Ein komponentenbasiertes Navigationssystem mit Client-Server Architektur

Tobias Dahinden
dahinden@karto.baug.ethz.ch
Institute of Cartography
ETH Zurich
CH-8093 Zurich

14th July 2005

Abstract

In diesem Artikel wird ein Client-Server System beschrieben. Das System kann Positionsdaten aus einem Empfänger lesen. Diese Daten werden in ein 2D-Koordinatensystem transformiert. Anschliessend werden diese Koordinaten in eine XML-Zeichenkette umgewandelt. Die XML-Zeichenkette wird dem Client per TCP (HTTP/UDP) gesandt.
Der Vorteil von diesem System ist seine Modularität. Es kann sehr schnell an neue Hardware und Empfängertechnologien angepasst werden. Die Daten werden in einem systemunabhänigen Format (XML) via Internet verschickt. Als Client kann man einen herkömmlichen Webbrowser sowie einen speziellen Navigationsclient mit einem SVG-Interface verwenden.

1 Einführung

Navigationssysteme sind heute Teil vom täglichen Leben. Man investiert viel Geld in Systeme, die die Fahrzeugflut überwachen. Die ETH entwickelte ein System, dass es ermöglicht Daten von einem beliebigen Empfängergerät zu lesen und diese Daten per Internet einem Client schickt. Die Entwicklung dieses Systems war Teil von einem EU-Projekt.

1.1 Projektziel

Das Projekt hat verschiedene Ziele:

- Wir wollten einen Server entwickeln, der auf ≪Personal digital assistants (PDA)≫läuft. Der Server muss Daten von einem Empfänger (wie z.B. GNSS, INS, DMC) oder einem ASCII-Dokument lesen können. Der Server muss auf HTTP und UDP Anfragen antworten können. Der Server soll als Antwort dem Client eine XML-Zeichenkette senden. Es sollte daher möglich sein eine Anfrage mit einem herkömmlichen Webbrowser zu schicken.
- Wir wollten einen Client entwickeln, der sowohl Position als auch Metadaten (z.B. die Anzahl Satelliten bei GPS-Empfängern) anzeigen kann.
- Wir wollten einen Client, der die Position auf einer Karte anzeigen kann.
- Wir wollten den Client für die Navigation (Zielfindung) brauchen können. Es sollte möglich sein, Zielkoordinaten einzugeben und der Client soll die Richtung und Distanz zu diesem Ziel anzeigen können.
- Das ganze System sollte abstrakt programmiert werden. D.h. wir wollten uns nicht auf ein bestimmtes Navigationssystem beschränken oder eines bevorzugen.
- Für Testzwecke implementierten wir die Software für einen Sharp Zaurus, einen Dell Laptop und GPS.

2 Die Komponenten des Servers

Die Hauptkomponente vom Server ist ein Interface mit dem Namen ≪ServerReceiver≫. Diese Klasse ermöglicht es HTTP und UDP anfragen zu beantworten. Um die Antwort zu generieren muss diese Klasse einen sogenannten ≪Inputstream≫benutzen. Um das zu machen benutzt diese Klasse eine andere Klasse namens ≪InputStream≫als Interface um den Seriellen Port anzusprechen. Dieser Schritt braucht leider eine geräteabhängige Routine. Mit dieser Klasse können die Daten vom Empfänger an die Hauptklasse weitergeleitet werden.

Anschliessen muss die Hauptklasse die Daten interpretieren. Dazu wird die Klasse ≪Protocol≫und ≪Message≫benutzt. Das Protokoll muss in Abhängigkeit vom Empfänger benutzt werden. GPS braucht ein Protokoll namens NMEA. Wir implementierten die ≪Message≫-Klasse so, dass die Koordinaten des empfängerspezifischen System (z.B. WGS84 für GPS) in ein anderes Koordinatensystem überführt werden können. Der Vorteil von diesem System ist, dass der Client nicht wissen muss, welches System ein Empfänger benutzt. Der Nachteil liegt jedoch darin, dass man nur Koordinaten in System erhalten kann, die beim Server implementiert sind. Der Client muss dem Server sagen, welches Koordinatensystem er benutzen möchte.

