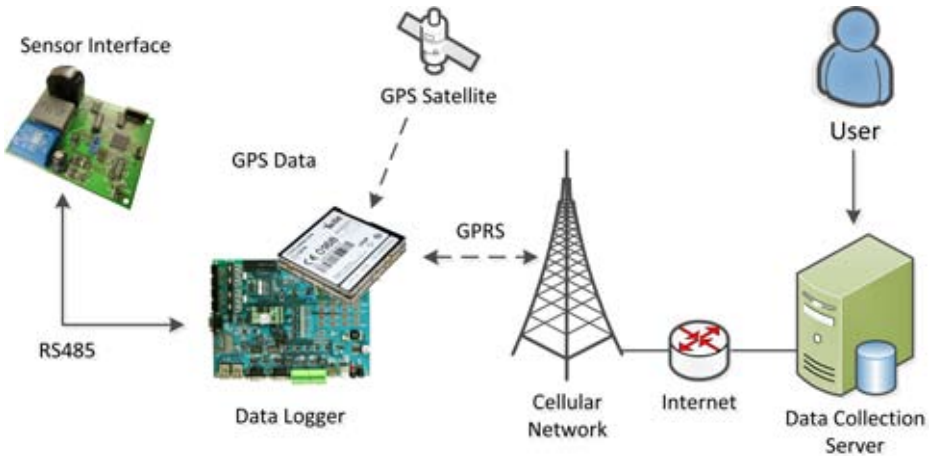


Figure 2: Data Acquisition System Design

The data acquisition system is divided into four main components:

1. a *sensor interface*, which contains sensors suitable for the designated use case, and a serial communication interface (RS485);
2. a *data logger device*, which is an embedded computer, has a local storage (e.g. SD card) and a communication module (GSM) attached, captures data from the sensor interface and transfers the data over the internet;
3. a *data collection server*, a software application that receives data from different data logger devices and stores the data inside a database;
4. a *data analysis application*, a software application that is used to access the database and evaluate the captured data. The design of this application is entirely dependent on the specific use case.

The components, particularly the sensor interface and data logger device, do not necessarily have to be separate. Depending on the actual use case, it could make sense to combine them in order to reduce the complexity or create a more compact design. We keep the sensor interface and the data logger separate due to our power tool use case. The reason is that the functional prototype we develop is simply too big to be attached to a portable power tool without any serious impact on usability.

Interfacing with Sensors

When designing a universal data acquisition system, support for many different digital sensors requires a standardized interface between the sensor interface and the data logger device. The sensors themselves might have different physical interfaces such as SPI or I^2C. This is why we use a small Atmel AVR microcontroller [AAVR] that controls the sensors and implements a serial communication protocol common to all sensor interfaces. The sensor interfaces are physically connected through an RS485 bus interface, which allows a single master (data logger device) and multiple slaves (sensor interfaces).

Using a microcontroller is an advantage, because only little hardware development is necessary, most of the functionality is provided through software. The AVR microcontroller is programmed in C, as the GCC toolchain and an Eclipse plugin are freely available. Configuration values such as data capture intervals are stored inside the AVR's EEPROM, this allows the sensor interface to be configured at runtime. The actual sensors are queried internally by the sensor interface, and the data is preprocessed and cached in memory. When sensor data is queried through the RS485 bus, the cached data is returned. This takes some load off the data logger, as the sensor data is already preprocessed on the sensor interface.

Data Logger Device

The data logger device is the core component of the data acquisition system, as it captures data from the sensor interfaces and either stores it locally or transmits it to a network server. Developing a hardware platform is a time-consuming task, this is why we have evaluated different existing platforms based on ARM processors, as these are widely used in embedded systems due to their flexibility and low power consumption. We have evaluated development boards from Roundsolutions (Aarlogic), Olimex (CS-E9302) and Quickembed (S3C2440SBC). However, they are pure developer board and could not easily be integrated in a custom design. The platform we have finally chosen is the ARM-based Eddy CPU module by SystemBase [SYSB], as it is compact, provides the necessary connectivity and runs a customizable embedded Linux operating system. The module itself can be detached from the developer board and mounted on a custom board. Applications can be programmed in C using the GCC toolchain, and after adding the C++ standard library (libstdc++) to the system, even C++ can be used for development. We choose C++, because it allows an object-oriented approach resulting in a more structured software design, while still being efficient enough to run on an embedded system.

The embedded data logger software is divided into separate threads, as we need concurrent data retrieval and network connectivity. One thread fetches data from the sensor interfaces in defined intervals. The interval duration depends on how often new data is required. To address the problem of a potential lack of a network connection, captured data is stored temporarily and transmitted as soon as a connection is available again. We implement this functionality by pushing the data, which is to be transmitted over the network, into a memory-based FIFO. If no connection is available, the data stays there until the connection is available. This is done in the same thread as the data capture. A second thread, which handles the network connection, takes data from the FIFO when a network connection is available. Of course, the FIFO is limited due to system memory limitations.

Future implementation could also introduce a persistent local storage such as an SD card. This would allow data to be saved until a network connection is available, even if the device is turned off.

GPRS Data Transfer

A mobile data acquisition system that transmits its acquired data automatically and independent of its location needs a mobile network connection. We choose a GPRS connection through the cellular network, because unlike wired or wire-

less LAN connections, it is available almost everywhere on Earth.

There are different embedded GSM modules on the market. They primarily differ in size and functionality. Our hardware of choice is a Telit GM862-GPS embedded module, since it provides all the GSM/GPRS functionality in a single package and can be controlled through a serial RS232 interface. It also includes a GPS receiver that allows recording the location of the device. Other products need a separate SIM slot or GPS receiver. To find out whether a GPRS connection provides enough bandwidth for the captured data being transmitted, tests using different packet sizes are conducted[2].

Packet size (bytes)	Average upload (bps)
32	1718
64	2850
128	4017
256	5362
512	6374
1024	9500

Table 1: GPRS upload performance in relation to packet size

These test results in Table 1 show that the data transfer is more efficient with increasing packet size. This is primarily due to the overhead produced as a smaller packet size requires more send commands to be executed to transmit the same amount of data. Additionally, the upload bandwidth is limited to 9600 bps when GPRS is used. Fortunately, this is enough for our power tool scenario, as we only capture data when the power tool is switched on.

Acquired sensor data is transmitted using a custom data transfer protocol that uses a TCP/IP connection. We intend to use a custom protocol, because we need as little communication overhead as possible in order to keep communication costs low. The protocol we use has been developed by Michael Glettig and Benjamin Wyrsch during a student project [GW10]. The protocol design is kept as simple as possible, so it can be implemented in embedded systems that have only little resources, and to minimize the amount of data being transmitted in order to keep communication costs low.

The basic concept of the communication protocol is the distinction of use cases. Each protocol configuration stands for one use case, which is identified by a *protocol ID*. Additionally, a *revision ID* allows different versions of the configuration on the same server. The reason different versions need to be distinguishable is that a relational database is used to store the data, and the database layout may change with protocol revisions, which could result in conflicts with existing data inside the database tables. Sensor data is divided into data *channels* and *sub-channels*, where each data channel consists of a group of one or more sub-channels. A sub-channel is a single data value of one of the common numeric data types, such as integers of different size and floating point numbers. We use this structure because there are data values that are related to each other, for example, a GPS position always contains a latitude and a longitude value. When data is transmitted, it is up to the data logger which channels are transmitted in a single packet, because the channels can have different data capture intervals, and the amount of data transmitted should be kept to a minimum. Nevertheless, when a channel is to be transmitted, all its sub-channels must be transmitted along with it because their data values are related. Each channel has a unique ID within a specific protocol configuration.

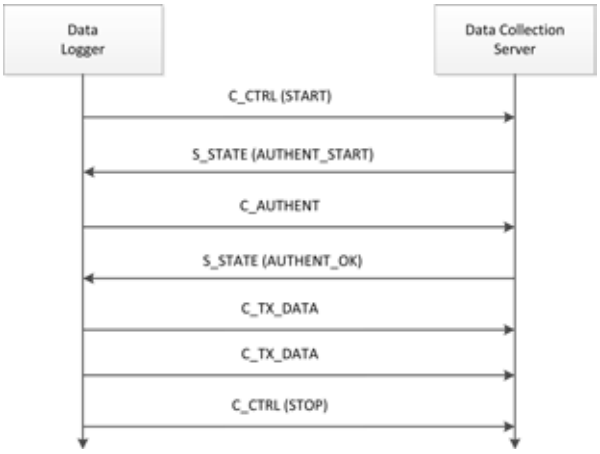


Figure 3: A typical conversation between a data logger and the data collection server

Data logger devices are identified through a unique device ID, which is a 64-bit integer. Additionally, each device uses a password to authenticate itself before transmitting data. This provides basic security against unsolicited data collection.

The communication protocol uses binary data packets of the following structure:

- a header (5 bytes) containing the packet ID (1 byte), the data header size (2 bytes) and the data payload size (2 bytes)
- an optional data header of variable size which describes the structure of the data payload
- a data payload of maximal 65'535 bytes

There are four types of communication packets:

- The *C_CTRL* packet is sent to the server to control the connection. It contains a single payload byte 00h (start a connection) or FFh (stop connection). The start packet is acknowledged by a status response. The stop packet is not

2 The GM862-GPS module and a Swisscom SIM card are used for all GPRS tests. Results may vary when other providers are used.

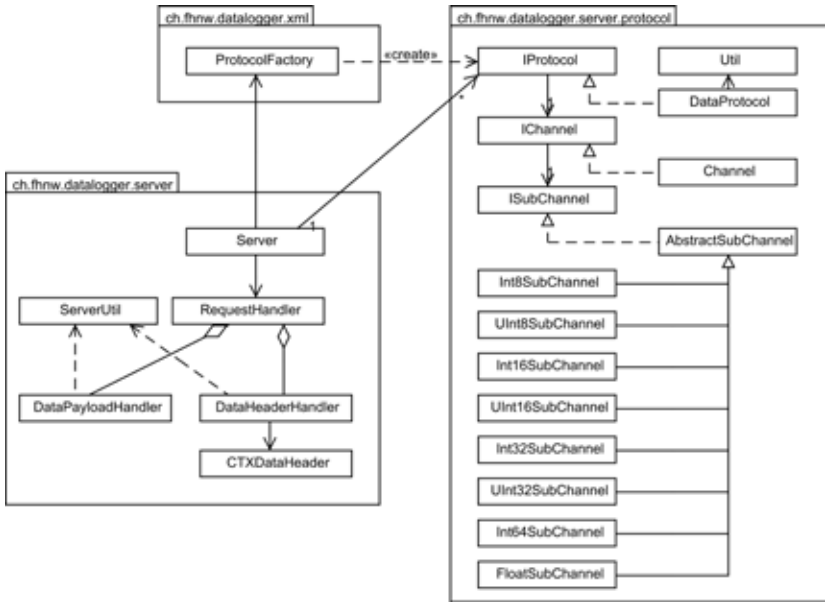


Figure 4: The software design of the data collection server

acknowledged as the TCP socket is closed immediately by the server.

- The *C_AUTHENT* packet is sent after initiating a connection and a server response that notifies the client that the server is ready for authentication. It is acknowledged by a status response telling the client that either the authentication was successful, or it failed either due to wrong credentials or an unsupported protocol ID. There is a 1-byte data header specifying the device password length. The payload contains the device ID, the password as well as the protocol ID and revision (1 byte each). The password supports ASCII format only, which circumvents problems with Unicode handling.
- The *C_TX_DATA* packet contains channel data and is sent by the client when the connection is established and the client has been authenticated. Data packets are streamed by the client without receiving an acknowledgement. The data header contains an array of the channel IDs whose data is present in the payload and the size of each channel payload. The channel order is arbitrary, but it must be the same as the order of the channels in the payload. The order of the sub-channel values is fixed.
- The *S_STATE* is a server status response that acknowledges *C_CTRL* and *C_AUTHENT*.

An example of a typical communication sequence is shown in Figure 3. The communication protocol is described in more detail in [GW10] and [MK11].

