



Fortgeschrittenen-Praktikum
Mobile Navigationssysteme

Christoph Wellner, Christian Hahn, Michael Schellenbach

2. Februar 2004

Inhaltsverzeichnis

1	Einleitung	7
2	Installation von SuSE Linux 7.3 auf Xybernaut MA IV	9
2.1	Einleitung	9
2.2	Vorbereitungen	10
2.3	Installation	10
2.4	Konfiguration	10
2.4.1	Maus	10
2.4.2	XServer	11
2.5	Sound	12
2.6	Wireless Local Area Network (WLAN)	12
3	Aufbereitung des Kartenmaterials	13
3.1	Beschreibung des Programms	15
3.2	Funktionsweise	16
4	Erweiterung der graphischen Umgebung von ARREAL	19
4.1	Einleitung	19
4.2	Veränderungen für die Navigation im Innenbereich	20
4.2.1	Darstellung des Innenbereichs	20
4.2.2	Beschreibung der Lokalisierung im Innenbereich	20
4.2.3	Die neue Klasse IRDevice	20
4.2.4	Positionierung bei unterschiedlichem Infrarotempfang	21
4.2.5	Wechsel zwischen Innen und Außen	22

4.3	Weitere Änderungen der graphischen Umgebung	22
4.3.1	Kalibrierungsmodus	22
4.3.2	Zusatzfunktionen	24
5	Partner-Radar und Kommunikation mittels Text-Nachrichten	27
5.1	Einleitung	27
5.2	Architektur	27
5.3	Übersicht über die Objekte und Dateien	28
5.4	Konfigurations-Dateien	29
5.5	Network - Objekt	30
5.6	ServerObjectUDP	30
5.7	ServerObjectTCP	30
5.8	NetworkObject	30
5.9	Starten der Netzwerkunterstützung	31
6	Chat-Unterstützung	35
6.1	Informationsfenster	35
6.1.1	Angemeldete Benutzer	35
6.1.2	Nachrichteneingabe	36
6.1.3	Empfangene und gesendete Nachrichten	36
6.2	Senden einer Nachricht über das Netzwerk	37
7	Wichtige Hinweise zum neuen ARREAL-System	39
7.1	Lauffähige Versionen	39
7.2	Aufgetretene Fehler bei anderen Versionen	40
8	Doxygen Klassen-Dokumentation	41
8.1	Network Klassenreferenz	41
8.1.1	Ausführliche Beschreibung	42
8.1.2	Beschreibung der Konstruktoren und Destruktoren	42
8.1.3	Dokumentation der Elementfunktionen	42
8.1.4	Dokumentation der Datenelemente	44

8.2	NetworkUser Klassenreferenz	44
8.2.1	Ausführliche Beschreibung	45
8.2.2	Beschreibung der Konstruktoren und Destruktoren	45
8.2.3	Dokumentation der Elementfunktionen	45
8.3	NetworkUserIterator Klassenreferenz	45
8.3.1	Ausführliche Beschreibung	46
8.3.2	Beschreibung der Konstruktoren und Destruktoren	46
8.3.3	Dokumentation der Elementfunktionen	46
8.4	ServerTCP Klassenreferenz	46
8.4.1	Ausführliche Beschreibung	47
8.4.2	Beschreibung der Konstruktoren und Destruktoren	47
8.4.3	Dokumentation der Elementfunktionen	47
8.4.4	Dokumentation der Datenelemente	48
8.5	ServerUDP Klassenreferenz	48
8.5.1	Ausführliche Beschreibung	48
8.5.2	Beschreibung der Konstruktoren und Destruktoren	48
8.5.3	Dokumentation der Elementfunktionen	49
8.6	Parser Class Reference	49
8.6.1	Detailed Description	50

Kapitel 1

Einleitung

Das ARREAL System [2] ist ein Fußgängernavigationssystem das es den Benutzern bzw. dem Anwender ermöglicht, seinen Standort auf einer Karte zu visualisieren und Informationen zu Gebäuden abzurufen. Anfangs existierte Kartenmaterial über den Campus der Universität des Saarlandes. Später wurde weiteres Kartenmaterial, wie z.B. über das CeBit Gelände in Hannover hinzugefügt.

ARREAL ist ebenso in der Lage einen Benutzer ähnlich einem Autonavigationssystem mittels Richtungspfeilen und Sprach-Ausgaben zu einem angegebenen Ziel zu leiten bzw. zu führen. Die Lokalisierung erfolgt per GPS (Global Positioning System) Empfang, die Karte über das Gebiet indem sich der Benutzer befindet, wird über ein Clifton-Display in einer Spezialbrille bzw. auf ein Panel angezeigt. Dieses System bietet Anwendungsmöglichkeiten als Städteführer für Touristen oder als Orientierungshilfe in großen Gebäuden wie z.B. einem Flughafen.

Bislang wurde dieses Navigationssystem auf einem Sony Vaio Notebook, das in einem Rucksack auf dem Rücken getragen wird, eingesetzt. Unsere Aufgabe war es nun im ersten Teil unserer Arbeit ARREAL auf Xybernaut Hardware (MA IV) zu portieren. Diese Hardware besteht aus einem 900g leichten Pentium-PC, der am Gürtel getragen wird. Das dazugehörige Headset bietet einen VGA Monitor (640x480 Pixel) zur Visualisierung und ein Mikrofon zur Sprachsteuerung. Dazu war zuerst die Installation eines Betriebssystems nötig, welche wir im zweiten Kapitel näher erläutern werden.

Im zweiten Teil wird erwähnt wie das System für den Einsatz bei der CeBit aufgerüstet wurde, wobei vor allem die Aufbereitung des Kartenmaterials des CeBit Geländes im Mittelpunkt stand.

Im dritten Teil wird die Erweiterung des Navigationssystem um eine Netzwerk-Umgebung beschrieben, welche es den Benutzern ermöglicht, kleine Text-Nachrichten auszutauschen, sowie die Positionenangaben der verschiedenen Benutzern auf dem eigenen Display zu verfolgen.

Kapitel 2

Installation von SuSE Linux 7.3 auf Xybernaut MA IV

2.1 Einleitung

In diesem Teil unserer Arbeit beschreiben wir, wie man SuSE-Linux [5] auf einem Xybernaut MA IV installiert und konfiguriert, damit ein funktionstüchtiges Verhalten erreicht wird. Wir verwenden die Distribution 7.3, aber jede andere Linux Distribution ist für unseren Zweck auch ausreichend. Als Ausgangspunkt diente das Linux-MA4-HowTo von Torsten Bergander, der aber im Gegensatz zu unserem Vorhaben eine WinXX/Linux-Installation beschreibt. Wir beschränken uns auf eine reine Linux-Installation. Dieses HowTo ist leider nicht mehr verfügbar, da Xybernaut nun den MA IV mit vor-installiertem Linux ausliefert [8]



Abbildung 2.1: Xybernaut samt Headset

2.2 Vorbereitungen

Die Firma Xybernavt bietet passend zu den MA IV zwei mögliche Arten eines Displays an, zum einen wäre dies ein Head-Mounted-Display (HMD) und zum anderen ein Flatpanel, das am Arm getragen wird (siehe [4] für weitere Hinweise).

Zum einfachen Arbeiten haben wir an den Port-Replikator noch einen externen Monitor und ein Keyboard angeschlossen. Die am Replikator angeschlossene PS/2-Maus funktionierte nicht, was aber eher auf eine defekte Maus zurückzuführen ist. Aus diesem Grund mußten wir uns mit der eingebauten und auch funktionierenden Maus, deren Umgang anfangs gewöhnungsbedürftig war, begnügen. Als weiteres Hilfsmittel hatten wir eine PCMCIA-Netzwerkkarte zur Verfügung die uns dahingehend half, dass wir die Möglichkeit hatten Linux über das Netzwerk zu installieren, da wir ohne CD ROM-Laufwerk auskommen mußten.

2.3 Installation

Die eigentliche Installation bereitet keine weiteren Probleme, weshalb wir uns auf das Nötigste beschränken, da der Installationvorgang bei jeder Distribution und im speziellen bei SuSE Linux sehr genau im Handbuch [11] beschrieben wird. Da es sich bei dem MA IV um einen richtigen PC handelt, dürften dabei auch keine Probleme entstehen. Wir wählten eine Standard- Installation, da man bei SuSE über YAST einfach fehlende Pakete im Nachhinein noch einspielen kann, und die Festplatte für eine Installation aller Pakete als zu klein angesehen wurde.

2.4 Konfiguration

Zuerst beschreiben wir das Konfigurieren der Maus und des XServer, jeweils für das HMD und das Flatpanel. Danach die Sound-Konfiguration und zum Schluß alles, was sonst noch für unsere Zwecke benötigt wird, wie z.B. eine Wireless Local Area Network Karte (WLAN), die den Einsatz des Netzwerks, dass im dritten Teil näher erklärt wird, ermöglicht.