Soweit möglich wurde der Server in Java programmiert. Jede Java Virtual Machine (JVM) die Java 1.1 oder Personal Java unterstützt kann den Server starten. Leider musste aber der Zugriff auf die Serielle Schnittstelle Geräteabhängig programmiert werden, wir benutzen dazu C. Dieser Teil muss für jedes Gerät das unterstützt werden soll, kompiliert werden. Das ist ein grosser Mangel an diesem System.

Abbildung 1 zeigt die Klassen des Servers im Detail. Die Klassen sind gemeinsam in einem JAR-Archiv gespeichert. Es ist möglich, jede einzelne Klasse zu ersetzten und das System so zu erweitern.
.

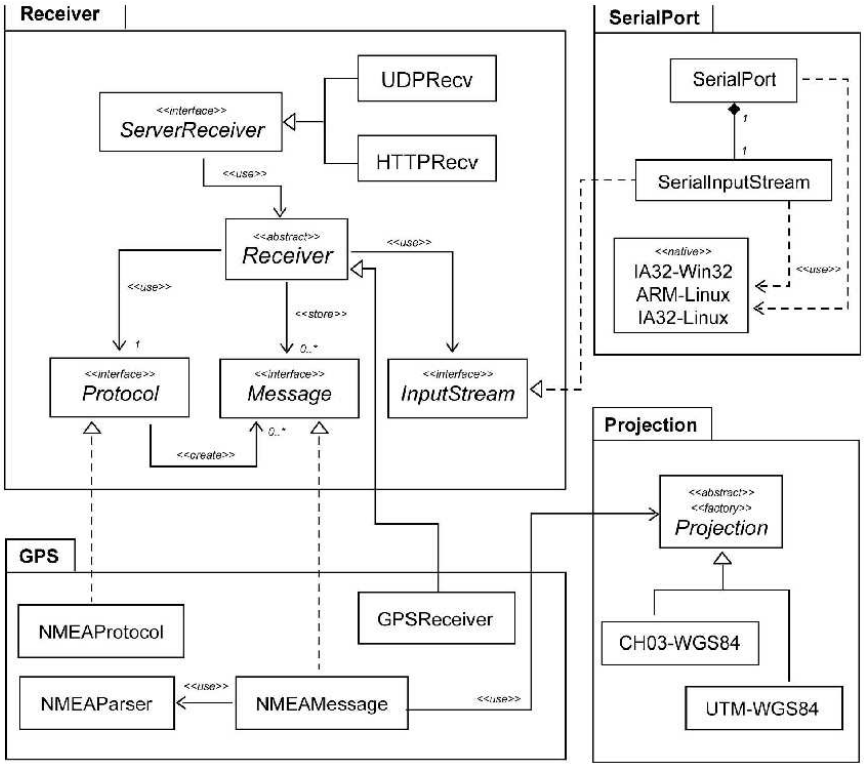

Figure 1: Die UML-Representierung der Klassen des Servers.

3 Die verschiedenen Clients

3.1 Web browser

Da der Server HTTP anfragen beantworten kann, ist es möglich einen herkömmlichen Webbrowser als Client zu benutzen. Die Adresse für einen Anfrage lautet:

```
http:///MyServer:MyServerPort/fix?projType=MyProjection
```

dabei steht MyServer für die IP-Adresse des Servers. MyServerPort wird beim Starten des Servers festgelegt. (Als standard Port brauchen wir Port 4444. Normalerweise wird dieser Port von Programmen wie ”kbr524” oder ”nv-video” benutzt. Sollten diese Programme serverseitig verwendet werden, muss der Port des Servers geändert werden.) MyProjection ist das Projektionssystem in dem die Daten geliefert werden sollen. Wobei eine entsprechende Projektion auf dem Server implementiert sein muss.