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<protocol>
  <title>Example data acquisition protocol</title>
  <protocol_properties>
    <id>0x12</id>
    <revision>0x00</revision>
    <description>Example Data Acquisition</description>
  </protocol_properties>
  <data_channels>
    <channel>
      <id>1</id>
      <name>Temperature</name>
      <preserve>true</preserve>
      <subchannel>
        <name>T1</name>
        <bytes>4</bytes>
        <format>float</format>
      </subchannel>
      <subchannel>
        <name>T2</name>
        <bytes>4</bytes>
        <format>float</format>
      </subchannel>
    </channel>
  </data_channels>
</protocol>
```

Listing 1: An example of a protocol configuration

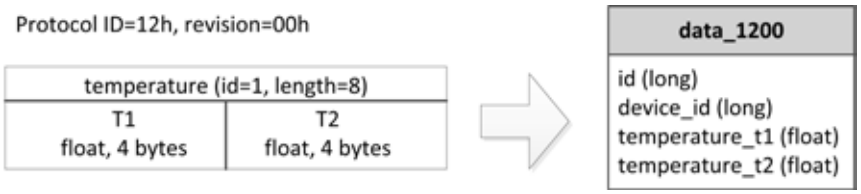


Figure 5: An example of the mapping of a data channel to the database table

Data Collection

Collection of captured data is performed on a centralized system. We implement this system in the form of a Java based stand-alone server software that uses a TCP/IP socket to listen for incoming data. Incoming data is processed and persistently stored in a PostgreSQL database.

The data collection server is designed as a truly use-case-independent tool, which means it can be configured to match any data acquisition scenario without the need of source code modification. The class diagram of the server software is shown in Figure 4. The main server class contains a list of protocol configurations that are loaded on start-up and the communication handler classes that process the client connections in a multi-threaded fashion. For each connection, a new thread is created. This method limits the scalability of the system, but this is no issue here, as the primary goal is to create a basic functional prototype that demonstrates the concepts.

Support for different use cases without code modification is provided through protocol configurations that are loaded on start-up. These protocol configurations are described in XML format, which makes them easy to edit by hand and parse. An example with a channel containing two sub-channels is provided in Listing 1.

For each protocol configuration, a separate file is placed in the *protocols* subdirectory of the application directory. All protocol configurations in this directory are loaded by the *ProtocolFactory* upon start-up.

The *ProtocolFactory* possesses the method *IProtocol loadFromXml(String filename)*, which parses a protocol configuration and creates a protocol object structure according to the class diagram in Figure 4. Each protocol object contains a list of channel objects, which each contain a list of sub-channel objects. Their class depends on the data type. Supported types are all signed and unsigned integer types between 8 and 64 bits, as well as 32-bit floating point numbers. Each protocol object also contains a channel map that allows to find channels by ID quickly, and it provides a method *setupSQL()*, which generates appropriate CREATE and INSERT statements according to the protocol object structure.

The database layout is quite simple. There is a global device table called *devices*, which bears the two fields *device_id* and *password*. Besides the device table, only one table is required per protocol configuration, this is the data table. Data channels are mapped to the database as shown in the example in Figure 5. The data table is named *data_<protocolID><protocolRevision>*, this provides a unique name for each protocol ID and revision. The table of the example in Listing 1 would be named *data_1200*. Common to each data table is a unique *id* field and a *device_id* field that references the device table. The remaining data fields depend on the specific protocol configuration. Their naming convention is *<channelname>_<subchannelname>*, and the order of the fields is equal to the order of channel and subchannel objects inside the protocol object. There is one caveat with certain programming languages and database systems, which also concerns Java and PostgreSQL: They do not support unsigned integer data types. This is why unsigned integers are represented by a signed integer of twice the width in the database, for example, an 8-bit unsigned integer is represented as a 16-bit signed integer. For this reason, there is no unsigned 64-bit integer, as a signed 64-bit integer is the largest integer type available to JDBC, unless database-specific big integers are used. Integers are interpreted this way so no additional conversion is necessary and the original number range is available.

When a data logger opens a connection to the data collection server, the appropriate protocol configuration is chosen through its ID and revision. As soon as a C_TX_DATA packet is received, the payload is processed in the method *void processData*() of the protocol object. The binary payload data is deserialized according to the channel IDs and their length of each channel in bytes. Deserialization of the sub-channel values is done by just converting from the binary little-endian representation of integer and floating point numbers. By default, Java uses big-endian, but this behavior can be changed by using a little-endian *ByteBuffer*. If there is a mismatch between the transmitted channel length and the length defined in the protocol configuration, the current channel is skipped and processing is synchronized to the beginning of the next channel.

As mentioned before, not all channels have to be transmitted in every C_TX_DATA packet. This requires a well-defined behavior to handle omitted channels. In this implementation, each channel has a *preserve* flag, which is specific to the protocol configuration on the server side. If the

flag is true, the sub-channel values that have been received last are kept in memory and inserted into the database, even if no data for this channel is received. If the flag is false, or if no data has ever been received since server start-up, NULL is inserted for all sub-channel values of the omitted channel. This shows explicitly that no data has been received from this channel. There is no other way than to insert NULL or a valid value, as we always have to insert a full row into the database.

Conclusion

The functional prototype described in this article demonstrates that the fundamental concepts of the design are a suitable basis for a working data acquisition system. Parts of the design, mainly the GPRS communication protocol and an implementation of the data collection server, are already actively used in a larger scale test of the MiniDISC device [MDIS]. The functional prototype we have developed is capable of capturing data from different sensors and propagating the captured data to a centralized data collection server without user interaction, therefore covering the entire process chain. Furthermore, it fulfills the requirement of a location-independent and mobile system.

The sensor interfacing and data collection concepts are already proven to be flexible and adaptable to different use cases. The configurability of the data logger device, however, still needs to be improved in order to allow an adaptation without changing the embedded software on the device. Also, the general stability, performance and reliability of the system are not optimal yet.

Future Research

The research conducted during the project *Active Data Logger* has given us insight into the topic of data acquisition and general embedded hardware and software development. There is still a lot of potential for improvements that could turn the universal data acquisition system into a product that could be usable in a productive environment.

The Institute of Mobile and Distributed Systems is planning to acquire future projects with industrial partners in order to continue research on this topic. These projects could be based directly on the current research, or focus on completely new goals, while still benefiting from the results of this research.

References

[AAVR] Atmel AVR: http://www.atmel.com/avr

[GW10] Feinstaub-Messnetzwerk. Michael Glettig, Benjamin Wyrsch, Fachbericht HS2010, Institut für Aerosol- und Sensortechnik, Fachhochschule Nordwestschweiz, 2011.

[MDIS] MiniDISC Feinstaubmessgerät: http://fierz.ch/minidisc/

[MK11] Active Data Logger. Matthias Krebs, Master Thesis, FS2011, Institut für Mobile und Verteilte Systeme, Fachhochschule Nordwestschweiz, 2011. http://webapache.imvs.technik.fhnw.ch/~christoph.stamm/reports/P9_2011_ActiveDataLogger.pdf

[SC09] IP209 – Projekt Scintilla. Michael Bodmer, Dominic Feer, Harun Gezici, Roland Kappeler, Adrian Roth, Michael Schneider, Thomas Weber, Institut für Mobile und Verteilte Systeme, Fachhochschule Nordwestschweiz, 2009.

[SYSB] Eddy CPU Module by SystemBase: http://www.embeddedmodule.com

Energieoptimiertes Data Center

Die Effizienz eines Data Centers in Bezug auf dessen Energieverbrauch wird mit steigenden Energiekosten ein immer wichtigeres Thema. Das KTI-Projekt „Energie optimiertes Data Center" befasst sich deshalb mit der mathematischen Modellierung des Energieverbrauchs eines Data Centers. Ziel des Projektes ist eine benutzerfreundliche Planungs- und Optimierungssoftware, welche mit Hilfe der entwickelten Modelle den Energieverbrauch eines Data Centers und dessen Teilsysteme berechnet. Unsere Software wird den Data Center Planern helfen, verschiedene Varianten von Data Centern zu planen und zu vergleichen, um dem Kunden die energieeffizienteste Lösung vorzuschlagen.

Cyrill Grüter, Peter Gysel, Christoph Meier | peter.gysel@fhnw.ch

Der Energieverbrauch wird zu einem immer wichtigeren Kostenfaktor bei der Planung und beim Betrieb von Data Centern. Dies führt dazu, dass auf dem Gebiet der Data Center Effizienz verschiedene internationale Organisationen tätig sind. Unter anderem setzt sich „The Green Grid" [GG1] mit dem Thema auseinander, wie die Effizienz von Data Centern gemessen werden kann. Die „American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE)" befasst sich mit der Effizienz der Kühlung [ASH]. Es ist zu erwarten, dass in naher Zukunft Energieeffizienz und entsprechende Normen, Gütesiegel usw. eine wichtige Rolle bei der Auswahl von Datencenter-Anbietern spielen werden. Unser KTI-Projekt[1] „Energie-optimiertes Data Center" befasst sich daher mit der mathematischen Modellierung des Energieverbrauchs eines Data Centers. Ziel des Projektes ist eine benutzerfreundliche Planungs- und Optimierungssoftware, welche mit Hilfe der entwickelten Modelle den Energieverbrauch eines Data Centers und dessen Teilsysteme berechnet.

Die am KTI-Projekt beteiligte Firma green.ch möchte als Betreiberin grosser Rechenzentren ihren Energieverbrauch nachhaltig senken. Die am Markt erhältlichen Optimierungsmassnahmen sollen dabei herstellerunabhängig auf ihre Energieeinsparpotentiale verglichen werden, so dass die beste Variante identifiziert werden kann. Obwohl diese Fragestellung eher technischer Natur ist, müssen auch ökonomische Aspekte wie Nachhaltigkeit und Investitionsschutz berücksichtigt werden. Die ebenfalls am Projekt beteiligte Firma R+B Engineering AG erstellt Planungen für Data Center Betreiber. Sie benötigt ein Werkzeug, um definierte Data Center Designs auf ihren Energieverbrauch zu untersuchen und bereits im Entwurfsstadium zu optimieren. Das Werkzeug soll es ermöglichen, dem Kunden rasch Optimierungsvarianten und deren Einsparungen aufzeigen zu können.

In unserem KTI-Projekt wird zur Berechnung des Energieverbrauchs eines Data Centers zunächst auf Basis der relevanten Normen und Vorgaben ein mathematisches Modell entwickelt. Anhand von Messungen in realen Data Centern wird dieses Modell anschliessend validiert und verfeinert.

In einem zweiten Schritt haben wir das Modell in eine Planungs- und Optimierungssoftware überführt, welche den Energieverbrauch eines geplanten Data Centers berechnet. Ein Planer kann somit verschiedene Varianten berechnen und vergleichen. Damit kann er dem Kunden die am besten geeignete Variante rasch vorschlagen.

Energiefluss in einem Data Center

Das Ziel der Data Center Betreiber ist der sichere Betrieb ihres darin untergebrachten Datacom Equipments. Der Betrieb von Datacom Equipment erfordert unter anderem einen bestimmten Energieeinsatz. Zum Energiebedarf des eigentlichen Datacom Equipments kommt der Energiebedarf von Hilfsbetrieben (Licht, Lüftung, Arbeitsplätze, etc.) hinzu. Diese Systeme haben immer eine gewisse Verlustleistung, welche so klein wie möglich sein sollte.

Für die Modellierung des Energieverbrauchs eines Data Centers muss dessen Energiefluss betrachtet werden. Dabei wird die Energie in Form von elektrischem Strom dem Data Center zugeführt und verlässt dieses in Form von Wärme wieder (siehe Abbildung 1).