2.4.1 Maus

Die Maus (und später auch der XServer) wurden über XF86Setup konfiguriert. Für die einbaute Maus wählt man als Protokol "PS/2" und als Anschluß "/dev/psaux". Bei den restlichen Einstellungen kann man es bei den Default- Werten belassen. Durch Druck auf "Apply" sollte dann der Maus-Zeiger zu bewegen sein.

2.4.2 XServer

2.4.2.1 Head-Mounted-Display - (HMD)

Der MA IV besitzt eine NeoMagic-Grafikkarte. Alternativ kann man aber auch den SVGA-Server verwenden. Auf jeden Fall sollte man eine Auflösung von 640x480 wählen und eine Farbtiefe von 16bpp. RAMdac und Clock sollte man automatisch bestimmen lassen. Als Frequenzen für das Display wählt man 30-80 Hz vertikal und 60-100 Hz horizontal. Alle anderen Einstellungen kann man nach belieben verändern.

2.4.2.2 Flatpanel-Display - (FPD)

Die Installation des Flatpanel-Displays verlangt einiges an Einstellungen und zusätzlichen Treibern.

Zuerst muß man sich den Linux-Treiber für das FPD von [4] herunterladen. Ist dieser entpackt, muß man das Makefile an die Installation anpassen, da der Treiber für eine Redhat-Installation konzipiert ist bzw. wurde. Zuerst muß man den Pfad für die include-Dateien anpassen: Dazu ändert man jeden Eintrag in der Datei, der ähnlich wie "i386-redhat-linux/ egc-2.95" lautet, ab (für SuSE 7.3 z.B.: "i486-suse-linux/2.95.3"). Anschließend muß man man noch die Pfade zu den X11-Header-Files anpassen. Am einfachsten ist es jedoch, entsprechende Links zu setzen. Dann kann man den Treiber mit "make" und "make install" kompilieren und installieren. Nachdem der Treiber kompiliert ist, muß man die Datei "/etc/XF86Config" anpassen. Hier fügt man folgende Zeilen ein:

```
Section "Module"
    Load "xf86MobAssist.so"
EndSection
```

oder man ändert die Load-Section entsprechend ab. Dann fügt man noch folgende Zeilen an:

```
Section "XInput"
    SubSection "MobileAssit"
        Port "/dev/ttyS0"
        DeviceName "MobAssist0"
        ClickDelay 0
        AlwaysCore
    EndSubSection
EndSection
```

Weitere Informationen hierzu findet man unter [4]. Die Installation des XServers

verläuft analog zum HMD. Nach einem Neustart des XServer sollte nun das FPD mit allen Funktionen korrekt arbeiten.

2.5 Sound

Als Soundtreiber verwenden wir den ALSA-Treiber (Advanced Linux Sound Arcitecture). Allerdings ist das Kernelmodul es1968 in der von uns verwendeten Distribution fehlerhaft. Um diesen Fehler zu beheben, muß man zuerst das Skript `alsa-fixes.sh` von [6] herunterladen. Nachdem man den Patch eingespielt hat, kann man die Soundconfiguration über `“alsaconf”` starten. Falls die Sound-Konfiguration schon vorgenommen wurde, stoppt man zuerst mit `“rcalsasound stop”` den Soundtreiber und spielt dann den Patch ein. Anschließend startet man der Treiber erneut mit `“rcalsasound start”`. Für weitere Hinweise sei auf die SuSE-Supportdatenbank verwiesen.

2.6 Wireless Local Area Network (WLAN)

Eine ausführliche Beschreibung zur Installation einer WLAN-Karte und der Infrastruktur findet sich im Wireless-HowTo, zu finden z.B. hier [7].

Kapitel 3

Aufbereitung des Kartenmaterials

Während unserer Arbeit wurde uns angeboten, ARREAL auf der CeBit 2002 vorzustellen. Aus diesem Grund erweiterten wir das System um Daten von dem CeBit-Messegelände in Hannover. Dazu benötigten wir einen genauen Plan des Geländes mit den Standorten der Gebäude, sowie deren Innenansicht mit den Lageplänen der Stände.

Da die Karten für die Innenansicht auf die gleiche Weise erstellt werden, wie die für die Außenansicht und wir mit den vorhandenen Daten vor den gleichen Problemen standen, beschreiben wir hier nur die Bearbeitung der Daten für die Außenansicht.

Die Karten wurden von den Organisatoren der CeBit bereitgestellt. Diese lagen im AutoCAD-Format .dwg vor. Um diese mit ARREAL darstellen zu können, mußten die Karten aufbereitet werden. Zuerst wurden von den Karten unnötige Details entfernt, wie z.B. Bäume, unnötige Knoten auf geraden Linien usw.. Dadurch wird die Größe der zu ladenden Karte und der Objekte reduziert, wodurch diese schneller gezeichnet wird.

Diese so aufbereiteten Karten wurden dann mit Hilfe eines VisualBasic-Programms in das ARREAL-Format umgewandelt. Dieses Programm verwendet das AutoCAD-ADO um die DWG-Karten öffnen und auslesen zu können. Dann wird eine XOF-Datei [1] erstellt, die nun von ARREAL gelesen werden kann.