Abbildung 2 zeigt eine Anfrage via Webbrowser. Die Antwort ist XML, die gesamte Zeichenkette wird angezeigt. Falls der Browser aber XML interpretieren kann sieht die Antwort wie in Abbildung 3 aus. Ältere Browser kennen in der Regel XML nicht. Neuere Browser haben meist eine Option um die Interpretation zuzulassen oder nicht.

Die XML-Zeichenkette ist unabhängig vom Empfänger und Projektionssystem. Zudem kann sie verschiedene zusätzliche Informationen betreffend Empfänger und Datenqualität enthalten.

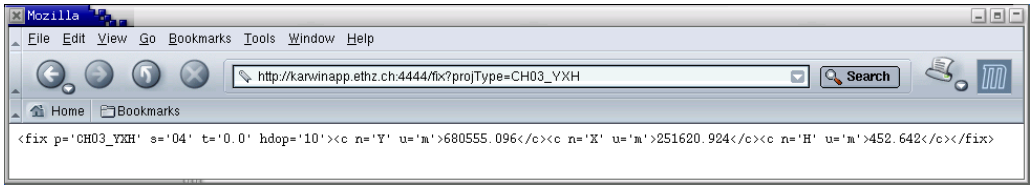

Figure 2: XML in einem Webbrowser ohne XML Interpretierung.

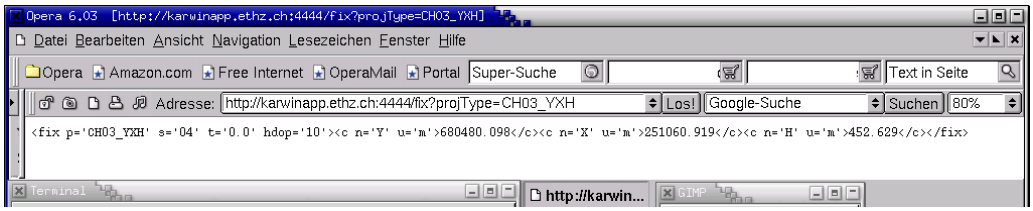

Figure 3: XML in einem Webbrowser mit Interpretierung.

3.2 Der Democlient

Es ist nicht immer interessant XML direkt zu lesen. Deshalb implementierten wir einen Java-Client der in konstaten Zeitabständen Anfragen an den Server sendet. Dieser Client interpretiert die XML-Zeichenkette und zeigt einiges des Inhaltes. Der Client ist vorallem für Testzwecke interessant.

Wir programmierten den Client so, dass er mit JVM 1.1 oder Personal Java läuft. Die Portnummer, eine Nachricht, eine Projektion und die IP Adresse des Servers können in einer Graphischen Benutzerschnittstelle (GUI) eingestellt werden. Abbildung 4 zeigt den Client vor dem Verbindungsaufbau. Abbildung 7 zeigt, wie er Daten empfängt.

3.3 Der Navigationsclient

Koordinaten in numerischer Form, wie z.B. im Democlient, sind nicht immer leicht verständlich und nützlich um eine bestimmte Position zu finden. Deshalb haben wir einen Client programmiert, der eine Position auf einer Karte anzeigen kann. Auch dieser Client sollte Geräteunabhängig sein.

3.3.1 SVG als Format für den Viewer

Um eine Position auf einer (Bildschirm-)Karte zu zeigen, muss die Karte irgendwie digital gespeichert und in einem sogenannten Viewer gezeigt werden. Für das Speichern und Anzeigen hatten wir folgende Anforderungen:

- Die Karten sollten auf dem Client gespeichert werden können und dürfen nur minimal Speicherplatz beanspruchen.
- Die Karten sollten in einem offenen Format gespeichert werden, was bedeutet, dass die Dokumentation des Formates frei zugänglich sein soll.
- Die Karten sollten als Kacheln gespeichert werden. Dabei müssen die Kacheln nicht a priori nach Norden ausgerichtet sein.