Die Energie wird in Form von elektrischem Strom dem Data Center zugeführt. Je nach Ausbau und Anschlussart des Data Centers fliesst der Strom durch mehrere Stufen, bevor er zum Datacom Equipment gelangt. Am Eingang des Data Centers wird der Strom auf einen Transformator geführt. Dieser setzt die Spannung des Anschlusspunktes (üblicherweise Mittelspannung im Bereich von 10 kV bis 50 kV) auf die im Data Center benötigte Niederspannung um. Vom Transformator wird die Energie weiter zu einer

1 KTI Projekt 10941.1 PFES-ES

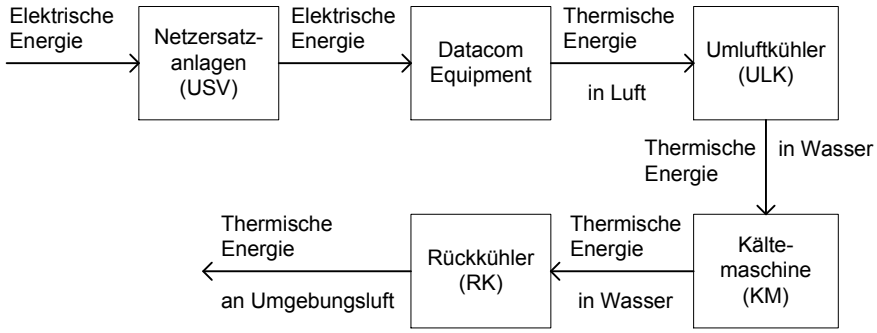


Abbildung 1: Energiefluss in einem Data Center

unterbrechungsfreien Stromversorgungsanlage (USV) geführt. Diese schützt die angeschlossenen Geräte vor verschiedenen Netzstörungen wie Versorgungsunterbrüchen, Überspannung und weiteren Störungen. Am Ausgang der USV wird das eigentliche Datacom Equipment angeschlossen. Dieses setzt die elektrische Energie in Wärmeenergie um. Die Wärme wird meistens in Form von Luft abgeführt[2].

Die Energie in Form von warmer Luft muss aus dem Data Center abgeführt werden um das Datacom Equipment stets genügend zu kühlen. Dazu wird die Luft durch ein wassergekühltes Register geführt. Hier findet ein Wärmeaustausch zwischen der Luft und dem Wasser statt. Die Luft wird gekühlt und das Kaltwasser dadurch erwärmt. Die Energie im Kaltwasser muss nun wiederum abgeführt werden. Um das Kaltwasser auf eine genügend tiefe Temperatur kühlen zu können, wird eine Kältemaschine eingesetzt, welche entweder luft- oder wassergekühlt sein kann. Bei der luftgekühlten Variante wird der Kondensator der Kältemaschine direkt mit der Umgebungsluft gekühlt. Die Wärmeenergie wird dadurch an die Umgebungsluft abgeführt.

Gesamtmodell

Für die Berechnung des Energiebedarfs betrachten wir den zuvor dargestellten Energiefluss im Data Center. Wie im vorhergehenden Abschnitt gezeigt setzt sich der Energiefluss eines Data Centers aus mehreren Komponenten und Teilsystemen zusammen. Unser Gesamtmodell basiert auf fünf unabhängig berechenbaren Teilsystemen. Dies sind die Netzersatzanlagen (USV), das Datacom Equipment (Server, Switches, Storage, etc.), die Umluftkühlgeräte (ULK), die Kältemaschinen (KM) sowie die Rückkühler (RK). Für jedes dieser Teilsysteme ist ein eigenständiges Modell entwickelt worden, die alle zu einem Gesamtsystem zusammengeführt worden sind.

Die Eingaben für das Modell werden in der oberen Zeile in Abbildung 2 dargestellt. Diese Angaben müssen einerseits durch den Betreiber eingegeben werden und andererseits kommen sie von Datenblättern der Hersteller der Kühlgeräte. Die untere Zeile umfasst die Ausgaben, die der Betreiber mit Hilfe der Berechnungen erhält. Die Verbindungen in der Mitte sind Zwischenresultate, die jedoch direkt wieder als Eingaben für die nächste Stufe verwendet werden. So ergeben sich auch die Abhängigkeiten zwischen den verschiedenen Teilsystemen.

Modellierung der Teilsysteme

In den folgenden Abschnitten werden die einzelnen Teilsysteme, auch Blöcke genannt, und deren Modellierungen kurz erläutert. Für den Block „Umluftkühler (ULK)" zeigen wir exemplarisch die mathematischen und physikalischen Details des Modells auf.

Datacom Equipment: Das Design eines Data Centers beginnt mit der Definition der IT-Leistung. Die IT-Geräte sind die eigentliche Quelle für den Energieverbrauch in einem Data Center. Alle weiteren Komponenten hängen von diesem Verbrauchswert ab. In einem bestehenden Data Center kann die Leistung der installierten IT-Systeme gemessen werden, der Verbrauch ist demnach bekannt. Wird ein neues Data Center geplant, stellt die für das Datacom Equipment zur Verfügung stehende elektrische Leistung eine Designgrösse dar und muss abgeschätzt werden. Wir modellieren das Datacom Equipment als einen Block mit dem Eingangsparameter elektrische Eingangsleistung (P_{IT}). Dieser kann entweder einen fixen Wert darstellen oder mit Hilfe unserer Modellierungen für den Energieverbrauch eines Servers, eines Netzwerk Switches und eines Storage Systems berechnet werden. Die Grundlage für die Modellierung bildet der Energiesatz: Die gesamte elektrische Energie, welche dem Datacom Equipment zugeführt wird, wird durch dieses in Wärme umgewandelt. Der Anteil des Datacom Equipments am Gesamtenergieverbrauch des Data Centers beträgt zwischen 50% und 80%.

USV: Die USV-Anlagen werden für den maximalen angeschlossenen elektrischen Bedarf geplant. Anhand der Anlagedaten und dem Betriebspunkt kann festgestellt werden, mit welchem Wirkungs-

2 Es sind auch wassergekühlte Systeme auf dem Markt, bei welchen die Chips die Wärme direkt an Kühlwasser abgeben.

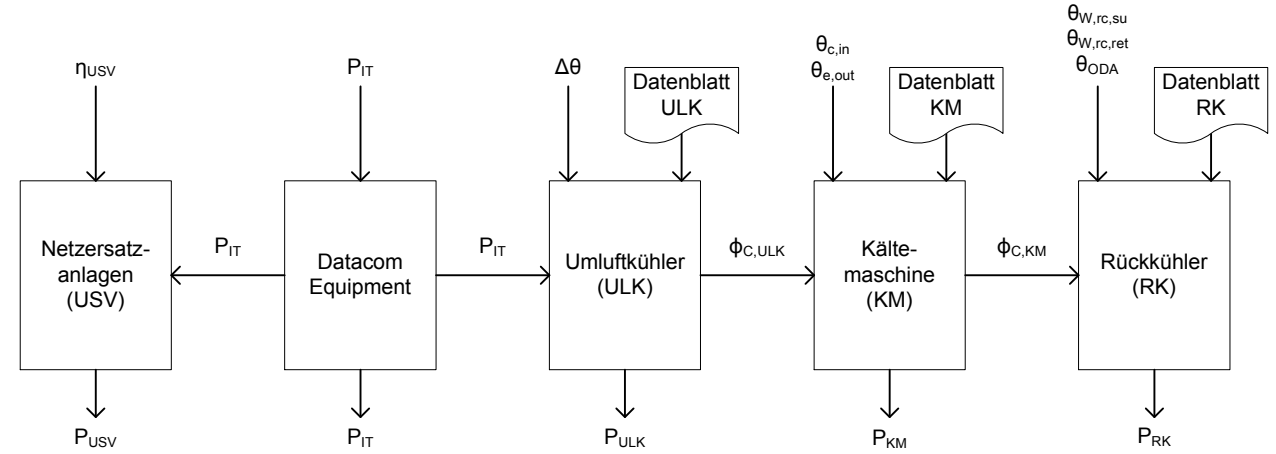


Abbildung 2: Schematische Darstellung des Gesamtmodells

grad die Anlage arbeitet. Das heisst, dass der Block USV als Eingabeparameter den Wirkungsgrad (η_{USV}) und vom Datacom Equipment Block die angeschlossene Leistung erhält. Die USV-Anlagen erreichen dabei nur einen Anteil von 2% bis 4% des gesamten Energieverbrauchs.

Umluftkühler (ULK): Der Zweck der Umluftkühler besteht darin die erzeugte Wärme des Datacom Equipments abzuführen. Aktuell werden nur luftgekühlte Systeme mit Umluftkühlgeräten betrachtet. Später sind an dieser Stelle auch andere Systeme wie frischluft- und wassergekühlte Systeme einsetzbar. Für die Berechnung des elektrischen Leistungsbedarfs der ULK benötigt unser Modell Angaben aus den Datenblättern (Ventilatorleistung ($P_{F,req}$) und Volumenstrom ($q_{V,req}$) bei Volllast) und Werte für zwei allgemeine Parameter der Luft (Wärmekapazität (c_p) von Luft und Luftdichte (ρ)). Je nach Bau- und Betriebsart des Data Centers wird ein bestimmter Temperaturhub zwischen der Zu- und Abluft des ULK erreicht. Diese Differenz (Δθ) erwartet das Modell als Designgrösse und ist vom Betreiber zu definieren. Die Wärmeleistung (P_{th}) wird durch die elektrischen Verbraucher im Data Center (Datacom Equipment) erzeugt und ist somit bekannt. Das Modell für die Kälteabgabe berechnet aus den angegebenen Parametern den benötigten Volumenstrom qV:

$$qV = P_{th}/(\rho \cdot c_p \cdot \Delta\theta).$$

Im Idealfall ist der Temperaturunterschied (Δθ) möglichst gross, so dass der Volumenstrom kleiner wird. Aus dem Volumenstrom lässt sich nun die dafür benötigte elektrische Leistung P_F im Teillastbetrieb mit Hilfe von bestehenden Normen des Schweizerischen Ingenieur- und Architektenvereins (SIA) näherungsweise berechnen [SIA]:

$$P_F = P_{F,req}/100 \cdot (19{,}67 \cdot (q_V/q_{V,req})^3 + 96{,}82 \cdot (q_V/q_{V,req})^2 - 25{,}98 \cdot (q_V/q_{V,req}) + 10{,}42).$$

Der Anteil der Umluftkühler am gesamten Energiebedarf des Data Centers liegt in der Praxis im Bereich von 4% bis 10%.
Die Abbildung 3 zeigt beispielhaft die aufgenommenen Messwerte eines ULKs und die nach unserem Modell berechnete elektrische Leistung. Der Arbeitsbereich eines ULKs liegt typisch bei einer Auslastung von 50% bis 100%. In diesem Bereich weicht die Modellierung zwischen 3,4% bis maximal 14,7% vom Messwert ab. Wobei die grösste Abweichung bei einem Volumenstrom von1,1 m^3/s auf eine Fehlmessung zurück geführt werden muss.

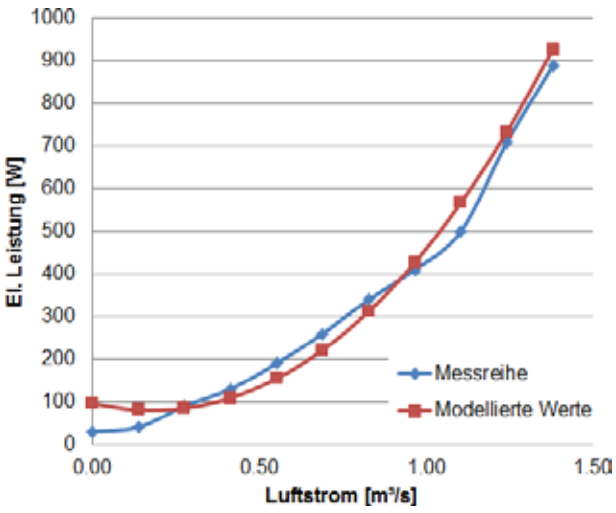


Abbildung 3: Berechnete Werte vs. gemessene Werte ULK

Kältemaschine (KM): Die Kältemaschine erhält vom ULK Block die geforderte Kühlleistung ($\varphi_{C,ULK}$). Zur Berechnung des elektrischen Energiebedarfs für dessen Kühlung benötigt die Kältemaschine vom Betreiber die Temperatur des Kühlwassers im Verflüssiger ($\theta_{c,in}$) und die Temperatur des Kaltwassers im Verdampfer ($\theta_{e,out}$). Zusätzlich benötigt das Modell der Kältemaschine Daten aus dem Datenblatt des eingesetzten Maschinentyps. Mit Hilfe dieser Eingaben lässt sich danach der Energieverbrauch der Kältemaschine im Modell berechnen.

Rückkühler (RK): Der Rückkühler erhält von der Kältemaschine die geforderte Kühlleistung ($\varphi_{C,KM}$). Zur Berechnung des elektrischen Energie-

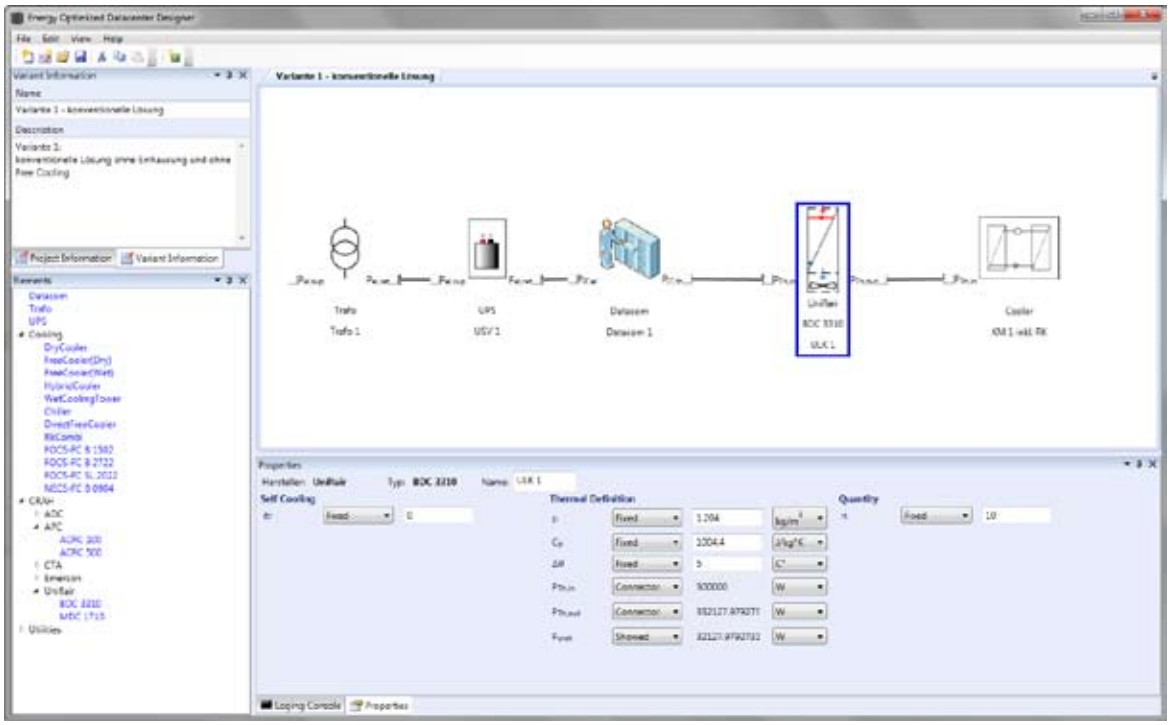


Abbildung 4: Variante eines Data Center im EoD Designer

bedarfs für dessen Kühlung benötigt die Kältemaschine vom Betreiber die Kühlwassertemperaturen im Vorlauf ($\theta_{W,rc,su}$) und im Rücklauf ($\theta_{W,rc,ret}$). Zusätzlich wird die Umgebungstemperatur (θ_{ODA}) benötigt. In den heute eingesetzten Systemen werden Kältemaschine und Rückkühler häufig kombiniert. Solche Kältemaschinen mit integriertem Rückkühler haben einen Anteil von 10% bis 20% am Gesamtenergieverbrauch des Data Centers.

Ergebnisse der Modelle

Die oben beschriebenen Blöcke berechnen für jede Stunde im Jahr die benötigte elektrische Leistung und stellen sie zur Analyse und zur grafischen Auswertung in Form eines Vektors für die weitere Verarbeitung zur Verfügung. Das Modell für die USV-Anlage berechnet die Verlustleistung (P_{USV}). Das Modell der ULK berechnet die benötigte elektrische Leistung der ULK (P_{ULK}). Dasselbe gilt für die Kältemaschine und das Rückkühlwerk, welche die benötigten elektrischen Leistungen (P_{KM} und P_{RK}) berechnen. Die Summe dieser Teilleistungen entspricht dem elektrischen Gesamtbedarf (P_{tot}) des Modells:

$$P_{tot} = P_{IT} + P_{USV} + P_{ULK} + P_{KM} + P_{RK}$$

Das Konsortium „The Green Grid“ hat einen weltweit anerkannten Wert für die Effizienz der eingesetzten elektrischen Energie in einem Data Center entwickelt [GG2]. Dieser Effizienzwert zeigt, wie gross der Anteil des Energieverbrauchs des Datacom Equipments im Vergleich mit den restlichen Systemen (Kühlsysteme, USV, etc.) ist. Dieser sogenannte „Power Usage Effectiveness“ (PUE) lässt sich mit Hilfe der Ergebnisse unseres Gesamtmodells wie folgt berechnen: $PUE = P_{tot}/P_{IT}$

Planungs- und Optimierungssoftware: EoD Designer

Die im vorhergehenden Abschnitt gezeigten Modelle für die Teilsysteme und das Gesamtsystem sind in eine Planungs- und Optimierungssoftware überführt worden. Die Software erlaubt eine Abschätzung des Energieverbrauchs eines gesamten Data Centers. Dabei können die fünf Teilsysteme jeweils einzeln konfiguriert werden. Dies erlaubt die Simulation verschiedener Varianten eines Data Centers und damit deren Vergleich. Der Planer kann dadurch die Auswirkungen der verschiedenen Varianten auf den Energieverbrauch des Data Centers darstellen. Eine solche Software ist nach unserem derzeitigen Wissensstand auf dem Markt noch nicht erhältlich. Die Firma R+B Engineering AG setzt unsere Software bereits für ihre Dienstleistungen ein und verschafft sich dadurch einen Wettbewerbsvorteil. In den nachfolgenden Abschnitten werden die beiden Hauptbestandteile der Software (Erstellen von Varianten und Reports) vorgestellt.

Erstellen von Varianten: Der Kernpunkt der Software bildet die Erstellung von verschiedenen Varianten von Data Centern. In einer Variante werden die fünf Teilsysteme individuell konfiguriert. Begonnen wird mit der Definition der Leistung des Datacom Equipments. Anschliessend wird die Energiezufuhr, bestehend aus Trafos und USV Anlagen, konfiguriert. Um die entstandene Wärme abzuführen, werden zusätzlich noch Umluftkühler und Kältemaschinen (inkl. Rückkühler) hinzugefügt. Eine Komponente kann da-

Abbildung 5: Report einer Variante im EoD Designer

bei mehrmals vorkommen, z.B. können im gleichen Data Center zwei verschiedene USV Anlagen eingesetzt werden. Die einzelnen Komponenten werden mit den für die Modelle benötigten Werten konfiguriert. Wie im Kapitel Gesamtmodell bereits beschrieben wurde, müssen dabei einige Werte vom Betreiber manuell definiert werden. Andere Werte werden automatisch aus bereits vorliegenden Datenblättern geladen. Einige Ausgaben der Komponenten dienen zudem direkt als Eingaben für anschliessende Komponenten. Der Betreiber hat zudem die Möglichkeit, Daten aus von ihm angefertigten Textdateien zu laden (z.B. die Wetterdaten eines Standorts). Die Software beinhaltet bereits viele vordefinierte Elemente (Komponenten). Der Benutzer hat aber die Möglichkeit, eigene Geräte hinzuzufügen. Dafür muss er lediglich eine neue Spezifikation mit einigen Grunddaten aus dem Datenblatt im XML Format erstellen. Dies gibt ihm auch die Möglichkeit, bestehende Geräte anzupassen. Ein Beispiel für eine Variante eines Data Centers wird in Abbildung 4 dargestellt.

Reports: Für jede Variante kann der Benutzer die Auswirkungen auf den Energieverbrauch auswerten. In einem solchen Report werden die Anteile des Energieverbrauchs der Teilsysteme gemessen am gesamten Verbrauch des Data Centers aufgelistet (prozentual und absolut als kWh). Eine Jahresbilanz zeigt den Energieverbrauch der einzelnen Systeme und des gesamten Data Centers pro Monat. Der Report berechnet zudem die Energiekosten pro Jahr und den international anerkannten PUE Wert. Damit kann das geplante Data Center mit anderen bereits bestehenden Data Centern verglichen werden. Neben der Jahresbilanz kann auch eine Auswertung pro Monat, in welcher jeder Wochentag ersichtlich ist, angezeigt werden. Der Report liefert somit dem Planer alle wichtigen und benötigten Informationen, um dem Kunden die Vor- und Nachteile jeder Variante aufzuzeigen. Der Kunde erkennt die Auswirkungen auf den Energieverbrauch und die damit verbundenen Energiekosten sofort, ohne dass weitere Berechnungen nötig sind. In Abbildung 5 wird ein solcher Report für eine Variante aufgezeigt.