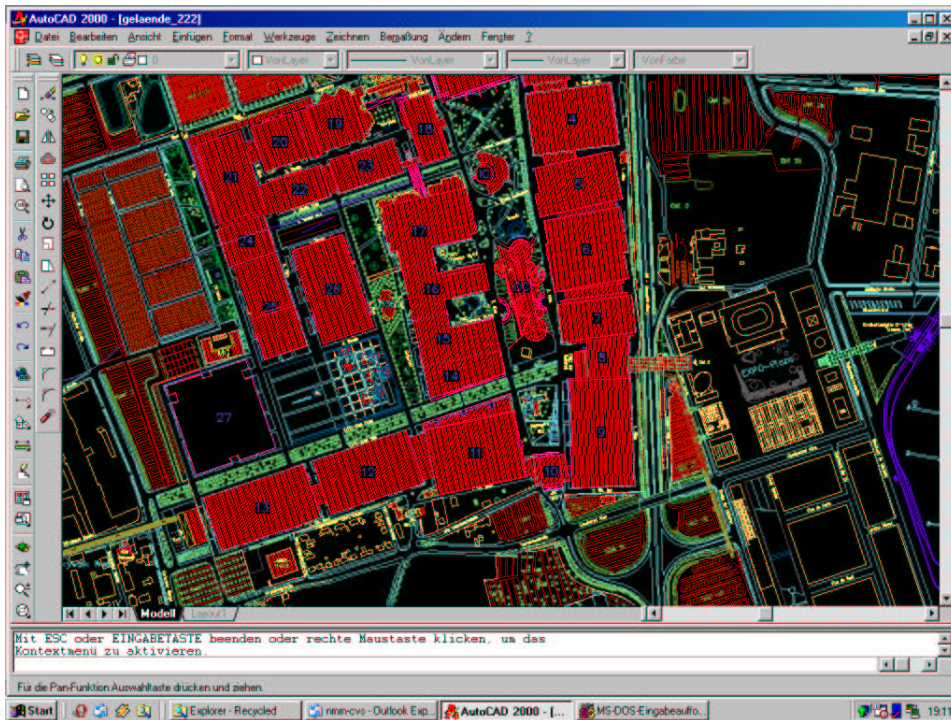


Abbildung 3.1: AutoCAD-Plan der CeBit mit allen Details

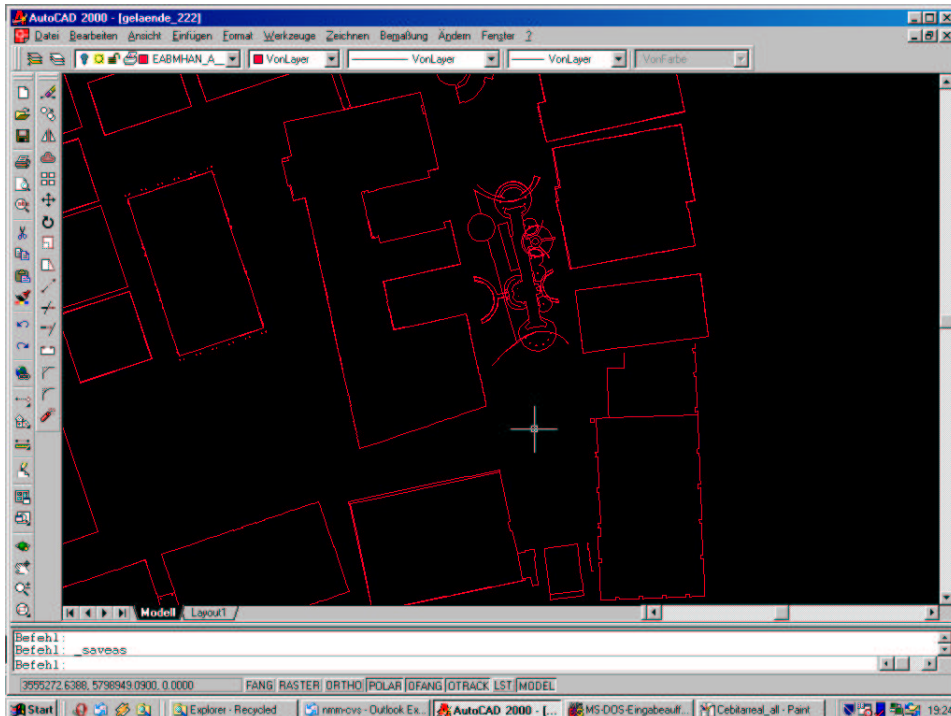


Abbildung 3.2: AutoCAD-Plan der CeBit mit wenigen Details

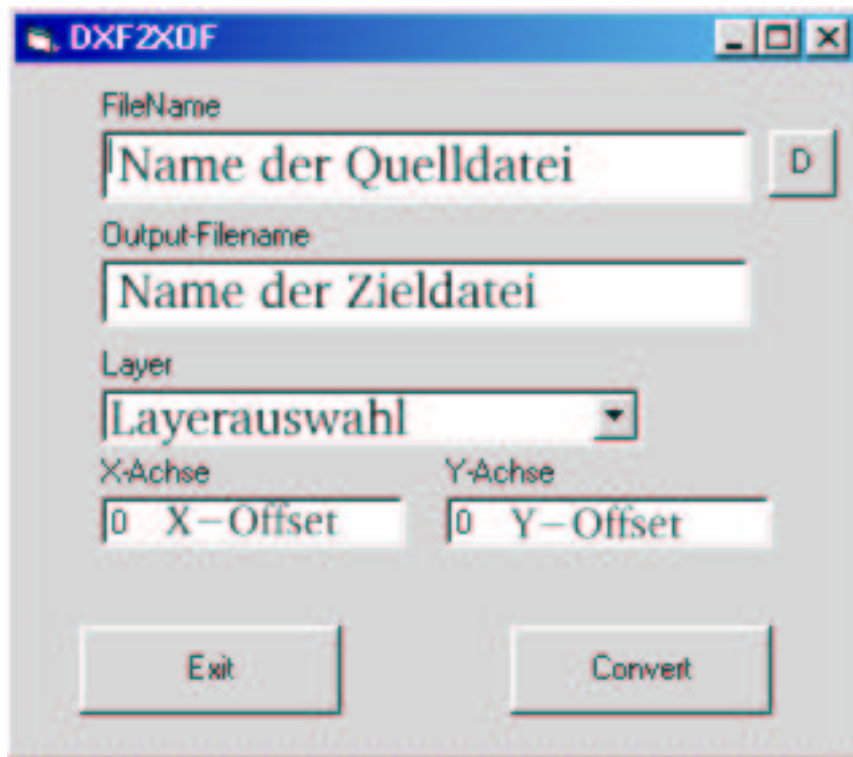


Abbildung 3.3: Benutzeroberfläche

3.1 Beschreibung des Programms

Um die AutoCAD-Datei korrekt auslesen zu können, müssen die Linien von Typ 'Polyline' sein. Diese können in 2D- oder 3D-Koordinaten angegeben werden.

Name der Quelldatei Name und Pfad der Quelldatei, die ausgelesen werden soll.

Name der Zieldatei Name und Pfad der Zieldatei, die dann als Eingabedatei für ARREAL dient.

Layerauswahl Da immer nur 1 Layer ausgelesen werden kann, besteht hier die Möglichkeit zu bestimmen, welcher Layer ausgelesen werden soll.

X- und Y-Offset Hiermit kann festgelegt werden, wie weit der Nullpunkt der auszulesenden Karte von Nullpunkt der ARREAL-Karte abweichen soll, damit diese korrekt angezeigt wird.

```
##LINE_TYPE
##399
##9
##373.79014000017 739.67719999980 0
##206.96500000031 702.61499999929 0
##218.79000000003 649.02499999944 0
##209.30500000016 646.93499999959 0
##220.09415000025 597.33640999998 0
##396.56500000041 636.43499999959 0
##396.56500000041 636.43499999959 0
##396.56500000041 636.43499999959 0
##373.79014000017 739.67719999980 0
```

Abbildung 3.4: Darstellung eines rechteckigen Gebäudes in der XOF-Datei

3.2 Funktionsweise

Jedes Gebäude in der AutoCAD-Datei besteht aus einer einzigen Linie, die die Umrandung darstellt. Solche Linien tragen intern den Namen POLYLINES, so daß sie über diesen Typ identifiziert werden können. Ein Knick in dieser Linie ist als Knoten-Punkt abgespeichert. Da die Darstellung der Gebäude in dieser AutoCAD-Datei 2-dimensional ist, muß man, um die XOF-Datei zu erstellen, eine gewisse Höhe der Mauern festlegen. Die Knoten-Punkte der Polylinien werden nun so in die XOF-Datei geschrieben, daß die Mauern der Gebäude als Flächen dargestellt werden können. Dazu nimmt man eine Linie zwischen 2 Knoten-Punkten, und hat nun mit der festgelegten Höhe 4 Punkte, die ein Rechteck aufspannen. Dieses Rechteck zerlegt man in 2 Dreiecke und schreibt die so gewonnenen Punkte in die XOF-Datei.

Hinweis:

Die Datei, die von diesem Programm erzeugt wird, wurde für die ARREAL-Version für die CeBit 2002 verwendet!