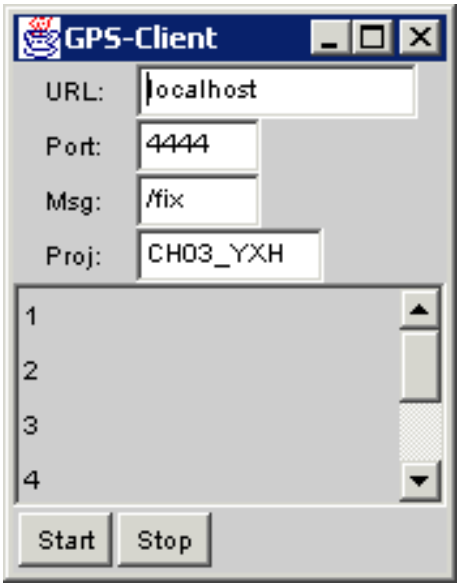

Figure 4: Der Democlient ohne Verbindung zum Server.

- XML muss interpretiert und vom Viewer dargestellt werden können. Die Darstellung soll als kleiner Kreis in der Karte erfolgen.

Punkt eins impliziert, dass die Karten in einem Vektorformat gespeichert werden sollen. Es gibt jedoch de facto nur zwei Vektorformate im Web: ein proprietäres von Macromedia namens Flash und der W3C standard SVG. Flash liegt ausserhalb unserer Vorstellungen, weil es nicht offen ist. SVG hat zudem den Vorteil, dass es ein XML-Dialekt ist und daher XML direkt integrieren kann. Weiter erlaubt SVG Rastergraphiken wie PNG, JPEG und GIF zu integrieren. [wwwSVG]

3.3.2 Ein Personal Java SVG-Viewer

Als wir dieses Programm entwickelten gab es noch keinen Opensource SVG-Viewer, der auf JVM 1.1 bzw. Personal Java läuft. Deshalb programmierten wir unseren eigen Viewer. Sonst wäre es nicht möglich gewesen alle PDAs als Clients zu unterstützen. (Inzwischen gibt es einen zweiten Opensource SVG-Viewer für Personal Java, der an der Universität Minho [wwwVie] entwickelt wird.)

Der Viewer unterstützt nicht die volle SVG Spezifikation. Er ist für unsere Ansprüche designed. Wenigstens kann er ”basic shapes” (Definition siehe [wwwSpe]), ”svg”, ”symbol”, ”use”, ”path”, ”text”, ”image” (inkl. base64) und CSS styles interpretieren. Für das Zoomen und Verschieben haben wir einige Funktionen von ≪awt≫(eine Java-Graphik-Bibiliothek) benutzt. Details zum Viewer finden sich übrigens auf [wwwGPS]. Auf der selben Webseite can der Viewer auch als Java Applet getestet werden.

3.3.3 Kombination von Viewer und Client

Für dieses Projekt kombinierten wir den SVG-Viewer mit dem Democlient. Alle Funktionen sind von JVM 1.1 bzw. Personal Java unterstützt. Entsprechend dem Democlient können alle Parameter mit einem GUI gesetzt werden.

Um die Koordinaten in die Karten zu integrieren, mussten wir die Karten georeferenzieren. Die Referenzierung geschieht mit sechs Parametern. Da verschiedene Karten das selbe Gebiet zeigen können, machten wir noch ein Popdown-Menu um die Karte auszuwählen.

Zusätzlich programmierten wir eine Möglichkeit um einzelne oder mehrere Punkte zu speichern. Diese Option kann gebraucht werden um Daten zu erfassen.