Fazit

In diesem Projekt ist ein möglichst einfaches mathematisches Modell für den Energieverbrauch eines ganzen Data Centers entwickelt worden. Anhand von Messungen in verschiedenen operationellen Data Centern ist dieses verifiziert worden und dabei hat sich gezeigt, dass die Vorhersagen zwischen -5% und +10% vom tatsächlichen Energieverbrauch abweichen. Damit ist das Modell hinreichend genau, um vernünftige Abschätzungen zu liefern. Die auf der Modellierung aufgebaute Software gibt dem Planer die Möglichkeit, verschiedene Varianten für sein Data Center einander gegenüber zu stellen. Mit Hilfe der Auswertungen kann er die Varianten vergleichen und dem Kunden den im Zusammenhang mit der Energie effizientesten Vorschlag offerieren.

Referenzen

[ASH] ASHRAE: http://www.ashrae.org

[GG1] The Green Grid: http://www.thegreengrid.org

[GG2] The Green Grid, Data Center Power Efficiency Metrics (PUE): http://www.thegreengrid.org/~/media/WhitePapers/White_Paper_6_-_PUE_and_DCiE_Eff_Metrics_30_December_2008.pdf?lang=en

[SIA] SIA, Merkblatt 2044

Automatisierung von Systemtests im industriellen Umfeld

Continuous Integration Umgebungen werden in der Regel im Software-Entwicklungszyklus für die kontinuierliche Integration der Software eingesetzt. In diesem Artikel zeigen wir, dass sich solche Systeme auch hervorragend für die Realisierung von automatisierten Testinfrastrukturen im industriellen Umfeld eignen. Im vorliegenden Anwendungsfall wird damit eine fast vollständig automatisierte System- und Akzeptanztestumgebung von Softwareprodukten zur Überwachung und Steuerung von Messtechnik-Sensoren erreicht.

Anja Kellner, Martin Kropp | martin.kropp@fhnw.ch

Das Unternehmen, für das die Test-Automatisierungslösung entwickelt wurde, ist ein weltweit führender Hersteller von Mess- und Sensortechnik in der Schweiz. Immer wichtiger werden dabei für Kunden Komplettlösungen, welche einen zunehmenden Anteil an Softwareanwendungen zur Steuerung und Überwachung der Anlagen beinhalten. Die Software wird dabei weltweit in verschiedenen Sprachen und auf unterschiedlichen Windows-Betriebssystemen ausgeliefert.

Die hohen Qualitätsansprüche, welche das Unternehmen an seine Hardware stellt, gelten analog auch für die entwickelte Software, so dass das Testen der Software einen entsprechend hohen Stellenwert einnimmt.

Aktuell werden die Tests auf System- und Akzeptanztestniveau (Black-Box-Tests) durch manuelle Ausführung der zu testenden Applikation mittels umfangreichen schriftlichen Testspezifikationen ausgeführt. Die Testergebnisse werden ebenfalls manuell erfasst und dokumentiert. Die zu testende Applikation liegt in allen Fällen nur als Binär-Datei vor. Der Quellcode der Applikation wird bei diesen Tests nicht berücksichtigt.

Aufgrund der stetig grösser werdenden Software-Produktpalette wird das manuelle Testen immer zeitaufwendiger und damit auch kostspieliger. Aus diesem Grund sollen die Tests der Software in Zukunft weitestgehend automatisiert ausgeführt werden können. In diesem Beitrag beschreiben wir die Konzeption und den Aufbau einer kompletten Testinfrastruktur, die eine weitgehend automatisierte Durchführung von System- und Akzeptanztests ermöglicht.

Zielsetzungen

Durch eine vollständig automatisierte Testinfrastruktur soll in erster Linie der enorme Aufwand für die Systemtests gemindert werden. Zusätzlich werden damit folgende Ziele verfolgt:

- *Nightly-Tests* sollen eine automatisierte Ausführung der Tests über Nacht erlauben. Die sonst ungenutzten Rechnerressourcen können damit sinnvoll eingesetzt werden. Die Tests können manuell, über einen zeitlichen Trigger oder auch durch eine neue Version der zu testenden Applikation angestossen werden.
- *Multi-Plattform Tests* sollen es erlauben, die zu testenden Applikationen auf mehreren Betriebssystemen und in unterschiedlichen Internationalisierungen automatisiert zu testen.
- Da die Software nur in Kombination mit Hardware (Messgeräte) eingesetzt wird, müssen auch *"Real-World" Tests* durchgeführt werden. Dies bedeutet, dass die Tests immer mit realen Messgeräten durchgeführt werden. Hierfür soll die notwendige Infrastruktur aufgebaut werden.
- Die automatisierte *Archivierung der Testergebnisse* soll es erlauben, zu jeder Zeit die Ergebnisse von bereits ausgeführten Tests einzusehen. Dies ist insbesondere im Fall von Servicefällen wichtig, für die geprüft werden muss, mit welchem Ergebnis die Tests zuvor ausgeführt worden sind.
- Auch die *Reproduzierbarkeit* in Kombination mit der Verwendung von realer Hardware ist ein weiteres Ziel, mit der die Qualität der Tests verbessert werden soll.
- Schliesslich soll auch die Test-Implementierung historisiert werden, so dass die Entwicklung der Tests jederzeit rekonstruiert werden kann. Hierfür ist die Verwaltung des Testcodes in einem *Versionsverwaltungssystem* vorgesehen.

Das gesamte Testkonzept lässt sich in vier Teilbereiche unterteilen:

- Konzept der Testautomatisierung
- Definition der Infrastruktur
- Definition des Testprozesses
- Umsetzung des Testkonzeptes

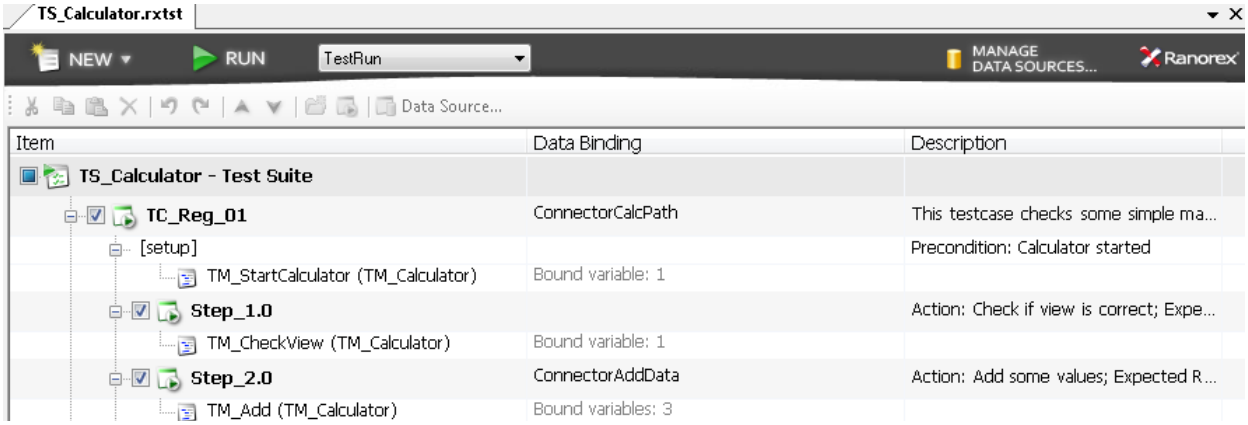


Abbildung 1: Screenshot: Ausschnitt Testsuite-View (Beispiel: Windows Calculator)

Testautomatisierung

Bei den durchzuführenden Tests handelt es sich um Systemtests, d.h. reine Black-Box-Tests, bei welchen die zu testende Applikation zur Ausführung kommt. Für die Automatisierung dieser Tests bieten sich idealerweise GUI-Testautomatisierungs-Werkzeuge an. Diese ermöglichen es, auf dem GUI der zu testenden Applikation Aktionen auszuführen und danach den Zustand zu prüfen. Die meisten dieser Werkzeuge verfügen auch über einen Capture-Replay Mechanismus, der es ermöglicht, Bedienabläufe aufzuzeichnen, als Skript anzupassen und ablaufen zu lassen.

Bei der Evaluation verschiedener GUI-Testwerkzeuge hat sich das Tool Ranorex [Ran] als das am besten geeignete durchgesetzt. Dieses Produkt ist insbesondere in Bezug auf die Objekterkennung der graphischen Steuerelemente auf der GUI der zu testenden Applikationen den anderen evaluierten Produkten überlegen.

Die Objekterkennung von Ranorex basiert auf XPath[1]-ähnlichen Pfaden. Jedes Objekt auf der GUI der zu testenden Applikation ist über einen solchen Pfad eindeutig identifizierbar. Diese Pfade werden über das sogenannte *Object-Repository* den entsprechenden GUI-Objekten zugeordnet.

Ranorex *Testmodule* bilden die möglichen Aktionen ab und werden in C# implementiert. Die neueste Version Ranorex 3.x verfügt zudem über eine sogenannte *Testsuite-View* (siehe Abb. 1), die es erlaubt mittels Drag&Drop Testmodule in beliebiger Reihenfolge auszuführen. Die Testmodule sind in der linken „Item" Spalte in einer Baumstruktur mit dem Präfix „TM_" referenziert.

Infrastruktur

Die Infrastruktur für die Testautomatisierung soll möglichst vor äusseren nicht-beeinflussbaren Einflüssen geschützt sein. Aus diesem Grund empfiehlt sich ein separates Netzwerk für die Testautomatisierung. Die Abbildung 2 zeigt das im konkreten Anwendungsfall realisierte Testnetzwerk:

- *File-Exchange-Server*: Mit diesem können Daten zwischen dem Firmennetzwerk und dem Automatisierungsnetzwerk ausgetauscht werden. Insbesondere dient dieser Server dazu, die Installationspakete der verschiedenen Produkte und Produktversionen zu verwalten.
- *Proxy*: Dieser dient als Schnittstelle nach aussen (Zugriff auf Lizenzserver, Internet und das Versionsverwaltungssystem).
- *Rack mit Automation-PC*: Die so genannten Racks erfüllen zwei zentrale Funktionen:
- Zum einen enthalten sie die Hardware (Messgeräte, Bussysteme,...), für die Ausführung der *„Real-World"-Tests*. Ausserdem sind sie mit einem weiteren Computer ausgerüstet, auf dem die Tests effektive zur Ausführung kommen. Dieser Rechner verfügt über zwei Netzwerkkarten: die eine ist mit dem Automatisierungsnetzwerk verbunden und die andere mit den Geräten im Rack (privates Netzwerk des Racks). Der Automation-PC ist als sogenannter *headless* Computer aufgesetzt, verfügt also weder über einen Bildschirm, noch Maus oder Tastatur.
- *Remote-PC*: Dieser PC dient dazu, per Remote-Verbindung auf die Automation-PCs zuzugreifen. Dies kann nötig sein, um den Status des Rechners zu prüfen, aber auch für die Entwicklung und Ausführung von Testskripten.

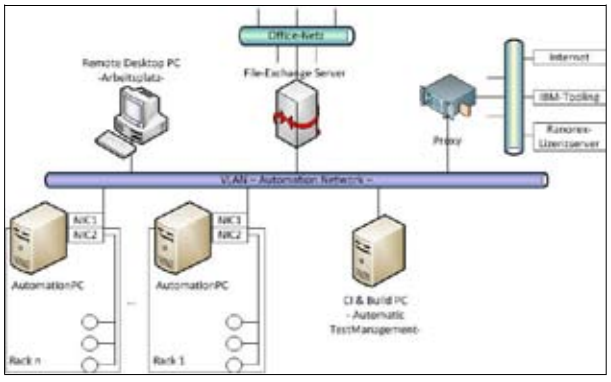


Abbildung 2: Netzwerk für Testautomatisierung

1 http://www.w3.org/TR/xpath/

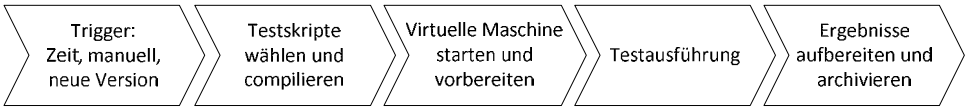


Abbildung 3: Realisierter Testprozess. Nur der erste Schritt wird auf dem CI-Server ausgeführt, die nachfolgenden auf einem Automations-PC.

- *CI- & Build-PC*: Auf diesem Computer ist die Continuous Integration Umgebung installiert. Er ist somit der Master, der die Ausführung der automatisierten Tests auf den Automation-PCs koordiniert und die zu testenden Applikationen auf die Automation-PCs verteilt.

Für die Testcode-Verwaltung wird das Versionsverwaltungssystem *ClearCase UCM* von IBM eingesetzt [IBM-CC], welches bei dem Unternehmen schon seit längerem im Einsatz ist. Es kann somit auf den Testcode zu jeder beliebigen Produktversion der zu testenden Applikation zugegriffen werden.

Um die Software-Produkte des Unternehmens in verschiedenen Sprachen und Betriebssysteme testen zu können, kommen bereits vorkonfigurierte virtuelle Maschinen zum Einsatz. Dies ermöglicht die Simulation der unterschiedlichsten System-Setups *(Multi-Platform-Tests)*. Zusätzlich bietet dieser Ansatz die Möglichkeit, die virtuelle Maschine gemeinsam mit den Testergebnissen so zu archivieren, dass damit auch die Umgebung der Tests selbst reproduzierbar ist. Die virtuellen Maschinen werden automatisiert auf die Automation-PCs der Racks geladen und gestartet.

Testprozess

Der gesamte Testprozess, vom Holen des benötigten Test-Quellcodes bis zur Archivierung der Testreports, soll vollständig automatisiert werden. Dabei soll die automatisierte Ausführung sowohl manuell als auch ereignisgesteuert ausgelöst werden können. Ereignisse können z.B. zu einer bestimmten Uhrzeit (Nightly-Build), oder durch das Vorliegen einer neuen Produktversion auf dem File-Exchange-Server ausgelöst werden. Der Prozess soll sowohl für alle vorhandenen, aktuellen Produkte als auch für eine ausgewählte Produktversion angestossen werden können.

Für die Automatisierung der Systemtests eignet sich eine Continuous Integration (CI) Umgebung wie sie üblicherweise für die Software-Integration zum Einsatz kommt, da beide Prozesse einige Gemeinsamkeiten aufweisen:

- Der benötigte Testcode muss in beiden Fällen aus einem Versionsverwaltungssystem geholt werden.
- Das Kompilieren des Testcodes ist beiden gemein.
- In beiden Fällen werden die Software-Tests ausgeführt. Während dies bei Software-Entwicklungsprojekten jedoch in der Regel Unit- und Integrationstests sind, werden im gegebenen Fall die Systems-Tests auf den bereits fertigen und automatisiert installierten Applikationen ausgeführt.
- Als Resultat wird in beiden Fällen ein Testreport erstellt.

Die Ausführung des gesamten Testprozesses (gemäss Abb. 3) wird über ein Continuous Integrati-

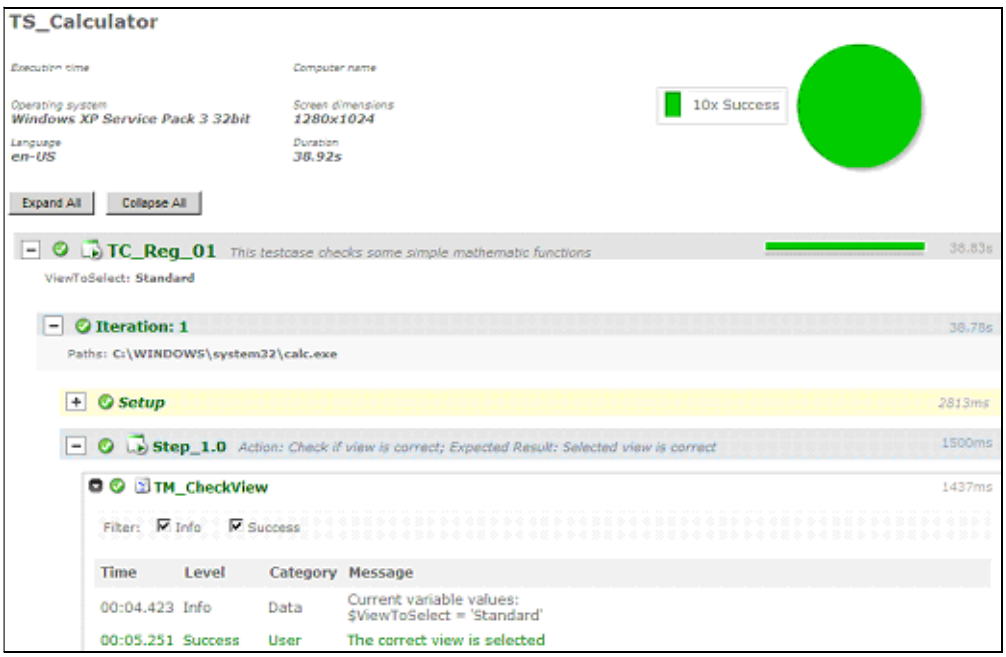


Abbildung 4: Screenshot: Ausschnitt eines erfolgreichen Testreports (Beispiel: Windows Calculator)

Abbildung 5: Jenkins Projektübersicht

on Werkzeug gesteuert. Die konkrete Umsetzung wird nachfolgend beschrieben:

1. Trigger: Die Test-Ausführung wird auf dem CI-Server entweder durch einen zeitlichen Trigger (*Nightly-Tests*), manuellen Trigger oder durch eine neue Version der zu testenden Software auf dem File-Exchange-Server angestossen.
2. Testskripte: Der Testcode wird aus dem Versionsverwaltungssystem geholt. Mittels MS-Build wird der Testcode kompiliert und zu ausführbaren Tests.
3. Virtuelle Maschine: Die benötigte virtuelle Maschine wird im gewünschten Zustand gestartet, so dass die Voraussetzungen für die Tests erfüllt sind.
4. Ausführen der Tests: Die Ausführung der Tests findet in einer virtuellen Maschine statt, auf welche über ein Remote-Excecution-Werkzeug zugegriffen wird.
5. Ergebnis Aufbereitung: Nach Ablauf aller Tests werden die Testergebnisse für ein geeignetes Reporting aufbereitet. Die Abbildung 4 zeigt am Beispiel von Ranorex, wie die Testergebnisse innerhalb von Jenkins verlinkt und mittels Dashboard angezeigt werden können. Zusätzlich werden die Testergebnisse im Versions-Verwaltungssystem abgelegt.