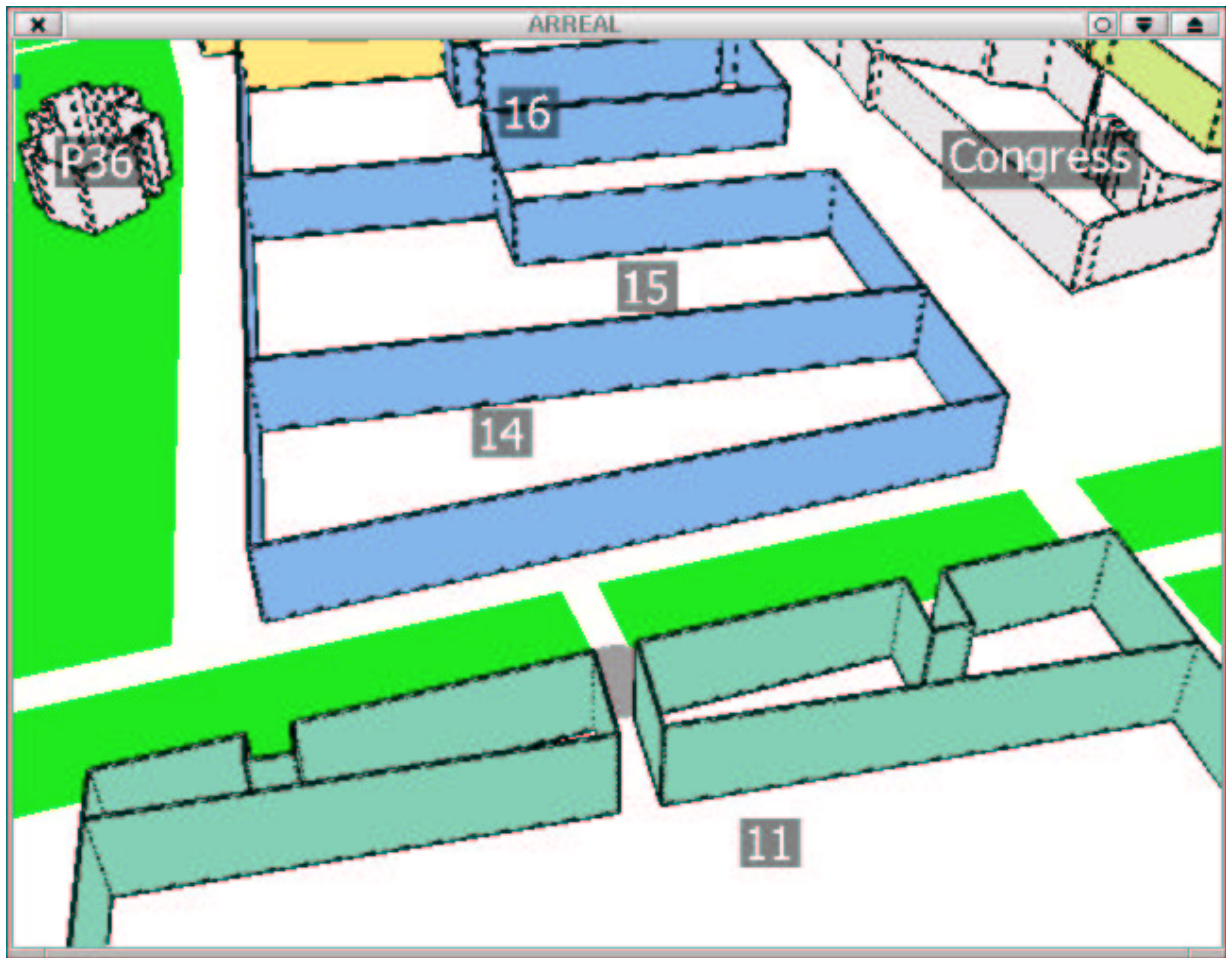


Abbildung 3.5: Darstellung der Karte in ARREAL

Kapitel 4

Erweiterung der graphischen Umgebung von ARREAL

4.1 Einleitung

Wie im ersten Kapitel beschrieben, agierte das ARREAL-System bisher nur als Fußgänger-navigationssystem im Freien. Um dieses System auch als z.B. Messeführer benutzen zu können, musste dieses System um eine Gebäudenavigation erweitert werden. Anhand anderer schon früher realisierter Navigationshilfen für Gebäude wählten wir als mögliche Realisierung eine Erweiterung, die in einer mit Infrarot-Sendern ausgestatteten Umgebung arbeiten soll. Dies ist zwar keine Universallösung gewesen, lies sich aber relativ einfach und schnell umsetzen und gab uns auch die Möglichkeit an unterschiedlichsten Stellen schnell Testumgebungen aufzubauen, da man dafür nur beliebig viele kleine Infrarot-Sender installieren musste. Im Laufe dieser Veränderungen des ARREAL-Designs kam es dann auch noch zu einigen Verbesserungen der graphischen Umgebung, die im Folgenden noch genauer erwähnt werden.

4.2 Veränderungen für die Navigation im Innenbereich

4.2.1 Darstellung des Innenbereichs

Die Visualisierung des Innenbereichs wird wie im ursprünglichen System anhand des entsprechenden Kartenmaterials erzeugt. Dies hat den Vorteil, daß man nichts an der Generierung ändern musste.

Wichtig war einerseits einen sauberen Übergang zu erzeugen, wenn der Benutzer ein Gebäude betritt und andererseits eine sinnvolle Positionierung des Benutzers darzustellen.

4.2.2 Beschreibung der Lokalisierung im Innenbereich

Wir benutzen im Innenbereich einfache Infrarotsender, die jeweils mit einer eindeutigen ID ausgestattet sind. Die einzige Aufgabe dieser Sender ist es, ständig diese ID über das Infrarotsignal zu senden. Zur Verarbeitung dieser ID wurde beim Installieren der Sender ihre Position auf der Karte und ihre Abstrahlrichtung festgelegt. Diese Daten werden in einer ASCII-Datei abgelegt und können durch die ID wieder ausgelesen werden und damit der Benutzer positioniert werden kann. Da diese Sender immer nur einen bestimmten Bereich beschreiben, kann die gezeigte Benutzerposition auch einige Meter von der wirklichen Position abweichen, daher muss man für genauere Bestimmung mehr Infrarotsender installieren, die sich dann in ihren Sendebereichen überschneiden. Die Berechnung der Position in einem solchen Fall wird im Abschnitt 'Positionierung bei unterschiedlichem Infrarotempfang' beschrieben.

4.2.3 Die neue Klasse IRDevice

Die erste Lösung, um die Infrarotsignale zu verarbeiten, bestand darin, ein von der Firma Eyeled [9] entworfenes Programm als zweiten Prozess laufen zu lassen, was nur die Aufgabe hatte, die empfangene ID zu lesen und in die Datei 'beaconid' zu schreiben. Das eigentliche Programm musste dann nur regelmässig testen, ob sich der Eintrag in dieser Datei geändert hat, was bedeutet, dass ein Infrarotsignal empfangen wurde. Diese Version funktioniert ohne Probleme, hat aber die Unschönheit, dass man zwei Programme nebeneinander laufen lassen muss. Daher sollte diese Variante geändert werden, so dass der IrDA-Empfang genauso wie der GPS-Empfang direkt im eigentlichen Programm verarbeitet wird.

Die Daten des Benutzers werden von der Klasse UserGPS verarbeitet, dieser wurde jetzt als Hilfe die Klasse IRDevice bereitgestellt, um den Infrarotempfang zu verarbeiten.

4.2.3.1 Beschreibung der Klasse IRDevice

Die Klasse IRDevice ist von der Klasse QObject der QT-Bibliothek [10] abgeleitet. Außerdem werden Funktionen aus `elt.h` benutzt, die für die Steuerung des Infrarotempfängers zuständig sind und das Übertragungsprotokoll kennen. Der Empfang des Infrarotsignals wird mittels der Signal-Slot-Technik der Klasse QObject verarbeitet. Dabei wird das Signal auf den Port des Infrarotempfängers gesetzt und schickt bei einem Ereignis (Empfang der IrDa-Signals) die empfangene ID zu dem Slot, der die Variable mit der aktuellen Baken-ID aktualisiert.

4.2.3.2 Anpassung des Makefiles

Da in der Klasse IRDevice auch die QT-Bibliothek benutzt wurde, muss diese noch in der Liste der benutzten Bibliotheken hinzugefügt werden. Außerdem muss die Headerdatei der Klasse IRDevice zuerst mit moc (Meta Object Compiler) vorkompiliert werden, um den eigentlichen Source-Code der Klasse zu erstellen (`moc_IRDevice.cc`). Danach werden `moc_IRDevice.cc` bzw. `IRDevice.h` noch in den kompletten Compilierablauf eingebunden.

4.2.4 Positionierung bei unterschiedlichem Infrarotempfang

Hier fand wiederum eine Erweiterung des Quellcodes statt, und zwar wurden weitere Routinen zur genaueren Positionierung im Innenbereich in der Datei `UserGPS.cc` implementiert. Zur Bestimmung der Benutzerposition bei Infrarotempfang werden vier Varianten unterschieden:

1. kein Empfang,
2. Empfang einer ID,
3. Empfang von zwei IDs und
4. Empfang von drei IDs.

Zu erwähnen ist natürlich noch, dass nie zwei oder drei IDs zeitgleich empfangen werden können, aber dass der Empfang zwischen unterschiedlichen IDs innerhalb eines kleinen Zeitintervalls wechseln kann.

Kommen wir nun zur Behandlung dieser Fälle:

kein Empfang Der Benutzer wird immer noch auf der letzten bestimmten Position angezeigt.

Empfang einer ID Der Benutzer wird in der Mitte des Sendebereichs mit Blickrichtung zum Sender positioniert.

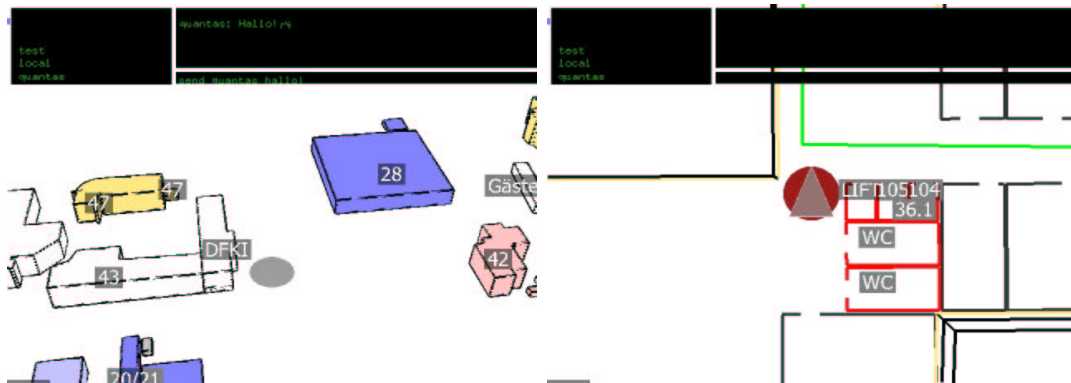


Abbildung 4.1: Perspektivenwechsel

Empfang von zwei IDs Hier wird zuerst der Mittelpunkt der Strecke berechnet, die zwischen Mittelpunkt des Sendebereichs des einen Senders und Mittelpunkt des Sendebereichs des anderen Senders liegt, und dort der Benutzer angezeigt.