3.3.4 Verschiedene Navigationsmöglichkeiten

Eine Karte ist nicht immer die optimale Lösung für die Navigation. Deshalb programmierten wir eine andere Art um zu Navigieren: Man kann eine Zielposition eingeben, der Client berechnet dann die Distanz und die Richtung zu diesem Ziel. Die Richtung wird dabei immer relativ zu Norden angezeigt (Abb. 5). Für den Fall, dass man die Nordrichtung nicht kennt, kann man die Richtung relativ zum aktuellen Sonnenstand berechnen lassen (Abb. 6). Diese Funktion ist jedoch nur bei guter Witterung nützlich. Eine verlässlichere Methode wäre ein zusätzlicher einsatz von einem digitalen Kompass (DMC).

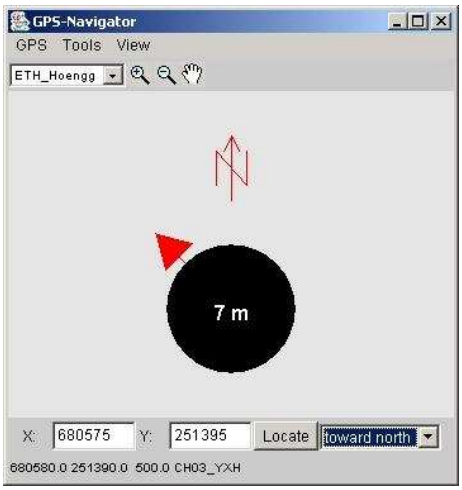

Figure 5: Der Navigationsclient mit Nordierung.

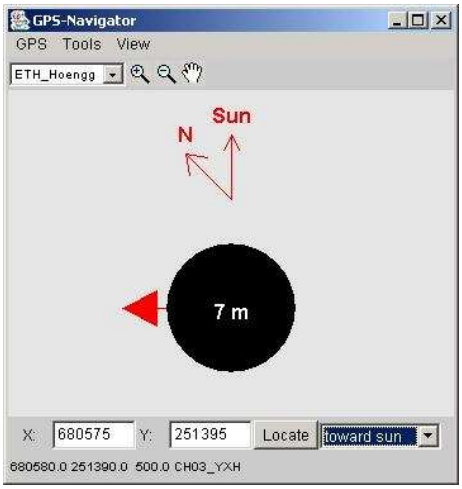

Figure 6: Der Navigationsclient nach der Sonne ausgerichtet.

4 Testimplementierung

Client und Server wurden abstrakt programmiert. Danach wollten wir testen, ob diese Implementierung auch funktioniert. Wir machten folgende Versuche:

- GPS Empfänger
- Sharp Zaurus PDA (Linux, Arm POSIX Processor) als Server und Client
- Dell laptop (Windows 2000, Intel M Processor) als Server und Client
- Linux workstation als Server (Eingabe jedoch nur mit ASCII-Dokument) und als Client
- Windows 2000 workstation als Server (Eingabe jedoch nur mit ASCII-Dokument) und als Client
- CH03 Projektion
- UTM Projektion
- 16 Kacheln einer Karte (GIF in SVG) der Region Zürich-Winterthur (Schweiz) 1:50’000 Projektion CH03
- 1 Kachel einer Karte (GIF in SVG) der Nordostschweiz 1:200’000 ProjeKtion CH03

Die Tests waren erfolgreich. Eine Schwierigkeit war wie erwartet der Teil des Servers, der in C programmiert ist. Diesen Programmteil mussten wir für alle Prozessoren kompiliert werden. Für den PDA mussten wir zudem Cross-compile-optionen anwenden, was nicht ganz einfach war.

Einige Probleme ergab es noch, wenn wir eine Linux workstation als Server benutzten. Auf Windows lief der Server nicht, falls er auf einer Remote-Disk gespeichert wurde. Sowohl Server als auch Client

laufen auf PDAs. Die Grösse von Server und Client sind weniger als 20 KB. Leider brauchen die Karten viel Speicher (3 MB) weil sie als Rasterbild vorliegen.