Umsetzung des Testkonzeptes mittels Continuous Integration

Zur Steuerung und Ausführung des gesamten Test-Prozesses wird ein Continuous Integration Werkzeug eingesetzt. Dabei kommt das Open Source Produkt Jenkins zum Einsatz [Jen]. Ausschlaggebend für dessen Wahl sind die grosse Funktionalität, die sehr gute Integration mit dem verwendeten Versionsverwaltungssystem, sowie die sehr gute Erweiterbarkeit mittels Plugins. Es existiert eine grosse Entwicklergemeinde, die sich sehr aktiv an der Weiterentwicklung des Systems beteiligt.

Jenkins selbst wird typischerweise auf einem eigenen Server installiert und betrieben. Jedes Projekt wird in Jenkins als separater *Job* definiert. Jeder Job kann separat konfiguriert werden und so für jedes Projekt der individuelle Build-Prozess definiert werden. Hierfür steht in Jenkins ein sehr komfortables Web Interface zur Verfügung.

Die Abbildung 5 zeigt einen Ausschnitt einer Projektübersicht von Jenkins für eine Umgebung mit 5 Projekten. Die Übersicht zeigt durch entsprechend eingefärbte Ballons für jedes Projekt an, ob der letzte Build-Durchlauf erfolgreich war, sowie die Stabilität des Build-Prozesses über die letzten

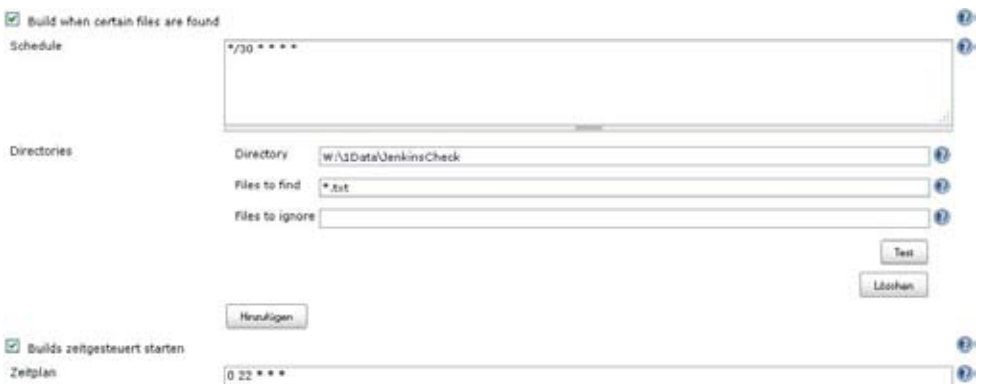


Abbildung 6: Build Trigger Konfiguration

Abbildung 7: Konfiguration des Build- und Test-Prozesses

5 Durchläufe mit Hilfe eines „Wetter"-Symbols in der Kolonne „W".

Die Abbildungen 6 bis 8 zeigen exemplarisch wie mittels Konfiguration in Jenkins der zuvor beschriebene Testprozess schrittweise umgesetzt wird.

Abbildung 6 zeigt die beiden Optionen zum Auslösen des Build-Prozesses. Der dateibasierte Trigger prüft alle 30 Minuten, ob eine bestimmte Datei (in diesem Fall ein beliebigesTextdokument) in einem ebenfalls spezifizierten Verzeichnis vorhanden ist (Hinweis: dies kann durch das zusätzlich installierte Plugin „Files Found Trigger" erreicht werden). Der zeitgesteuerte Trigger sorgt dafür, dass das Projekt mindestens jede Nacht um 0:22 Uhr gebildet wird.

Die Abbildung 7 zeigt die Konfiguration zur Ausführung des eigentlichen Build- und Test-Prozesses. Jenkins erlaubt die Konfiguration von beliebig vielen Einzelschritten. In Abbildung 7 sehen wir, dass zunächst die MSBuild-Datei zur Kompilation des Projektes ausgeführt wird. Anschliessend wird die vorkonfigurierte virtuelle Maschine auf dem Automation-PC gestartet und danach mittels Batch-Skript die Testausführung angestossen.

Die im Testprozess beschriebene Archivierung und Veröffentlichung der Testergebnisse wird in Jenkins mit Hilfe sogenannter „Post-Build" Aktionen konfiguriert wie dies in Abbildung 8 dargestellt ist.

Um die vollständige Testausführung während der Nacht überhaupt erst zu ermöglichen, sollte die Testausführung möglichst parallel und gleichzeitig auf den verschiedenen Automation-PCs durchgeführt werden. Für die gleichzeitige parallele Ausführung der Tests bietet Jenkins das Master/Slave-Konzept [Jen2].

Dieses Konzept erlaubt es mittels Jenkins-Konfiguration weitere Rechner als sogenannte Knoten in das System aufzunehmen. Dazu wird auf dem gewünschten Rechner ein Jenkins-Service gestartet, der es erlaubt diesen als Knoten in die CI-Umgebung aufzunehmen. Nun kann für jedes Projekt dezidiert angegeben werden, auf welchem Knoten das Projekt gebildet und ausgeführt werden soll. Wird für ein bestimmtes Projekt der Trigger aktiv und stellt fest, dass das Projekt ausgeführt werden soll, so wird die komplette Ausführung an den entsprechenden Knoten delegiert wie dies in Abbildung 9 dargestellt ist. Im gegebenen Fall werden die Automation-PCs aus den Racks als Knoten registriert, sodass diese parallel Tests ausführen können.

Abbildung 8: Archivierung und Veröffentlichung

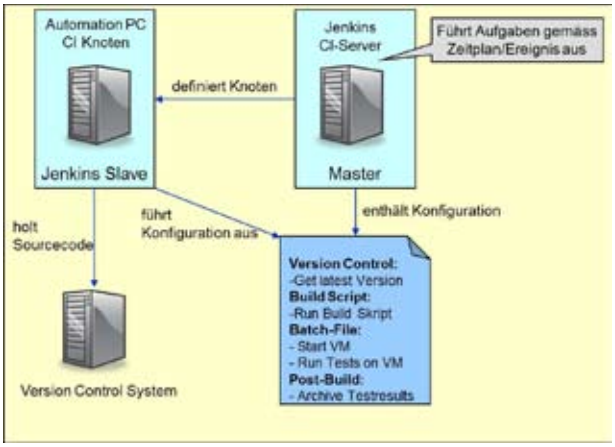


Abbildung 9: Jenkins Master/Slave Konzept

Erste Erfahrungen und Ausblick

Mit dem beschriebenen Testkonzept ist es möglich, sehr viele der bisher manuell ausgeführten Tests automatisiert auszuführen. Durch Nightly-Tests können die sonst ungenutzten Rechnerressourcen über Nacht sinnvoll genutzt werden. Die Analyse der Testergebnisse muss allerdings noch manuell vorgenommen werden, da sonst eventuell auch Fehler in ein Bug-Tracking-System eingetragen werden würden, die nichts mit den Testergebnissen, sondern mit dem gesamten Prozess zu tun haben. Beispiele solcher Fehler sind das Fehlschlagen eines Builds, eine falsche oder fehlerhafte virtuelle Maschine. Der Aufwand der Analyse ist jedoch vertretbar, wenn er mit dem Aufwand manueller Tests verglichen wird.

CI-Umgebungen scheinen auch den Anforderungen für automatisierte System- und Akzeptanztests gerecht werden zu können. Eine Umsetzung mit diesen Systemen ermöglicht es, dass die Test-Quellcodes immer zur Laufzeit kompiliert werden können. Dies stellt sicher, dass immer die aktuellste Version des Testcodes verwendet wird. Zusätzlich ist es möglich, eine Umgebung für die Tests zu schaffen, in der auch Hardware und unterschiedliche Betriebssysteme verwendet werden können

Der bisherige Einsatz hat gezeigt, dass die Ausführung der Tests mittels Remote-Commandline Werkzeugen in der virtuellen Maschine verbessert werden kann. Nach Beendigung der Tests liefert die Remote-Anwendung immer eine erfolgreiche Testausführung zurück, auch im Falle eines Test-Failures. Das tatsächliche Testergebnis kann nur durch Analyse des Test-Reports ausgelesen werden. Hier wäre es denkbar, auch die virtuelle Maschine an sich als Slave zu registrieren und das Build-Projekt in mehrere Teile aufzuteilen, die nacheinander auf jeweils unterschiedlichen Slaves ausgeführt werden. Dies bedeutet allerdings, dass die virtuelle Maschine entsprechend vorbereitet werden müsste.

Referenzen

[IBM-CC] IBM Homepage: http://www-01.ibm.com/software/awdtools/clearcase/, 15.08.2011

[Jen] Jenkins, Continuous Integration: http://www.jenkins-ci.org/, 15.08.2011

[Jen2] Kohsuke Kawaguchi, Distributed Builds: https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds, 15.08.2011

[Ran] Ranorex Homepage: http://www.Ranorex.com, 15.08.2011

Highspeed Videoverarbeitung

Hochgeschwindigkeitskameras werden oft zur Überwachung sehr schneller, vollautomatisierter, industrieller Prozesse eingesetzt. Im Zuge der Marktanforderung, dass die einmal aufgenommenen Prozesse immer schneller zu analysieren sind, soll das aufgenommene Bildmaterial nicht nur zur Dokumentation des Ablaufes dienen, sondern soll vielmehr Bildverarbeitung in Echtzeit verwendet werden. Dabei werden die Bilddaten mit hoher Bildrate (z.B. 200 Bilder pro Sekunde) an einen Rechner übertragen, auf diesem mit neu entwickelten, parallelen, adaptiven Algorithmen analysiert und bei Bedarf Ereignisse in Echtzeit ausgelöst, welche den Automationsprozess steuern. Die Herausforderungen dabei liegen primär in der effizienten Verarbeitung der hohen Datenmengen, welche in Zukunft durch ansteigende Bildraten noch erhöht werden sollen.

Martin Schindler, Christoph Stamm | christoph.stamm@fhnw.ch

In diesem Beitrag beschreiben wir ein gemeinsames Projekt mit dem in Baden-Dättwil ansässigen Unternehmen AOS Technologies. AOS ist ein Anbieter von Hochgeschwindigkeitskameras und dazu passender Software. Die Software-Reihe PROMON von AOS dient der optischen, industriellen Prozessüberwachung [PRO]. Unser Projekt Promon200[1] knüpft an PROMON an, mit dem Hauptziel, die übertragenen und/oder abgespeicherten Videodaten automatisch nach Unregelmässigkeiten in den überwachten Prozessen abzusuchen.

Die belgische Firma Tesin bietet mit ihrer Cyclocam[2] ein mobiles System bestehend aus einer Hochgeschwindigkeitskamera und einem Display an, um zyklische, industrielle Prozesse aufzuzeichnen, direkt vor Ort zu begutachten und eventuell anzupassen. Um ein Hochgeschwindigkeitsvideo mit beispielsweise 1000 Bildern pro Sekunde bei normaler Videobetrachtungsgeschwindigkeit von 25 Bildern pro Sekunde betrachten zu können, werden also 40 Sekunden Betrachtungszeit pro einer Sekunde Aufnahmedauer benötigt. Daher macht es Sinn, uninteressante Teile des Videos automatisch zu überspringen und so die Betrachtungszeit möglichst gering zu halten. Welche Passagen einer Sequenz für den Betrachter von Interesse sind, legt er selber mithilfe von Markierungen fest.

Unser Projekt unterscheidet sich von der Cyclocam dahingehend, dass wir nicht primär uninteressante Teile des Videos überspringen wollen, sondern mehrere Sequenzen eines zyklischen Prozesses vergleichen wollen, um wichtige Unterschiede zwischen den Sequenzen, also Unregelmässigkeiten im Ablauf automatisch feststellen zu können. Zudem wollen wir die Sequenzlänge anhand der sich wiederholenden Bilder möglichst genau selber ermitteln.

Aus verschiedenen Untersuchungen ist bekannt, dass Menschen bei der Betrachtung von zyklischen und langandauernden Videoaufnahmen sehr schnell ermüden und somit an Konzentration einbüssen. Dadurch passiert es schnell, dass die relevanten Unregelmässigkeiten im Ablauf schlicht übersehen werden. Hier kann ein automatisiertes System Abhilfe schaffen, falls es gelingt, dem System klar zu machen, was akzeptable und was unerlaubte Prozessabweichungen sind.

In sehr schnell ablaufenden Prozessen werden hohe Bildraten (> 200 Bilder pro Sekunde) und möglichst kurze Belichtungszeiten benötigt, um alle relevanten Details einzufangen und Bewegungsunschärfen zu vermeiden. Solche hohe Bildraten erhöhen die Schwierigkeit der Aufgabenstellung der automatisierten Prozessüberwachung wesentlich, weil für die durchschnittliche Bearbeitungszeit pro Bild weniger als 5 ms zur Verfügung stehen. Heutige Hochleistungs-PCs sind zwar in der Lage, innerhalb von 5 ms wenige Millionen von Grundoperationen auszuführen. Doch darf man dabei nicht übersehen, dass ein Bild oft aus einer Million oder mehr Bildpunkten besteht. Dies relativiert die Grösse der Anzahl Operationen, die ausgeführt werden können sehr schnell.

Ein automatisiertes, optisches Prozessüberwachungssystem für sehr schnell ablaufende Prozesse muss also in der Lage sein, innerhalb kürzester Zeit korrekte von falschen Prozessabläufen zu unterscheiden. Dies kann beispielsweise auf drei verschiedene Arten von statten gehen:

- In einem ersten Ansatz werden vom Operateur interessante Bildregionen markiert und mit einer Bildverarbeitungsoperation versehen. Eine solche Operation kann beispielsweise die Helligkeit im markierten Bereich bestim-

1 Das Projekt Promon200 ist vom Aargauer Forschungsfonds finanziell unterstützt und begleitet worden.
2 Tesin N.V. Cyclocam: http://www.tesin.be/

men oder die Frequenz einer aufleuchtenden Lampe ermitteln. Die Ausgaben dieser Bildverarbeitungsoperationen können dann mit Schwellwerten verglichen, zu Booleschen Resultaten umgewandelt und schliesslich mit Booleschen Operatoren verknüpft werden. So entsteht am Ende einer Verarbeitungskette ein Signal, welches angibt, ob ein Bild zu einem fehlerhaften Ablauf gehört.

- In einem zweiten Ansatz wird dem System anhand von Videosequenzen gezeigt, wie korrekte und wie falsche Abläufe aussehen können. Diese Sequenzen werden dann dazu verwendet, um den aktuellen Ablauf bildweise mit den gegebenen Sequenzen zu vergleichen. Bei Überschreiten einer gewissen Vergleichstoleranz werden die Bilder markiert und dem Operateur für eine genauere Überprüfung vorgelegt.
- In einem dritten Ansatz versucht das System selbständig die Repetitionen im Ablauf zu ermitteln. Wenn dies gelingt, so können entweder die Einzelbilder eines Ablaufs mit den entsprechenden Einzelbildern des vorgängigen Ablaufs automatisch auf grössere Abweichungen untersucht werden oder es wird über die ganze Sequenz hinweg eine Kennzahl berechnet, welche dann mit den bisherigen Kennzahlen der vorangegangenen Sequenzen verglichen werden kann.

Um einen der drei genannten Ansätze in Echtzeit ausführen zu können, braucht es äusserst effiziente Algorithmen, welche die Parallelität heutiger Computersysteme voll ausnutzen können und darüber hinaus in der Lage sind, die enormen Rechenkapazitäten, welche auf Grafikprozessoren zur Verfügung gestellt werden, für die eigenen Zwecke zu nutzen.

Die Verwendung vorgefertigter Bildverarbeitungsbibliotheken oder -komponenten von bekannten Anbietern wie beispielsweise *MathWorks*[3] oder *National Instruments*[4] wird in diesem Projekt aus dreierlei Gründen ausgeschlossen: Erstens soll PROMON ein eigenständiges Produkt bleiben, welches nicht in wesentlichen Teilen von Fremdkomponenten abhängen darf und zweitens sollen die neusten Entwicklungen im Bereich des *General Purpose Computing on Graphics Processing Units* (GPGPU) ohne lange Entwicklungszyklen in die bestehende Software einfliessen können. Schliesslich drittens, dient das vorliegende Projekt auch zu einem grossen Teil dem Knowhow Transfer im Bereich der parallelen Bildverarbeitung zwischen Hochschule und Industrie.