Empfang von drei IDs Hier wird das durch die drei Baken aufgespannte Dreieck betrachtet und der Schwerpunkt berechnet. Der Benutzer erhält dies als geschätzten Aufenthaltsort.

4.2.5 Wechsel zwischen Innen und Außen

Der Wechsel zwischen der Innenansicht, also Positionierung des Benutzers in einem Gebäude, und der Aussenansicht koppelt sich an das letzte empfangene Signal. Dabei hat das IrDA-Signal eine höhere Priorität. D.h., da man davon ausgeht, dass im Aussenbereich keine Infrarot-Baken angebracht sind, wird erst getestet, ob man eine solche empfängt und falls ja, die Gebäudeansicht gezeigt und der Benutzer an dieser Bake positioniert, ansonsten wird versucht ein GPS-Signal zu empfangen, um den Benutzer entsprechend auf der Karte für den Aussenbereich zu positionieren. Falls keines der beide Signalarten empfangen wird, bleibt die aktuelle Ansicht vorhanden.

4.3 Weitere Änderungen der graphischen Umgebung

4.3.1 Kalibrierungsmodus

Die Kalibrierung des Navigationssystem soll deswegen durchgeführt werden, da der GPS-Empfang öfters ein wenig in den Werten schwanken kann.

Im System werden zur Kalibrierung die GPS-Koordinaten und die Kartenkoordinaten dreier Punkte abgelegt, von denen man genau weiss, wo sie sich in Wirklichkeit befinden. Die Kalibrierung des Navigationssystems läuft nun wie folgt. Man startet mit der Taste 'Tab' den Kalibriermodus. Somit erscheint auf dem Display der Hinweis 'CLICK POINT TO CALIBRATE..' und es werden die drei vordefinierten Kalibrierungspunkte auf der Karte als rote Kreuze angezeigt.



Abbildung 4.2: Kalibrierungsmodus

Man geht nun zu einem dieser Punkte. Erreicht man diesen, so muss man ihn auf der Karte anklicken und startet damit den hier aufgeführten Algorithmus, der in `Locations.cc` implementiert ist. Die Kalibrierung kann beliebig oft wiederholt werden. Es handelt sich hierbei um einen 'Standard'-Algorithmus aus der Computergrafik [?] .

Kalibrierungsalgorithmus:

Eingabe:

- Koordinaten der Kalibrierpunkte A, B und C auf der Karte
- Koordinaten der Kalibrierpunkte A, B und C mit GPS
- Korrektur für x- und y-Koordinaten (Differenz zwischen aufgesuchtem Kalibrierungspunkt und aktueller Benutzerposition)

Berechnung:

$$\begin{aligned}
 T1 &= \begin{pmatrix} 1 & 0 & -CG.x \\ 0 & 1 & -CG.y \\ 0 & 0 & 1 \end{pmatrix} \\
 T2 &= \begin{pmatrix} 1 & 0 & CM.x + korrekturX \\ 0 & 1 & CM.y + korrekturY \\ 0 & 0 & 1 \end{pmatrix} \\
 Hilf1 &= \begin{pmatrix} AM.x - CM.x & BM.x - CM.x \\ AM.y - CM.y & BM.y - CM.y \end{pmatrix} \\
 Hilf2 &= \begin{pmatrix} AG.x - CG.x & BG.x - CG.x \\ AG.y - CG.y & BG.y - CG.y \end{pmatrix} \\
 Hilf &= Hilf1 * Hilf2^{-1} \\
 M &= \begin{pmatrix} Hilf(1,1) & Hilf(1,2) & 0 \\ Hilf(2,1) & Hilf(2,2) & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
 Transformation &= (T2 * M) * T1
 \end{aligned}$$

Somit erhält man die korrigierte Transformationsmatrix zum Umrechnen der GPS-Daten auf das Karten-Koordinatensystem.

4.3.2 Zusatzfunktionen

4.3.2.1 Offline-Modus für PC-Betrieb

Es besteht die Möglichkeit das System z.B. auf einem PC ohne GPS- und IrDa-Empfänger zu starten. Diese Variante kann - wie auf der CeBit durchgeführt - zur Präsentation des Systems benutzt werden. Dies bietet auch die Möglichkeit - ausgehend von einem Meszenario - PCs aufzustellen, die Besuchern einen Überblick des Geländes geben, andere Besucher, die mit dem Navigationssystem unterwegs sind, zu zeigen oder auch zusätzliche Informationen zu Gebäuden und Ständen zu zeigen. Hier gibt es auch die Möglichkeit, mittels der Cursortasten den virtuellen Benutzer ohne GPS-Empfang zu bewegen und auch den Empfang von IrDa-Baken zu simulieren.

4.3.2.2 Infos zu Gebäuden

Durch einen Mausclick auf Gebäude oder im Innenbereich z.B. auf einen Ausstellungsstand wird im Display eine kurze Information dazu angezeigt.

4.3.2.3 Scrollen der Karte

Im Normalfall wird die Anzeige der Karte immer auf die aktuelle Benutzerposition ausgerichtet. Um sich einen Überblick über die umliegende Gegend zu verschaffen oder andere Benutzer zu suchen, kann man mit der Maus bzw. dem Touchscreen durch 'Drag and Move' die Karte verschieben. Dies funktioniert sowohl im Innen- als auch im Aussenbereich.

4.3.2.4 Zoomen der Karte

Die letzte Zusatzfunktion ist das Hinein- und Herauszoomen der Karte, je nachdem, ob man sich mehr auf den lokalen Bereich konzentrieren will oder ein möglichst großes Umfeld sehen will. Diese Funktion ist immer auf die aktuelle Benutzerfunktion fokussiert.

Kapitel 5

Partner-Radar und Kommunikation mittels Text-Nachrichten

5.1 Einleitung

Man kennt die Situation: Eine Gruppe von Leuten geht über die CeBit, und Einer geht verloren. Sich nun wieder zu finden ist nicht ganz einfach. Sicherlich kann man über Handy's einen Treffpunkt ausmachen, doch auf einen Areal, auf dem man sich nicht auskennt, gestaltet sich dies schwierig und man kann nicht abschätzen, wie lange man braucht, um den Treffpunkt zu finden und zu erreichen. Einfacher geht es mit einer Navigationssoftware, die einen sicher zum abgesprochenen Treffpunkt führt, oder noch einfacher, daß man auf dem Display nicht nur seinen eigenen Standort sieht, sondern auch den der Gruppe, und ihr über eine Text-Nachricht mitteilt, sie möge doch bitte kurz warten, damit man aufschließen kann.

Aufgabe war es, ARREAL um genau diese Funktionalität zu erweitern, so daß Benutzer sich den Standort anderer Benutzer anzeigen lassen können und mit ihnen mittels Text-Nachrichten kommunizieren können.

5.2 Architektur

Das Positionsupdate wurde mit Hilfe des UDP-Protokolls realisiert. Dies ist zum ersten schneller als TCP und zweitens ist ein Verlust eines Pakets nicht tragisch, da die aktuelle Position in einer Endlosschleife versandt wird.

Bei Text-Nachrichten muß jedoch gewährleistet sein, daß diese auch beim Empfänger ankommt, deshalb fiel hier die Wahl des Protokolls auf TCP.

Aus Datenschutz-Gründen wählten wir eine peer-to-peer-Architektur. Jeder Teilnehmer schickt Positions-Daten an alle Teilnehmer aus seiner Liste. Auf der anderen Seite werden nur Daten von Leuten, die der Empfänger in seine Liste aufgenommen hat, ausgewertet