Abbildung 7 zeigt einen Screenshot vom laufenden Democlient. Abbildungen 8 und 9 zeigen den Navigationsclient.

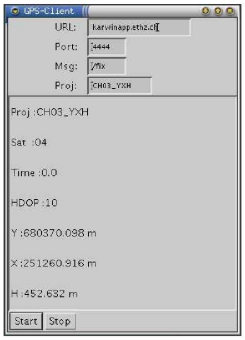

Figure 7: Der Democlient mit Serververbindung.

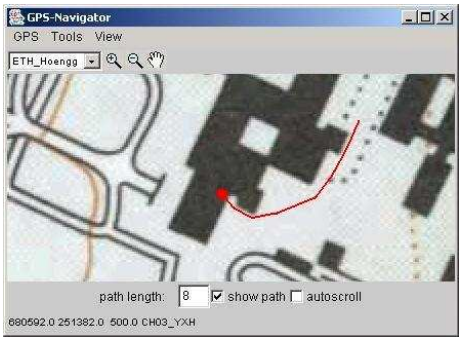

Figure 8: Der Navigationsclient mit einer Karte und dem IKA-SVG-Viewer.

5 Schlussfolgerungen, Limiten, Verbesserungen

Die Testimplementierung zeigte, dass das System funktionstüchtig ist. Weil das System abstrakt programmiert ist, kann es leicht an neue Hardware und neue Empfänger angepasst werden. Ein Problem ist jedoch, wie ausgeführt, der Programmteil in C.

Weitere Entwicklungsschritte könnten sein:

- Mehrere Kacheln anzeigen lassen. (Im Moment wird nur die aktuelle Kachel gezeigt.)
- Die Karten auf einem Mapserver speichern. (Im Moment müssen die Karten lokal gespeichert sein.)
- Implementieren von Filtern (z.B. Kalman) und weiteren Datenkorrekturalgorithmen.
- Mehrere Server von einem Client aus anzeigen lassen.
- Tests mit weiteren Empfängern (GLONASS, IMS, DMC). Die ETH Lausanne [wwwEPF] entwickelte ein Personennavigationssystem, welches sehr interessant für eine Testimplementation sein dürfte.

References

[wwwEPF] http://topo.epfl.ch/ Stand: 30.06.03

[wwwGPS] http://www.karto.ethz.ch/td/gps/ Stand: 30.07.03

[wwwSpe] http://www.w3.org/TR/SVG11/ Stand: 14.01.03

[wwwSVG] http://www.w3.org/Graphics/SVG/Overview.htm8 Stand: 06.08.03

[wwwVie] http://opensvgviewer.sourceforge.net/home.php Stand: 06.08.03

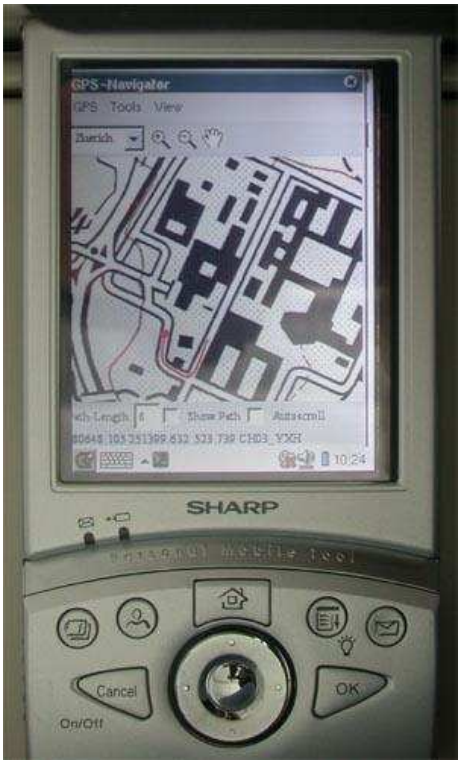

Figure 9: Server und Client auf einem Sharp Zaurus PDA (mit Embedded-Linux).