Sitzt der Schraubverschluss richtig?

Unter dem Begriff der industriellen Prozessüberwachung lässt sich Verschiedenes verstehen. Daher ist es angebracht, an dieser Stelle ein typisches Anwendungsszenario vorzustellen.

In einer Abfüllanlage für Getränkeflaschen soll der Prozess des Verschliessens von Flaschen mit Schraubverschluss überwacht werden. Bei diesem Ablauf kommt es immer wieder vor, dass Deckel beim Aufschrauben kaputt gehen oder dass bereits defekte Deckel auf die Flaschen aufgeschraubt werden. Die Aufgabe einer automatischen Prozessüberwachung besteht also darin, defekte Schraubverschlüsse zu detektieren.

Eine Hochgeschwindigkeitskamera mit passender Streaming- und Überwachungs-Software, wie beispielsweise PROMON, kann also dazu verwendet werden, um die problematischen Teile des Prozesses auf Video aufzunehmen. Da oft nicht im Vornherein klar ist, zu welchem Zeitpunkt es zu einer Fehlfunktion kommen wird, müssen über einen grösseren Zeitraum (z.B. 10 Stunden) Aufnahmen gemacht werden, die dann später, also offline, einer genauen Analyse unterzogen werden. Damit das menschliche Auge die fehlerhaften Abläufe überhaupt wahrnehmen kann, müssen Videoaufnahmen mit einer Bildrate von 200 Bildern pro Sekunde um einen Faktor sechs bis acht verlangsamt werden. Diese manuelle Durchsicht ist ein sehr zeitaufwändiger und ermüdender Vorgang. Daraus ergibt sich die Forderung nach einer automatisierten Lösung, welche die fehlerhaften Stellen möglichst autonom auffinden kann und dabei nicht mehr Zeit benötigt als ein Mensch.

Promon200

Promon200[5] versucht die zuvor genannten Ziele zu erreichen und darüber hinaus, den Übergang von der offline zur online Überwachung zu ermöglichen. Eine online Überwachung setzt nun aber voraus, dass die Bildanalyse mit der Bildrate der Hochgeschwindigkeitskamera Schritt halten kann. Eine zuverlässige online Überwachung erlaubt also nicht nur das viel schnellere Erkennen von Fehlabläufen im Prozess, sondern ermöglicht die Resultate der Echtzeitanalyse auch für die adaptive Anpassung und Optimierung des laufenden Prozesses zu verwenden.

Promon200 ist visuelles Werkzeug (Abb. 1). Es besteht im Wesentlichen aus vier Teilen: auf der linken Seite sind die vorhandenen Bildverarbeitungsalgorithmen, Booleschen Operatoren und Komparatoren aufgelistet; in der Mitte ist der Designer ersichtlich, mit dem die verschiedenen Bildverarbeitungsaufgaben zu einem Gesamtsystem zusammengebaut werden können; auf der rechten Seite oben ist das Livevideobild ersicht-

3 MathWorks, Computer Vision System Toolbox: http://www.mathworks.ch/products/computer-vision/?s_cid=global_nav

4 National Instruments, Vision Builder for Automated Inspection: http://www.ni.com/vision/vbai.htm

5 Promon200 ist die Software, welche im gleichnamigen Projekt innerhalb zwei Jahren entwickelt worden ist.

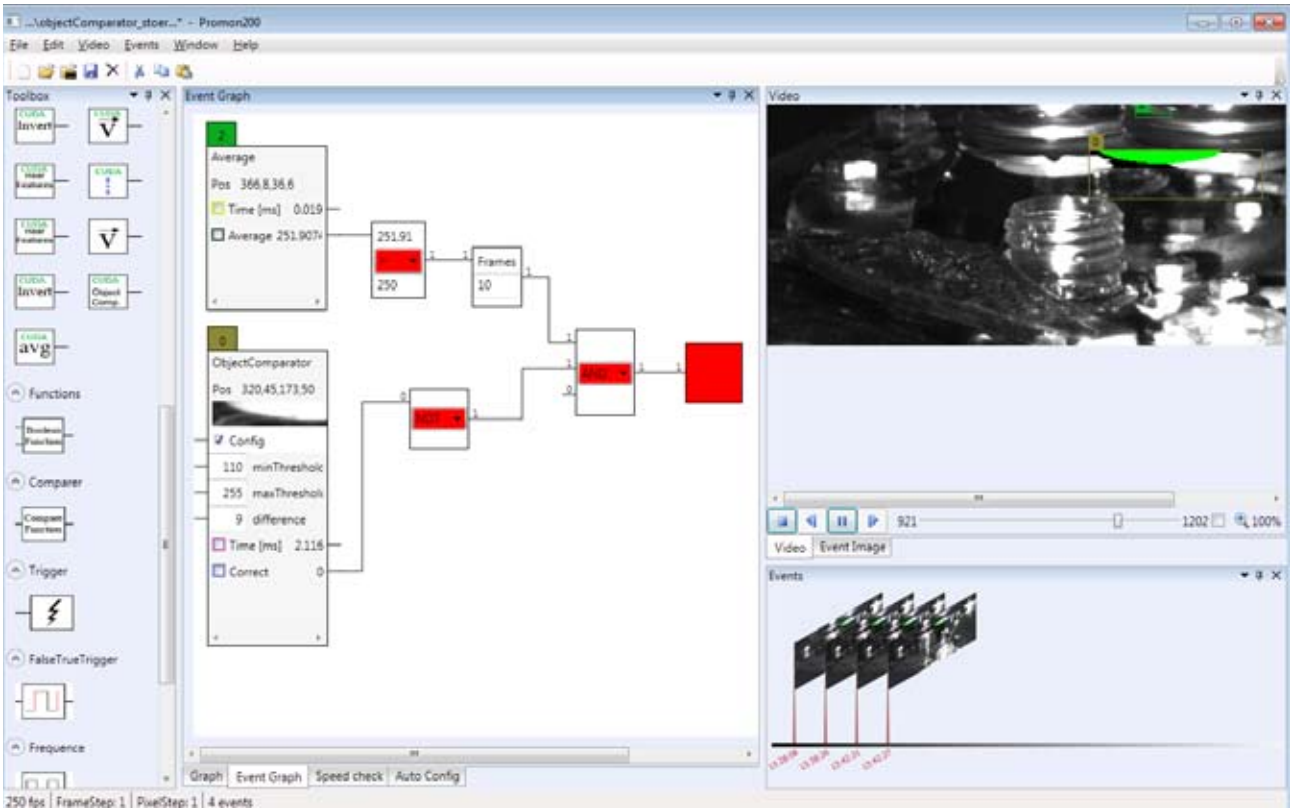


Abbildung 1: Defekter Deckel löst in Promon200 ein Triggersignal aus. Der aktuelle Zustand wird mit allen Messwerten als Ereignis abgespeichert.

lich und darunter sind die ausgelösten Ereignisse in Form von Flaggen aufgelistet.

Die Filterkette innerhalb des Designers zeigt, dass zwei unterschiedliche Regionen (grün und braun markiert) im Video überwacht werden. Sie ist vom Benutzer der Software mittels Drag-and-Drop zusammengestellt worden. Sobald die Ausgänge der beiden Filter einen benutzerdefinierten Schwellwert über- bzw. unterschreiten, wird ein Trigger-Signal ausgelöst, welches ein Ereignis generiert. Diese Ereignisse (Videoframe und der Zustand der Filterkette mit den zum Ereignis abgespeicherten Werten) können entweder in Echtzeit zur Prozesssteuerung dienen oder zu einem späteren Zeitpunkt abgerufen, analysiert und zur Verbesserung des Produktionsablaufs verwendet werden. Da Promon200 über eine Plug-in-Architektur verfügt, können nachträglich neu entwickelte Bildverarbeitungsoperationen (Filter) dem System einfach hinzugefügt werden, ohne dass das gesamte Framework aktualisiert werden muss.

Auf die folgenden drei Aspekte von Promon200 werden wir in diesem Artikel genauer eingehen:

- Detektion von Anomalien in repetitiven, industriellen Prozessen;
- adaptive Echtzeitalgorithmen mit automatischer Parametrisierung;
- Video- bzw. Bildverarbeitung mit Hilfe der GPU.

Erkennung von Anomalien

In der Einleitung haben wir bereits drei grundsätzliche Arten vorgestellt, wie Anomalien in sich wiederholenden, industriellen Prozessen detektiert werden können. Hier wollen wir vor allem auf den dritten Ansatz eingehen, wo das System selbständig die Repetitionen im Ablauf zu ermitteln versucht und dann mithilfe dieser Sequenzlänge in der Lage ist, eine charakteristische Kennzahl für die Sequenz zu ermitteln. Diese Kennzahl lässt sich dann einfach mit denen der vorangegangenen Sequenzen auf grössere Abweichungen untersuchen. Der grosse Vorteil dieses Ansatzes gegenüber dem zweiten Ansatz mit den vorgegebenen korrekten und falschen Sequenzen ist, dass nur die *Region of Interest* gewählt und nicht zuerst eine Datenbasis von falschen und korrekten Sequenzen erstellt werden muss. Zudem schränkt der dritte Ansatz die Arten von Anomalien, welche erkannt werden können, nicht von vornherein ein. Anderseits hängt seine Qualität wesentlich davon ab, ob es gelingt, die Sequenzlänge genau zu ermitteln und ob die Funktion zur Ermittlung der Kennzahl sensitiv genug ist.

Beim ersten Ansatz werden vom Operateur interessante Bildregionen markiert und mit untereinander verknüpften Bildverarbeitungsoperationen versehen. Hierbei geht man davon aus, dass bereits falsche Sequenzen vorliegen oder dass mit der Anomalie verbundene Nebenerscheinungen bekannt sind, welche sich detektieren lassen. Der Operateur leistet hier im Wesentlichen

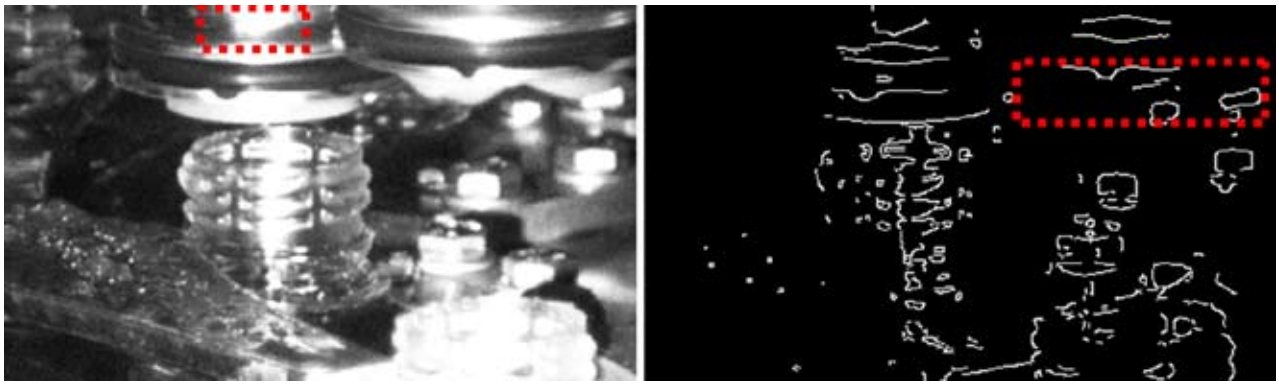


Abbildung 2: links: Graustufenbild mit markierter Region zur Frequenzdetektion; rechts: Kantenbild mit benutzter Region zur Berechnung der HOG-Feature-Vektoren.

eine vorgängige Analysearbeit, um signifikante Merkmale zu finden, anhand derer eine Anomalie erkannt werden kann. Eine solche Analyse ist oft schwierig und zeitaufwendig, kann aber durchaus auch bereits erste Hinweise zur Verbesserung des industriellen Prozesses bringen und somit sehr erwünscht sein. Ein weiterer kleiner Vorteil gegenüber dem selbstlernenden Ansatz ist, dass auf eine Bestimmung der korrekten Sequenzlänge verzichtet werden kann. Der grosse Nachteil hingegen ist wieder wie vorher, dass nur bereits bekannte Problemfälle detektiert werden.

Da der dritte Ansatz ein selbstlernender ist, braucht es einen gewissen Vorlauf, eine erste Lernphase von ein paar Hundert Bildern, um die Wissensbasis aufzubauen, welche anschliessend verwendet wird. Eine solche Lernphase kann in gewissen Situationen störend sein, ist in unserem Ansatz aber nicht weiter störend, da bereits für die Anpassung der Parametrisierung an die vorhandene Hardware eine Kalibrierungsphase vorgeschaltet wird.

Wie bereits erwähnt, müssen wir also im dritten, selbstlernenden Ansatz eine sensitive Funktion für die Sequenz zur Verfügung stellen, welche relevante Veränderungen zwischen den Sequenzen anzeigen kann. Eine solche Funktion basiert oft auf einem oder mehreren *Features*, wie zum Beispiel die durchschnittliche Helligkeit, oder das *Histogram of oriented gradients*[6]. In unserer sensitiven Funktion verwenden wir das HoG, weil wir davon ausgehen, dass sich der Inhalt eines Bildes aufgrund der darin enthaltenen Kantenrichtungen gut charakterisieren lässt.

Die Intensitäten und Richtungen von Kanten können mithilfe der ersten Ableitung über ein Bild einfach berechnet werden (Abbildung 2 rechts zeigt ein Kantenbild). Aus den Kantenrichtungen lässt sich anschliessend für alle Bildpunkte mit relevanter Kantenintensität eines ausgewählten Bereiches ein Histogramm mit acht Richtungsklassen (das HoG) erstellen (Abb. 3). Pro Video bild entsteht dadurch ein charakteristischer Vektor der Länge acht. Alle solchen Vektoren einer Sequenz werden dann in eine konstante Anzahl k Blöcke unterteilt, von denen jeweils ein Minima- und Maximavektor der Länge acht über alle Bilder im Block ermittelt werden. Ein Minima- bzw. Maximavektor gibt also den Minimal- bzw. Maximalwert für jede der acht Richtungen des HoG-Vektors aus den k Bildern des Blocks an. Pro Sequenz werden somit insgesamt $2 \cdot 8 \cdot k$ HoG Werte in einem Sequenz charakterisierenden Vektor zusammengefasst. Dieser Vektor ist der Rückgabewert der charakterisierenden Funktion.

Während der Lernphase wird ein entsprechender Referenzvektor der gleichen Länge $2 \cdot 8 \cdot k$ aus allen Frames der Lernphase gebildet. Dieser Referenzvektor wird mit dem Sequenz charakterisierenden Vektor der ersten Sequenz initialisiert und dann nach jeder Sequenz aktualisiert, wobei bei den Werten aus den Minimavektoren weiterhin das Minimum und bei den Werten der Maximavektoren das Maximum verwendet werden. Nach der Lernphase wird dann für jede Sequenz die charakterisierende Funktion berechnet und mit dem Referenzvektor verglichen. Solange die in der Sequenz bestimmten Min-Max-Wertebereiche vollständig innerhalb der Min-Max-Wertebereiche des Referenzvektors liegen, wird die Sequenz als normal betrachtet, d.h. sie weicht nicht wesentlich von den Sequenzen während der Lernphase ab. Nur wenn alle Sequenzen der Lernphase als korrekt bezeichnet worden sind, darf

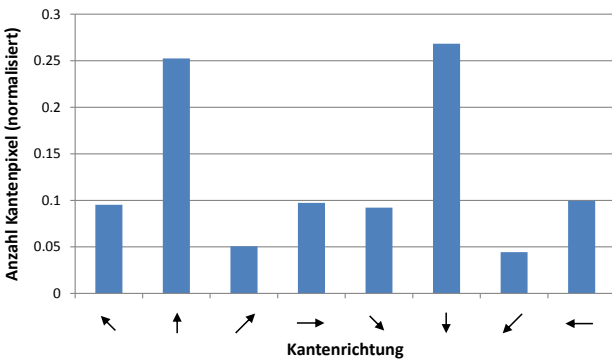


Abbildung 3: Histogramm der Gradienten (HOG)

6 http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

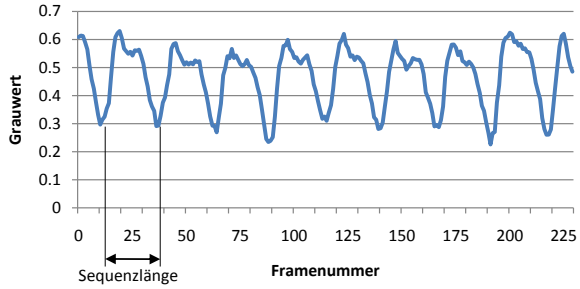


Abbildung 4: Grauwertverlauf über mehrere Frames

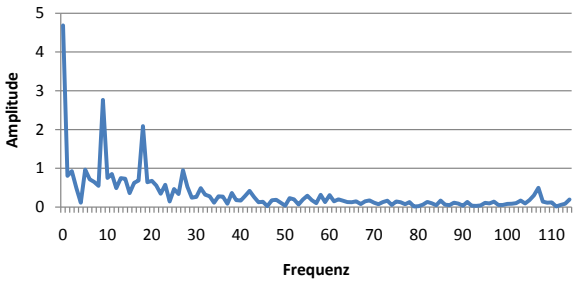


Abbildung 5: Frequenzspektrum des Grauwertverlaufs

angenommen werden, dass die zu überprüfende Sequenz auch als korrekt bezeichnet wird.

Zur Bestimmung der Sequenzlänge oder der Sequenzfrequenz wird während der Lernphase der durchschnittliche Grauwert an einer geeigneten Bildstelle gemessen. Diese Stelle muss sich dadurch auszeichnen, dass die Sequenzfrequenz optisch erkennbar ist (Abb. 4). Aus diesem zeitabhängigen Signal wird dann mittels FFT das Frequenzspektrum (Abb. 5) berechnet und daraus die dominante Frequenz abgelesen. Der erste Peak bei der Frequenz 0 beschreibt die Intensität des Grauwertsignals; der zweite Peak bei der Frequenz 9 zeigt die dominante Frequenz.

Geschwindigkeit und Güte

Wir beschränken uns hier auf ein paar wenige Resultate zum bereits mehrfach erwähnten Beispiel mit den Schraubdeckeln. Das Video besteht aus 42 Sequenzen mit durchschnittlich 25 Bildern der Grösse 640 x 240 Bildpunkten, wobei eine Sequenz das Aufschrauben genau eines Deckels zeigt. Im Video enthalten vier Sequenzen fehlerhafte Deckel, die sich durch ausgefranste Kanten von korrekten Deckeln unterscheiden (Abb. 6).

Für die Berechnung der charakterisierenden Funktion wählen wir eine Filterbereichsgrösse von 143 x 34 Bildpunkten. Damit erreichen wir auf unserem Testsystem eine Verarbeitungsgeschwindigkeit von 540 Bildern pro Sekunde und finden die vier falschen Deckel, ohne weitere Sequenzen als fehlerhaft zu bezeichnen.

Adaptive Parametrierung

Implementierungen von Bildverarbeitungsalgorithmen mit Echtzeitgarantien werden in der Video- und industriellen Bildverarbeitung schon heute eingesetzt. Diese Implementierungen sind jedoch oft plattformspezifisch und erfüllen entweder die Echtzeitbedingung dieser einen Plattform oder eben nicht. Diese duale Sichtweise der Einsatztauglichkeit einer Implementierung ist in der Bildverarbeitung meistens zu hart und kann oft umgangen werden, indem tiefere Bildauflösungen oder kleinere Ausschnitte prozessiert werden, ohne dass sich dabei die Resultate ändern. Das Erstellen von kleineren Bildauflösungen oder Ausschnitten wird daher als Pre-Processing-Schritt betrachtet und kann vom eigentlichen Algorithmus entkoppelt werden (Abb. 7a). Diese Entkopplung ist sehr wertvoll, weil dadurch bestehende, effiziente Bildverarbeitungsalgorithmen unverändert übernommen werden können. Was bei einer solchen Entkopplung jedoch oft verloren geht, ist die formale Spezifizierung der Abhängigkeit von der Laufzeit des Algorithmus. Das heisst, die Wahl der Parameter P1 des Pre-Processings (z.B. die gewünschte Bildauflösung oder die Ausschnittgrösse) ist nicht entkoppelt vom Algorithmus, sondern im Gegenteil, die Laufzeit des gewählten Algorithmus bestimmt indirekt die Parameter P1. Oft werden diese Parameter in meist aufwendigen Performanztests während der Entwicklungsphase ermittelt. Dadurch wird aber die Auswahl der Zielplattformen stark eingeschränkt, weil das Bildverarbeitungssystem seine Tauglichkeit nur auf dem Entwicklungsrechner gezeigt hat.