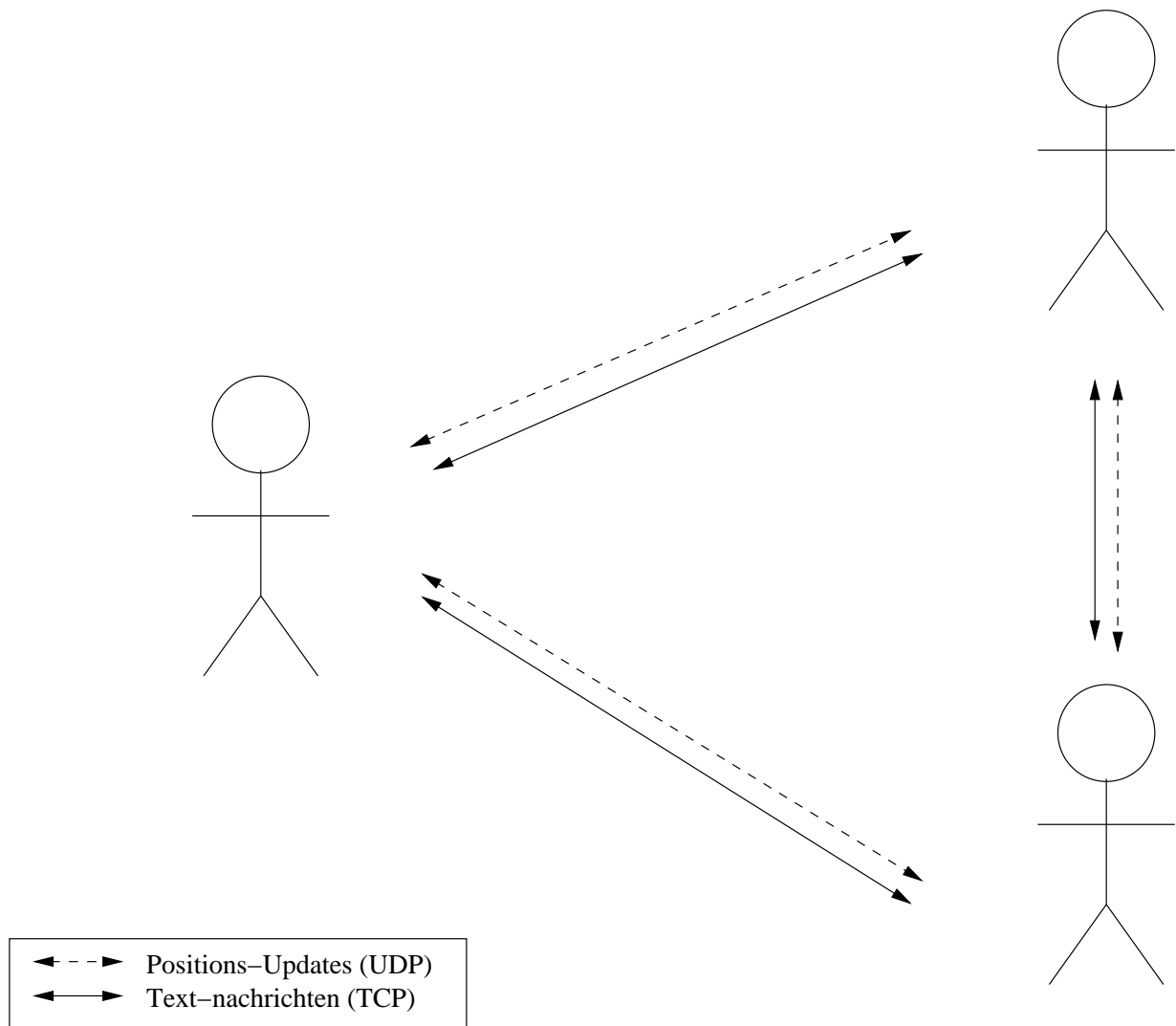


Abbildung 5.1: Kommunikation der Client's untereinander

und angezeigt. Bei diesem Vorgehen benötigt man keinen Server, der sie einzelnen Daten sammelt, verarbeitet und weiter verschickt. Diese Aufgabe übernehmen die einzelnen Client's für sich selbst. Die Architektur erlaubt es jedoch, einen Server zu konfigurieren, der die Position aller angeschlossenen Client's anzeigt, zum Beispiel für Überwachungszwecke.

5.3 Übersicht über die Objekte und Dateien

Dafür erstellen wir zuerst ein Netzwerk-Objekt. Dies ist das zentrale Objekt, mit dem die Netzwerk-Anbindung initialisiert und alle notwendigen Sockets und Erweiterungsobjekte erstellt werden.

```
name arreal
client_port_TCP 5001
client_port_UDP 5002
server_port_TCP 5003
server_port_UDP 5004
```

Abbildung 5.2: Beispiel einer Netzwerk-Konfigurations-Datei

```
quantas 192.168.123.2 5002 5001
local 192.168.123.1 5006 5005
```

Abbildung 5.3: Beispiel einer BuddyList

Ein Erweiterungsobjekt ist das `ServerObjectUDP`, das unsere aktuelle Position an angeschlossene Rechner verschickt. Das zweite Objekt ist `ServerObjectTCP`, mit dem in einer späteren Version die Text-Nachrichten verschickt werden sollen.

Als Konfigurationsdateien dienen uns zwei Text-Dateien. Zum einen eine Datei, in der Informationen über das Netzwerk steht, wie Namen im Netz, UDP- und TCP-Portnummer, usw. .

Die zweite Datei enthält die IP-Adressen oder Rechner-Namen der Leute, an die unsere aktuelle Position verschickt werden soll.

Zudem wird noch eine Klasse `NetworkObject` erstellt, die die Anzeige und Verwaltung der Daten der `RemoteUser` regelt. Diese dient als Interface zur Implementierung weiterer Objekt, zum Beispiel zur unterschiedlichen Darstellung von menschlichen Benutzer und Bussen etc.

5.4 Konfigurations-Dateien

Es gibt 2 Konfigurations-Dateien. Zum ersten die Datei *netconf* , in der festgelegt wird, welcher Name der Benutzer im ARREAL-Netzwerk hat und die Ports, für die UDP-/TCP-Kommunikation. Die zweite Datei nennt sich *buddyList* . In ihr stehen die Benutzer, an die unsere aktuelle Position verschickt wird und die uns ihre aktuelle Position schicken dürfen. Benutzer, die nicht in dieser Liste stehen, werden nicht angezeigt.

Eine Zeile in dieser Datei hat folgendes Format:

```
name IP-Adresse UDP-port TCP-Port
```

5.5 Network - Objekt

Dieses Objekt ist das Hauptobjekt. Es erstellt jeweils einen UDP- und einen TCP-Socket, um Daten von anderen Benutzern zu empfangen. Dies sind (bei UDP) die aktuellen Koordinaten der Benutzer. Für jeden angeschlossenen Benutzer wird ein `NetworkObject` erstellt (Erklärung weiter unten), die alle zusammen in einem Array verwaltet werden. so wird bei Empfang einer Nachricht die Position des jeweiligen Benutzers geupdatet und angezeigt. In diesem Objekt werden der UDP- und TCP-Server initialisiert, deren Beschreibung weiter unten folgt.

Um die eigenen Koordinaten an alle angeschlossenen Benutzer zu senden, muß an geeigneter Stelle die `send`-Funktion aufgerufen werden, die dann die Koordinaten verschickt.

5.6 ServerObjectUDP

Dieses Objekt dient dazu, die aktuelle Position an einen bestimmten Benutzer im Netzwerk zu senden. Es stellt einen UDP-Socket bereit, über den mittels der Methode

```
int send(double* xpos, double* ypos, string name, struct sockaddr_in client)
```

die Informationen gesendet werden können.

5.7 ServerObjectTCP

Dieses Objekt dient dazu, TCP-Nachrichten an andere Benutzer zu senden. Es stellt einen TCP-Socket für diese Zwecke bereit. Zudem stellt es beim Senden einer Nachricht den eigenen Netzwerk-Namen voran, der in der Datei `netconf` angegeben ist. Durch Aufruf von

```
int send(const char buffer[], struct sockaddr_in client);
```

wird der Inhalt von `buffer` an `client` gesendet.

5.8 NetworkObject

Dies ist das Interface zur Implementierung der einzelnen Remote-Objekte, die auf dem Display angezeigt werden können. Dadurch enthält jedes Objekt eine Variable für die x- und eine für die y-Koordinate. Zudem enthalten alle Objekte, die von dieser Klasse erben, eine `draw`-Funktion, nach deren Aufruf das entsprechende Objekt auf dem Display angezeigt wird.

5.9 Starten der Netzwerkunterstützung

Nachdem Das Netzwerk-Objekt initialisiert wurde, muß das Netzwerk mittels `net->start()` gestartet werden.