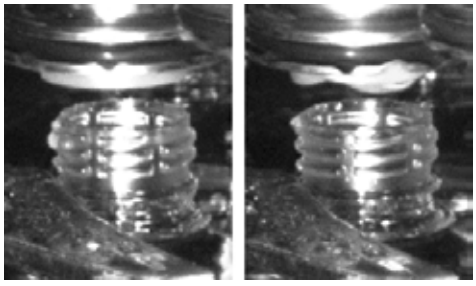


Abbildung 6: Beispiel eines guten (links) und eines fehlerhaften Deckels (rechts)

Anstatt die Parameter P1 manuell zu bestimmen, ist es oft sinnvoller, die passenden Werte dynamisch in einem Rückkopplungsprozess zur Laufzeit zu ermitteln (Abb. 7b). Diese Rückkopplung verschlingt natürlich wiederum Rechenzeit, welche der Hauptaufgabe abgeht. Daher wird in vielen zeitkritischen Anwendungen darauf verzichtet. In gewissen Ausnahmefällen kann jedoch eine dynamische Parametrisierung (z.B. maschinelles Lernen) eingesetzt werden, z.B. wenn eine vorgeschaltete Kalibrierungsphase zu Realbedingungen ausgeführt werden darf. So ist es möglich, die passenden Parameter P1 durch einen eingebauten Lernmechanismus zu ermitteln und für den Echtzeitbetrieb abzuspeichern. Bei Bedarf kann diese Kalibrierungsphase wiederholt werden.

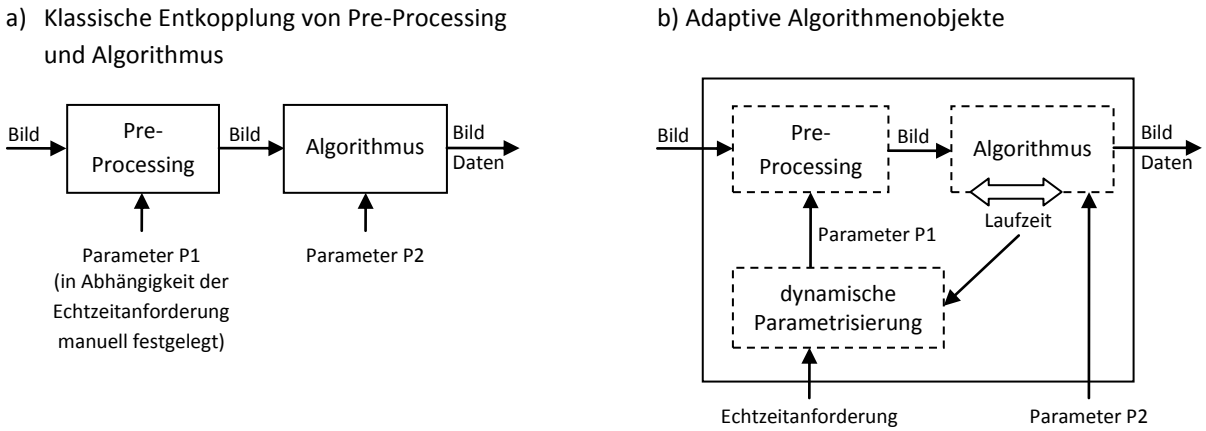


Abbildung 7: Ansätze zur Erfüllung von Echtzeitanforderungen in Algorithmen

Parametrisierung in Promon200

In Promon200 kann die komplette Bildverarbeitungskette an die Echtzeitanforderung, also an eine bestimmte Bildrate und eine maximale Verzögerungszeit, angepasst werden. Solange die gemessene und gemittelte Bildrate grösser oder gleich der gewünschten ist, läuft das System in einem stabilen Zustand. Sobald die gemittelte Bildrate jedoch unter den Sollwert sinkt, müssen geeignete Massnahmen getroffen werden, damit der stabile Zustand wieder hergestellt werden kann.

Welches die geeigneten Massnahmen sind, ist im Normalfall nicht a-priori klar. Daher unterstützt Promon200 drei softwaretechnische (Bildauflösung reduzieren, Videoframes überspringen, Filterregionen verkleinern) und zwei hardwaretechnische Möglichkeiten (Algorithmus auf der GPU ausführen, die Anzahl CPU-Kerne erhöhen). Oberstes Ziel bleibt dabei ständig, fehlerhafte Abläufe bei Aufrechthaltung der geforderten Bildrate zu erkennen. Auf die beiden hardwaretechnischen Massnahmen soll an dieser Stelle nicht eingegangen werden. Im Abschnitt über CUDA zeigen wir dann die spezifische Umsetzung von Bildverarbeitungsalgorithmen für die GPU.

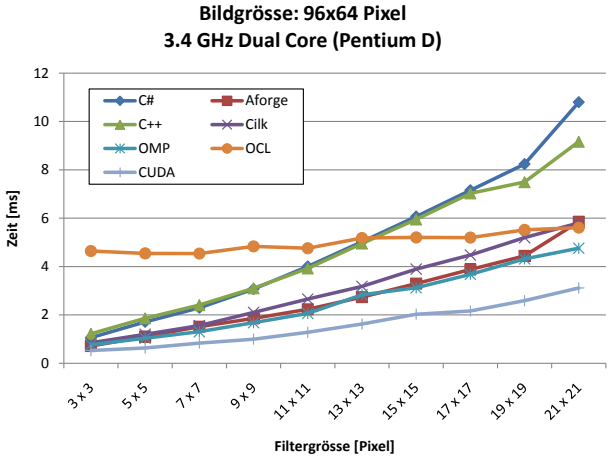


Abbildung 8: Performance eines Boxfilters mit verschiedenen Programmiersprachen und Parallelisierungsbibliotheken.

Dass alle drei softwaretechnischen Massnahmen mit einer offensichtlichen Daten- und Informationsreduktion einhergehen, darf nicht weiter überraschen, wenn man davon ausgeht, dass bei der Entwicklung der Algorithmen das Bestmögliche getan worden ist, um eine gute Effizienz zu erzielen. Entscheidend aber ist, ob die Datenreduktion wirklich zu einer Beeinträchtigung in einem realen Szenario führt, was dann der Fall wäre, wenn infolge der reduzierten Datenmenge nicht mehr alle fehlerhaften Abläufe erkennt werden könnten. Selbstverständlich ist es nicht möglich, alle Beeinträchtigungen infolge der Informationsreduktion im Vornherein auszuschliessen. Solange eine Datenreduktion aber nicht merkbar ist und zu keiner offensichtlichen Beeinträchtigung führt, wird sie vom Benutzer meist stillschweigend akzeptiert (so sind sich Benutzer einer Digitalkamera beispielsweise oft nicht bewusst, dass die Bildspeicherung im JPEG-Format eine Daten- und Informationsreduktion bewirkt).

Anstatt der bedarfsmässigen und automatischen Datenreduktion durch Promon200, könnte

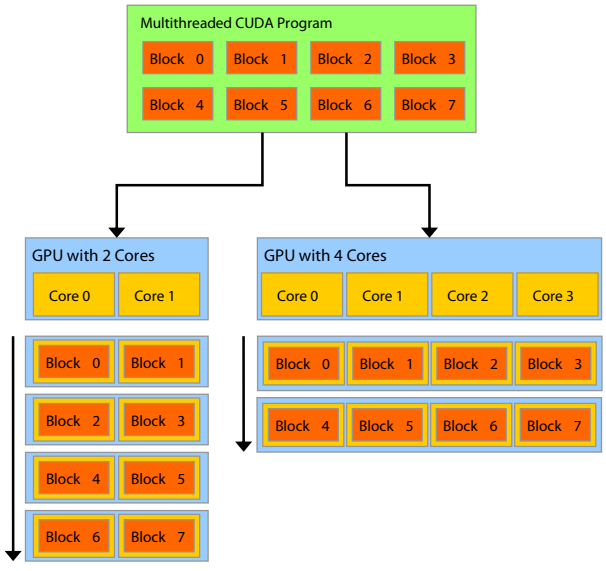


Abbildung 9: Aufteilung eines GPU-Programms in Blöcke (Quelle: [CUDA])

natürlich auch der Benutzer selber die Framerate der Hochgeschwindigkeitskamera reduzieren, eine kleinere Bildauflösung wählen oder den überwachten Bildausschnitt in der Grösse reduzieren. Für den Benutzer erleichtert sich jedoch die Arbeit wesentlich, wenn er alle Einstellungen gemäss seinen Erfahrungen machen kann und anschliessend das System die verschiedenen Datenreduktionsmassnahmen automatisch durchtestet und ihm die Optimierungsvorschläge unterbreitet, welche sich während der Test- oder Kalibrierungsphase von Promon200 bestens bewährt haben.

Bildverarbeitung auf der GPU

Der Grafikprozessor GPU (Graphics Processing Unit) ist primär für die Berechnung der Bildschirmausgabe zuständig. Seit etwa acht Jahren bieten die beiden grössten Grafikkartenhersteller Nvidia und AMD eigene APIs an, mit welchen sich auch andere Aufgaben auf der GPU ausführen lassen. Diese Art der Verwendung von GPUs nennt sich General Purpose Computation on Graphics Processing Units [GPGPU].

Die Leistungsfähigkeit von GPUs in der Bildverarbeitung lässt sich im Vergleich mit der parallelen und der sequentiellen Ausführung eines Algorithmus aufzeigen. In Abbildung 8 zeigen wir die gemittelte Ausführungsdauer eines einfachen Boxfilters mit gegebener Filtergrösse auf einem kleinen Bild der Grösse 96 mal 64 Pixel. Die beiden verwendeten Programmiersprachen (C# und C++) werden sich in sequentieller und in paralleler Ausführung mithilfe von Parallelisierungsbibliotheken (C#: Aforge[7]; C++: Cilk[8], OMP[9]) gegenübergestellt. Im Vergleich dazu zeigt die Grafik auch die Ausführungsdauer auf der GPU (OCL[10], CUDA[11]) inklusive der notwendigen Datentransferzeit. Interessant dabei ist vor allem der hohe Initialaufwand von OpenCL. Unterschiedlichste Performancetests mit verschiedenen GPU-APIs und verschiedenen Programmiersprachen haben gezeigt, dass die Verwendung von OpenCL unabhängig der Filtergrösse einen Initialaufwand von ca. 5 ms mit sich bringt, und somit die maximale Verarbeitungsrate auf der GPU auf ca. 200 Bilder pro Sekunde begrenzt. Infolge dieser Beschränkung verwenden wir die GPU mit einem herstellerspezifischen API, in unserem Fall dem CUDA API von Nvidia.

Ein Grund für den zusätzlichen Performanzgewinn infolge des GPU-Einsatzes liegt in der massiv parallelen Ausführung ohne Synchronisation, die gerade in der Bildverarbeitung anzutreffen ist. Während ein aktueller Prozessor von Intel (i7-980X) sechs Prozessorkerne hat, befinden sich auf dem aktuellen Topmodell von Nvidia (GeForce GTX 590) 1024 Kerne. Eine GTX 480, welche wir für die meisten Performancetests verwendet haben, hat 480 Kerne, eine Rechenleistung von 1345 GFLOPS und eine Speicherbandbreite von 177.4 GByte/s. Im Vergleich dazu hat ein Intel i7-980 vier Kerne, eine Rechenleistung von 80 GFLOPS und eine Speicherbandbreite von 4,8 GByte/s. Diese hohe Speicherbandbreite bei GPUs nützt in den meisten Fällen jedoch wenig, da die Videodaten zuerst vom CPU-RAM zur GPU transferiert werden müssen.

Programmieren mit CUDA

Die grosse Anzahl an Rechenkernen beeinflusst auch wesentlich die Programmierung der Algorithmen. Sie müssen einerseits hoch parallelisierbar sein und anderseits möglichst auf Synchronisation verzichten, damit die Rechenleistung einer Grafikkarte voll ausgenutzt werden kann. Da die GPU der *Single Program Multiple Data* Architektur (SPMD) folgt, kann lediglich ein einziges Programm auf der GPU in Ausführung sein. Das heisst, alle Prozessorkerne führen zwar das gleiche Programm aus, aber mithilfe der Thread-ID können dennoch individuelle Aufgaben pro Thread ausgeführt werden.

In der CUDA-Terminologie wird die in C geschriebene Thread-Funktion eines GPU-Programms als *Kernel*[12] bezeichnet. Die erforderliche Anzahl Ausführungen des Kernels wird beim Start als Parameter in Form von *BlockSize* und *GridSize* übergeben. Die beiden Parameter können ein-, zwei- oder dreidimensional gewählt werden, und bestimmen die Anzahl Kernelaufrufe pro Block und die Anzahl Blöcke. Die Blockgrösse ist bei aktuellen Grafikkarten auf 1024 begrenzt. Der Grund für diese Aufteilung in Blöcke liegt in der Skalierbarkeit. Da Blöcke unabhängig voneinander ausgeführt werden, wird eine GPU mit mehr Kernen das Programm automatisch in kürzerer Zeit ausführen als eine GPU mit weniger Kernen, ohne dass der Code angepasst werden muss (Abb. 9).

7 AForge.NET ist ein Open-Source-Framework für die Bereiche Maschinelles Sehen und Künstliche Intelligenz. Es ist in der .NET-Sprache C# geschrieben. http://www.aforgenet.com/

8 Intel® Cilk™ Plus ist eine Spracherweiterung zu C/C++, welche auf schnelle, einfache und zuverlässige Art erlaubt, die Mehrkernprozesse parallel zu benutzen. http://software.intel.com/en-us/articles/intel-cilk-plus/

9 OpenMP ist eine seit 1997 gemeinschaftlich von verschiedenen Hardware- und Compiler-Herstellern entwickelte Programmierschnittstelle. Der Standard dient zur Shared-Memory-Programmierung in C/C++/Fortran auf Multiprozessor-Computern. http://openmp.org/wp/

10 OpenCL ist eine Schnittstelle für uneinheitliche Parallelrechner, die z.B. mit Haupt-, Grafik- und/oder digitale Signalprozessoren ausgestattet sind, und mit der zugehörigen Programmiersprache „OpenCL C". http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf

11 Cuda API: http://www.nvidia.com/object/cuda_home_new.html

12 Wir verwenden hier den englischen Begriff Kernel in Abgrenzung zum deutschen Wort Kern, welches wir im Zusammenhang mit Prozessorkernen verwenden.

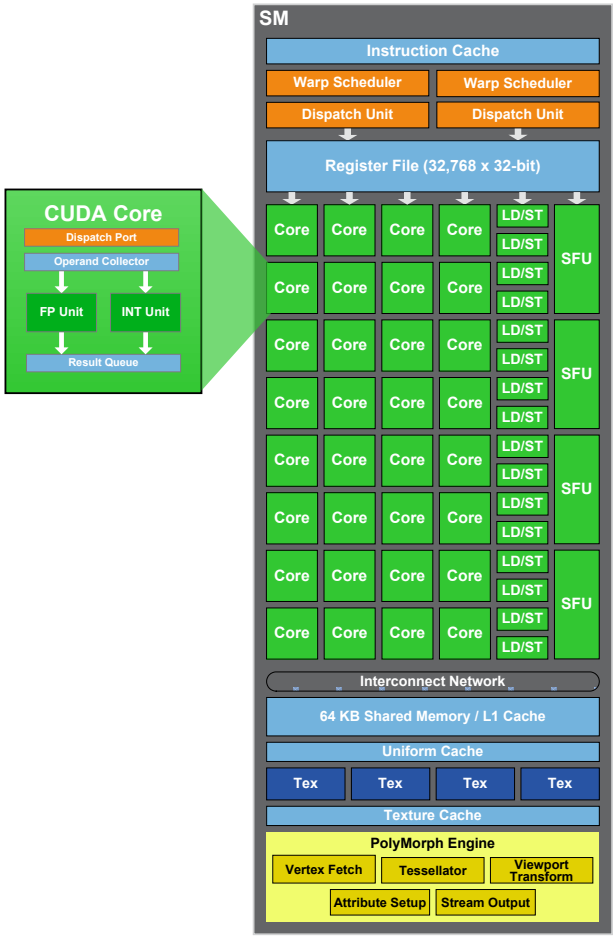


Abbildung 10: Aufbau eines Streaming Multiprocessors mit 32 Kernen. Eine GTX 480 hat 15 dieser SMs (Quelle: [GF100])

Innerhalb eines Blocks haben die Threads einen gemeinsamen, relativ kleinen (mehrere KByte), dafür aber schnellen Speicher (*shared memory*) zur Verfügung. Alle Threads haben zudem Zugriff auf einen gemeinsamen Speicher (*global memory*), der bei aktuellen Grafikkarten mehrere 100 MByte gross ist. Threads werden den sogenannten *Streaming Multiprocessors* (SM) zugewiesenund zwar immer in der Granularität eines Blocks. Die einzelnen Blöcke dürfen keine Abhängigkeiten untereinander haben, da deren Ausführungsreihenfolge nicht garantiert ist. Hingegen können Threads innerhalb eines Blocks an einer Barriere synchronisiert werden. Eine GTX 480 basiert auf der Fermi-Architektur [GF100] und hat 15 SMs mit je 32 Kernen (Abb. 10). Jedem SM können 1536 Threads zugewiesen werden. Das heisst eine optimale Aufteilung wäre zum Beispiel pro SM sechs Blöcke zu je 256 Threads.

Die Wahl der richtigen Blockgrösse kann die Gesamtperformanz eines GPU-Programms wesentlich beeinflussen. So ist aus der Tabelle 1 klar ersichtlich, dass sich die Berechnungsdauer des Skalarprodukts zweier Vektoren der Länge 2^{22} bis zu einem Faktor zehn unterscheiden kann, je nach gewählter Blockgrösse. Der Performancebericht von Nsight[13] in Abbildung 11 zeigt die Auswirkungen der Blockgrösse auf die Auslastung der GPU. Im linken Teil der Abbildung ist ersichtlich, dass bei einer Blockgrösse von 192 eine optimale Auslastung der SMs erreicht werden kann. Im rechten Teil der Abbildung ist die Auslastung der SMs in Abhängigkeit der Blockgrössen grafisch dargestellt. Zusätzlich zu diesen Angaben liefert Nsight genaue Zeitmessungen zu sämtlichen CUDA Funktionsaufrufen.

Der relevante Teil des Programms zur Berechnung des Skalarprodukts ist in Listing 1 ersichtlich. Es zeigt den Kernel für die GPU und die wichtigsten Teile des aufrufenden Hauptprogramms. Das Schlüsselwort *__global__* macht den GPU-Kernel für das Hauptprogramm sichtbar. In diesem feingranulierten Beispiel berechnet jeder Thread des Blocks genau eine Multiplikation zweier Vektorelemente und speichert das Resultat im gemeinsamen Speicherbereich *temp*. Mit dem Aufruf der Prozedur *__syncthreads()* wird sichergestellt, dass alle Threads eines Blocks an der Stelle ihre Multiplikation abgeschlossen haben, bevor der Thread mit der ID 0 die Summe in der lokalen Variable *sumPerBlock* bildet und diesen Wert mit der atomaren Funktion *atomicAdd()* zur globalen Variable *c* addiert. Diese atomare Addition ist notwendig, da mehrere Blöcke gleichzeitig ausgeführt werden können.

Blocksize	Allocation [µs]	Memcopy [µs]	Kernel [µs]	Total [µs]
1024	1496	3509	20126	25131
768	1507	3510	10690	15707
512	1482	3489	2606	7577
256	1496	3511	2052	7059
192	1485	3459	2037	6981
128	1483	3490	3009	7982

Tabelle 1: Zeitmessung am Beispiel des Skalarprodukts mit einer Vektorgrösse von 2^{22}

CUDA Tipps

Wie bereits erwähnt, sollten die typischen Eigenheiten einer GPU, nämlich die grosse Anzahl Rechenkerne, die relativ hohe Ausführungsgeschwindigkeit und der nicht unerhebliche Zeitaufwand beim Datentransfer zwischen Hauptspeicher und Grafikadapter, die Parallelisierung eines Algorithmus beeinflussen. Die hohe Anzahl Kerne spricht für eine recht feine Granularität, sofern diese nicht einen übermässigen Datenaustausch über die Thread-Grenzen hinweg bedingt. Vor allem die Wahl der Blöcke und deren Grösse haben einen grossen Einfluss auf die parallele Ausführungsgeschwindigkeit, weil der Datenaustausch innerhalb eines Blockes wesentlich

13 Nvidia Nsight: http://developer.nvidia.com/nvidia-parallel-nsight

```
#define TPB 512      // threads per block
#define N   TPB*8192 // number of vector elements

__global__ void dotproduct(float* a, float* b, float* c){
      __shared__ float temp[TPB];
      int index = threadIdx.x + blockIdx.x*blockDim.x;
      temp[threadIdx.x] = a[index]*b[index];
      __syncthreads();
      if (0 == threadIdx.x) {
            float sumPerBlock = 0;
            for(int i=0; i<TPB; ++i)
                  sumPerBlock += temp[i];
            atomicAdd(c, sumPerBlock);
      }
}

int main(void){
      // memory allocation and initialization
      ...

      // copy inputvectors to GPU
      cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
      cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);

      // kernel launch (<<< Gridsize, Blocksize>>>)
      dotproduct<<< N/TPB, TPB>>>(d_a, d_b, d_c);

      // copy result back from GPU
      cudaMemcpy(h_c, d_c, sizeof(float), cudaMemcpyDeviceToHost);
}
```

Listing 1: Skalarprodukt-Kernel und Teil des aufrufenden Hauptprogramms

schneller ist, als der Zugriff auf den globalen GPU-Speicher. Somit ist bei einer Dekomposition einer Problemstellung in Blöcke darauf zu achten, dass die Dekomposition entlang der natürlichen Datengrenzen erfolgt, z.B. entlang der Eingabe- oder Ausgabedaten. Generell ist darauf zu achten, dass der Speichertransfer zwischen der CPU und der GPU möglichst gering ist, da dieser ca. 10- bis 20-mal langsamer ist als der Zugriff auf das *global memory* der GPU. Wo immer möglich sollte jedoch das *shared memory* anstatt dem *global memory* verwendet werden, um von einer besseren Zugriffszeit profitieren zu können.

Unterschiedliche GPUs unterscheiden sich nicht nur in der Anzahl der Rechenkerne, der Grösse des Speichers und der Taktfrequenz, sondern haben auch unterschiedliche Befehlssätze. Diese Geräteeigenschaften lassen sich über das CUDA-API abfragen. Ebenso ist die Aufteilung der Threads in Blöcke unterschiedlich zu handhaben. Damit die Streaming Multiprocessors während der ganzen Ausführungszeit des GPU-Programms möglichst voll ausgelastet sind, müssen die Blöcke so dimensioniert werden, dass erstens pro SM mehrere Blöcke vorhanden sind, um bei Synchronisation und Speicherzugriff zwischen laufbereiten und wartenden Blöcken schnell wechseln zu können, dass zweitens die Anzahl Threads pro SM ein ganzzahliges Vielfaches der Blockgrösse ist und drittens, dass alle aktiven Blöcke pro SM nicht mehr shared memory und Register benötigen, als ein SM zur Verfügung stellt. Darüber hinaus gibt es noch weitere Bedingungen (z.B. ein Vielfaches der Warp-Size), die möglichst eingehalten werden sollen. Mithilfe des CUDA Occupancy Calculators [OCALC] lassen sich auf einfache Art plattformspezifisch gute Werte im Vorfeld ermitteln.

Nvidia bietet zu ihren CUDA-fähigen Grafikadaptern auch verschiedene Werkzeuge, z.B. ein Debugger und ein Performance-Analyzer. Für die Entwicklungsumgebung Visual Studio von Microsoft gibt es zudem ein Plug-In. Damit lässt sich

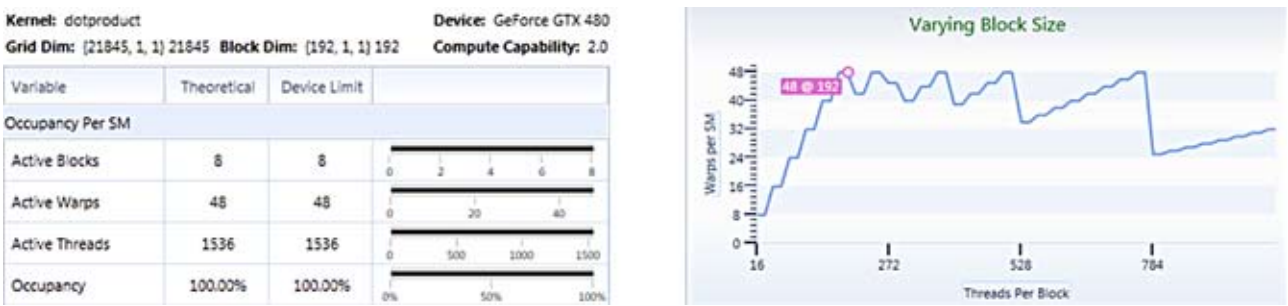


Abbildung 11: Auszug aus dem Performance Report von Nsight

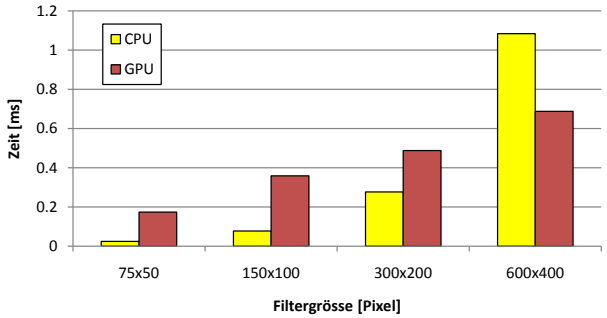


Abbildung 12: Geschwindigkeitsvergleich CPU/GPU des Durchschnittsgrauwertfilters

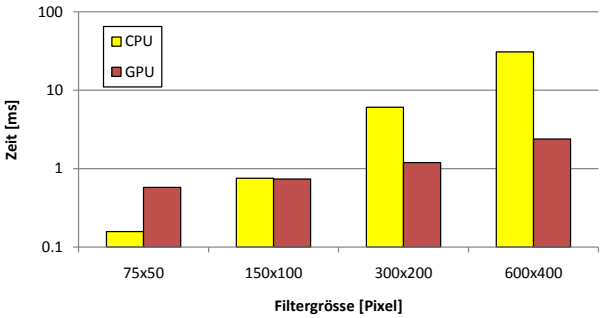


Abbildung 13: Geschwindigkeitsvergleich CPU/GPU des Templatematching-Filters

der Kernelcode debuggen, das heisst die Ausführung des Kernelcodes lässt sich anhalten und ein Wechsel zu einem beliebigen anderen Thread in einem anderen Block ist möglich. Voraussetzung dafür ist allerdings, dass neben einer CUDA-fähigen Grafikkarte noch eine zweite Grafikkarte zur Ansteuerung des Bildschirms vorhanden ist.

Promon200 und CUDA

Mehrere der in Promon200 verfügbaren Filter sind sowohl in einer parallelen C#- als auch in einer CUDA-Version vorhanden. Bei sehr einfachen Filtern mit linearem Aufwand, wie zum Beispiel der Bestimmung der durchschnittlichen Helligkeit, lohnt sich der Einsatz der CUDA-Variante auf unserem Testsystem erst ab einer Filterbereichsgrösse von 400 x 300 Bildpunkten (Abb. 12). Dies liegt daran, dass der Overhead, verursacht durch den zusätzlichen Datentransfer zwischen Hauptspeicher und Video-RAM, den Performanzgewinn bei kleineren Filterbereichen wieder zunichte macht. Bei einem rechenintensiveren Filter, wie beispielsweise dem Templatematching-Filter, lohnt sich die Ausführung auf der Grafikkarte bereits ab einer Filterbereichsgrösse von 150 x 100 Bildpunkten und ab einer Grösse von 300 x 200 Bildpunkten steht dann nur noch die CUDA-Variante zur Verfügung, weil die C#-Variante die Echtzeitforderung von 200 Bildern pro Sekunde nicht mehr einhalten kann (Abb. 13). An diesem Beispiel ist gut ersichtlich, dass in Abhängigkeit des Filtertyps und der Filterbereichsgrösse entschieden bzw. ausgetestet werden muss, welche von den zur Verfügung stehenden Varianten leistungsfähiger ist und somit zum Einsatz kommen soll.

Zusammenfassung und Ausblick

In umfangreichen Performancetests verschiedenster Filterimplementierungen auf der CPU und GPU haben wir die Leistungsfähigkeit unserer Implementierungen nachgewiesen. Für mehrere konkrete Anwendungsbeispiele hat Promon200 genau die fehlerhaften Sequenzen in den Abläufen gefunden. Die dabei geforderte Ablaufgeschwindigkeit von 200 Bildern pro Sekunde kann eingehalten und teilweise stark übertroffen werden. So lässt sich auf einem PC mit i7-Prozessor (3 GHz) und einer Nvidia GTX480 Grafikkarte der exemplarische Produktionsablauf einer Flaschenabfüllanlage mit einer Bildrate von bis zu 540 Frames pro Sekunde überwachen. In diesem Prozess werden Flaschen aussortiert, bei denen der Schraubdeckel fehlerhaft oder nicht korrekt verschraubt ist. Diese Aufgabe ist mit einem vollautomatischen Lern- und Testsystem umgesetzt worden, welches beliebige Anomalien in repetitiven Produktionsabläufen erkennt.

Referenzen

[CUDA] Nvidia CUDA C Programming Guide: http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf

[GF100] Nvidia GF100 White Paper: http://www.nvidia.co.uk/object/IO_89569.html

[GPGPU] General-Purpose Computation on Graphics Hardware: http://gpgpu.org/about

[OCALC] http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls

[PRO] PROMON Streaming, AOS Technologies AG: http://www.aostechnologies.com/process-monitoring/products-process-monitoring/promon-streaming/

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