Änderungen in der Datei ' Visual_GL.h ' :

```
#include ' 'netsup/network.h' '
```

Änderungen in der Datei ' Visual_GL.cc ' :

```
#include ' 'windows.cc' '
```

```
extern Network* net;
```

```
:
```

```
void keyPressed(unsigned char key, int x, int y)
```

```
{
```

```
:
```

```
case KEY_ESCAPE:
```

```
net->stop();
```

```
:
```

```
}
```

```
:
```

```
void drawGLScene(){
```

```
{
```

```
glutSetWindow(window),
```

```
:
```

```
renderAllSubWindows(),
```

```
glutSetWindow(window)
```

```
}
```

```
:
```

```
void visualize(int argc, char **argv) {
```

```
{
```

```
:
```

```
initWindows(window, net);
```

```
}
```

Änderungen in der Datei ' main.cc ' :

```
int main()
```

```
{
```

```
:
```

```
net = new Network();
```

```
net->start();
```

```
myUser.net=net
```

```
dummy.net=net
```

```
:
```

```
}
```

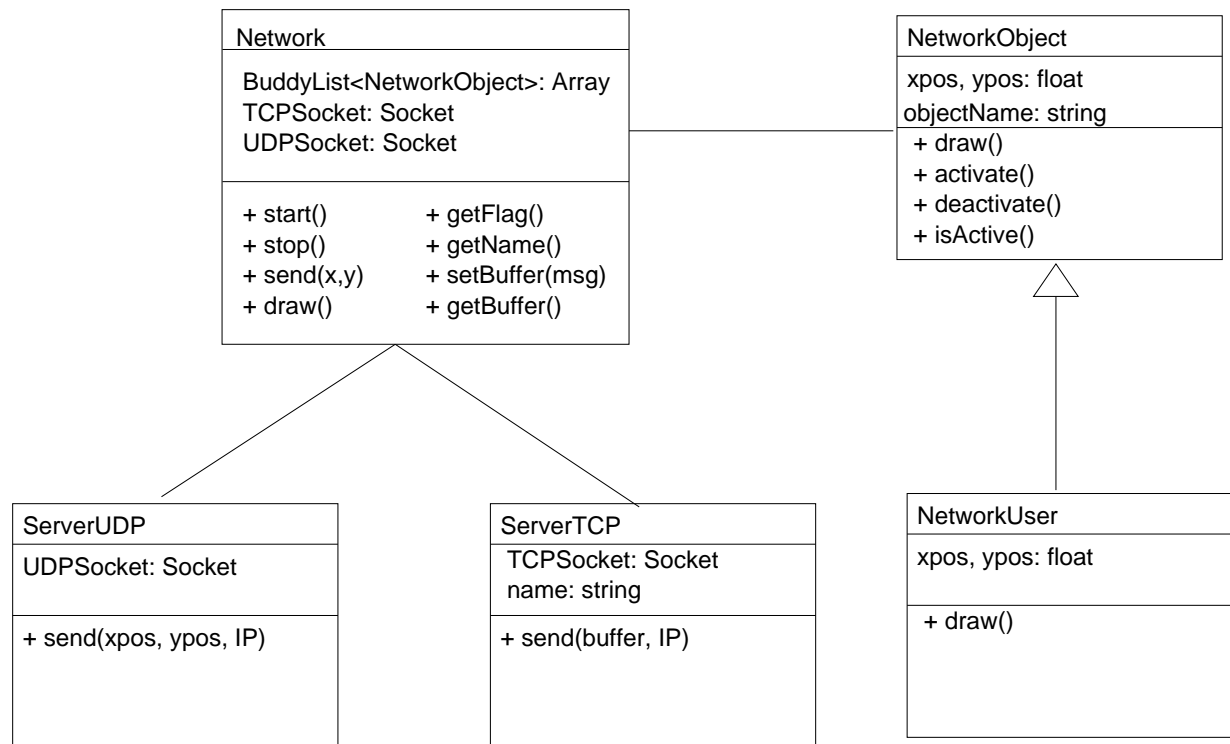


Abbildung 5.4: UML-Klassendiagramm

Damit die Benutzer, die in der *buddyList* angegeben sind, auch gezeichnet werden, muß in der Funktion, in der die komplette Szene gezeichnet wird, noch folgender Funktionsaufruf eingefügt werden:

```
net->draw();
```

Um die aktuelle Position auch zu senden, fehlt noch der Aufruf der Funktion

```
net->send(x, y);
```

an geeigneter Stelle. Wir wählten dazu einen Aufruf in der Methode `draw` des Objekts `UserGPS`. Dadurch wird die aktuelle Position an vorgegebene Benutzer gesendet.

Damit die Konfigurationsdateien auch gefunden werden, muß darauf geachtet werden, daß sich diese in dem Verzeichnis befinden, in dem das ARREAL-Programm gestartet wird. Dies ist im Moment das Verzeichnis, in dem die Daten für Karten etc. liegen.

Beim Beenden des Programms, muß die Funktion `net->stop()` aufgerufen werden. Dadurch wird eine Logout-Nachricht an alle angeschlossenen Benutzer geschickt, wodurch die Anzeige der eigenen Position auf deren Display abgeschaltet wird.

Zum Einbinden der Chat-Unterstützung ist zusätzlicher Code erforderlich.

Kapitel 6

Chat-Unterstützung

In diesem Kapitel soll das Versenden von kurzen Textnachrichten erklärt werden. Zuerst werden wir dabei auf die drei implementierten Fenster innerhalb des Systems näher eingehen. Später wird dann der Prozeß des Nachrichtenschickens und den dazu nötigen Methoden und Klassen erklärt.

6.1 Informationsfenster

In unserer Implementierung sind innerhalb des Systems drei Fenster integriert, die zum Einen Informationen über die teilnehmenden Benutzer wiedergeben, zum Anderen die Möglichkeit bieten Nachrichten an andere Benutzer zu versenden bzw. von teilnehmenden Benutzern Nachrichten zu empfangen. Die Teilfenster wurden mit Hilfe der Open GL Bibliothek implementiert. Die Methode

$$\text{glutCreateSubWindow}(\text{mainwindow}, \text{border}_1, \text{border}_2, \text{height}, \text{width})$$

bietet die Möglichkeit ein Fenster (Subwindow) innerhalb eines Fensters (*mainwindow*) zu kreieren. Die Variable *mainwindow* gibt dabei das zum Subwindow gehörige Mainwindow an. Ausserdem werden die Koordinaten des Subwindows (*border₁*, *border₂*) sowie die Grösse des Fensters spezifiziert.

6.1.1 Angemeldete Benutzer

Das System verfügt über ein Fenster (siehe Abbildung 5.1, linke Fenster) das Aufschluß über die zur Zeit angemeldeten Benutzer gibt. Der im Fenster erscheinende Name des jeweiligen Benutzer wird ebenfalls beim Versenden von Nachrichten benötigt, da dieser der Nachricht vorangestellt wird um den Empfänger dieser Nachricht zu kennzeichnen.

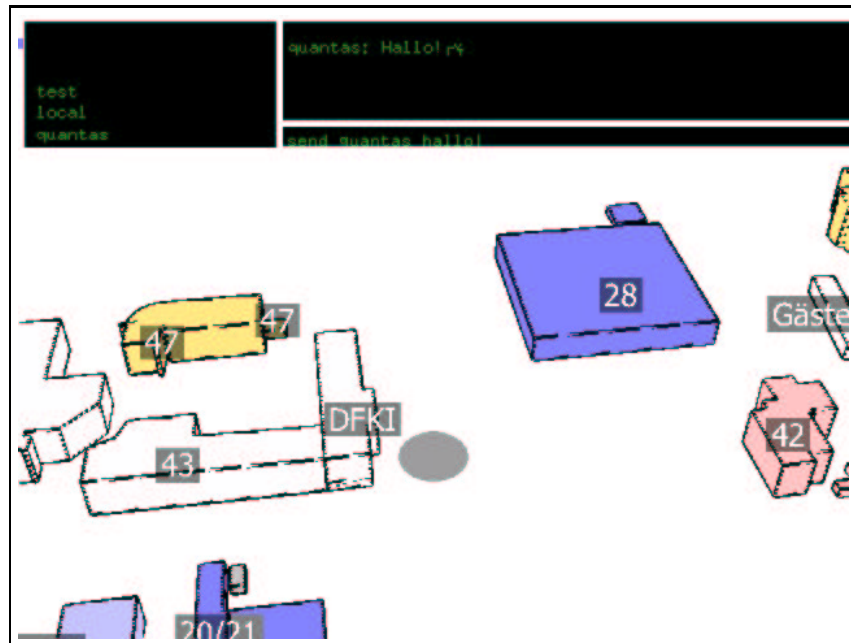


Abbildung 6.1: Die Karte: Hier sieht man die Umgebung in der sich der Benutzer befindet (in diesem Fall, Gebäude der Universität der Saarlandes), sowie die drei Fenster die zur Kommunikation zwischen den Benutzern dient.

Durch das erscheinen der Namen der angemeldeten Benutzer im Fenster kann der Nachrichtenversender sicherstellen, dass Nachrichten nicht an Benutzer gesendet werden, die zur Zeit nicht im System angemeldet sind.

6.1.2 Nachrichteneingabe

Dieses Fenster (siehe Abbildung 5.1) ermöglicht dem Benutzer das Versenden von Nachrichten an andere Benutzer. Das Fenster ist aktiv sobald man sich mit dem Mauszeiger im Bereich dieses Fensters befindet, d.h. nur dann wird das Schreiben von Nachrichten auch erkannt. Dies hat den Vorteil, dass man bestimmte Tasten des Keyboards nicht doppelt belegen muß, d.h. einerseits eine Tastaturbelegungen innerhalb des Systems, andererseits um Nachrichten schreiben zu können. Die geschriebenen Nachrichten werden mit der Return-Taste abgeschlossen und erscheinen dann im Nachrichten Fenster, welches wir im nächsten Abschnitt erklären werden.

6.1.3 Empfangene und gesendete Nachrichten

In diesem Nachrichtenfenster (siehe Abbildung 5.1) erscheinen Nachrichten, die von anderen Benutzern an die jeweilige Person gesendet wurden sowie Nachrichten die von der

Person selbst an andere Benutzer geschickt wurden.

Um die Karte des jeweiligen Terrains, in dem sich die Benutzer befinden, nicht zu verdecken haben wir uns darauf geeinigt, dass nur die letzten sechs Nachrichten, die empfangen bzw. selbst geschrieben wurden, im Nachrichten Fenster angezeigt werden. Die Nachrichten die vor den sechs Nachrichten gesendet wurden werden gelöscht.

6.2 Senden einer Nachricht über das Netzwerk

Für das Versenden von Nachrichten benutzen wir TCP als Transport-Protokoll, während für Positionsdaten, die Positionsveränderungen beinhalten, UDP verwendet wird.

Der Benutzer gibt über das "Compose Message Window" die zu sendende Nachricht ein. Die Nachricht selbst muß dabei einem speziellen Format entsprechen:

send Name Nachricht

Send ist ein Schlüsselwort innerhalb der Umgebung der Nachrichtenerstellung das angibt, dass der Benutzers eine Nachricht versenden möchte, *Name* gibt an, an welche Person diese Nachricht adressiert ist. Der User hat dabei zwei Möglichkeiten. Er kann eine Nachricht explizit an eine Person oder an alle Benutzer zusammen adressieren. Den Namen der jeweiligen Person entnimmt der Benutzer aus dem "buddy window", das Schlüsselwort um an alle Personen eine Nachricht zu verschicken ist *all*. Nach diesen beiden Schlüsselwörtern beginnt der Messagebody. Wenn der Benutzer die Nachricht mit der return-Taste abschließt, beginnt die Klasse *Parser* den Ausdruck zu interpretieren. Falls der Ausdruck nicht mit dem beschriebenen Format übereinstimmt wird eine Fehlermeldung an das "message window" übergeben. Anderenfalls wird die Methode *send()* der Klasse *serverTCP* mit den Argumenten "Nachricht" und "Adressat/en" aufgerufen. Sobald die Nachricht bei den Benutzern ankommt wird diese in der Klasse *network* an das "Message Window" des jeweiligen Benutzers und des Absenders übergeben.

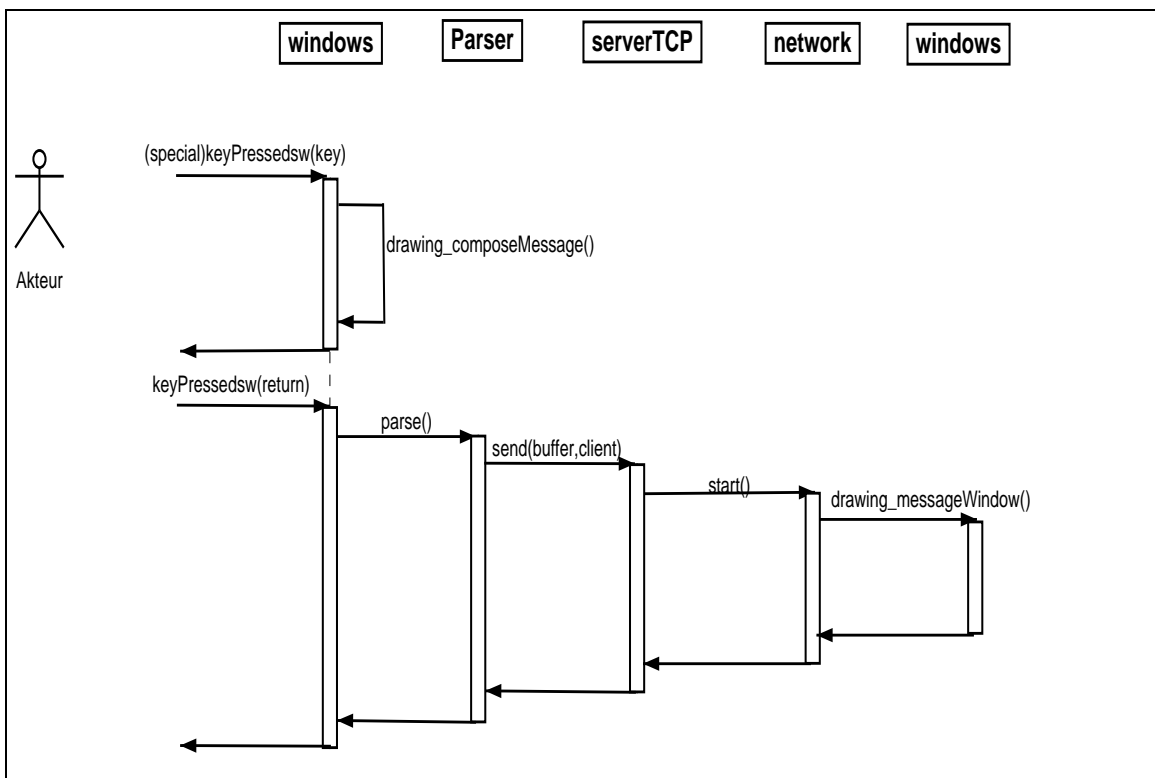


Abbildung 6.2: Protokoll für das Senden von Nachrichten

Kapitel 7

Wichtige Hinweise zum neuen ARREAL-System

Die Entwicklung des ARREAL-Systems läuft schon seit einiger Zeit. Da es mittlerweile Weiterentwicklungen z.B. im Bereich der benutzten Bibliotheken gibt, ist es wichtig kurz aufzulisten, welche man für einen stabilen Betrieb der Systems benötigt. Wichtige Teile sind dabei der C++-Compiler, die Version der QT-Bibliothek und auch die Version der OpenGL-Bibliothek (bzw. Mesa-Bibliothek).

7.1 Lauffähige Versionen

Folgende Versionen werden in unserem Xybernavt-Prototypen verwendet und verursachen keine Probleme:

- Als C++-Compiler wird die Version 2.95 des Gnu-Compiler gcc benutzt.
- Mit der QT-Bibliothek gab es zwar noch keine Probleme, das lag aber vielleicht auch nur daran, dass anhand der anderen aufgetretenen Fehler keine andere Version mehr getestet wurde. Lauffähig ist aber auf jeden Fall die Version 2.3.
- Als OpenGL-Bibliothek wird wie unter Linux üblich Mesa benutzt. Die ersten Versionen von ARREAL liefen mit Mesa 2.6, mittlerweile sind wir aber schon bei Mesa 3.4.2 gelangt. D.h. man kann sich getrost in diesem Bereich aufhalten.

7.2 Aufgetretene Fehler bei anderen Versionen

Im Verlauf der Entwicklung haben wir auch schon versucht auf neuere Systeme wie z.B. Suse 8.2 und auf neuere Bibliotheken umzusteigen. Hier listen wir einige markante Probleme bzw. Fehlermeldungen auf, die dabei entstanden:

- Benutzt man Mesa-Bibliotheken ab der Version 4.0 so entsteht ein nicht lokalisierter Speicherzugriffsfehler. Dieser tritt nämlich entweder direkt beim Start des Programms oder bei einer gewissen Aktion, z.B. Mausklick, erst nach einigen Sekunden oder Minuten auf.
- Probleme verursacht auch die bisher neuste Version des Gnu-Compilers (Version 3.2). Und zwar geht es um das Einbinden und Wiederfinden von Headerdateien aus der Standard-Bibliothek. Z.B. muss man statt `#include <iostream.h>` nun `#include <iostream>` benutzen und es haben sich einige Funktionen geändert. Da auch schon die Version 3.1 größere Änderungen mit sich gebracht hat, kann man davon ausgehen, dass dort dieselben Fehlermeldungen auftreten.

Kapitel 8

Doxygen Klassen-Dokumentation

8.1 Network Klassenreferenz

Netzwerk-Objekt.

```
#include <network.h>
```

Öffentliche Datenelemente

- **Network** ()
- **~Network** ()
- void **start** ()
- void **stop** ()
- int **send** (double x, double y)
- **NetworkUserIterator** * **newNetworkUserIterator** ()
- void **draw** ()
- bool **getFlag** ()
- string **getName** ()
- void **setBuffer** (string mes)
- string **getBuffer** ()

Öffentliche Attribute

- int **tcp_socket**
- int **udp_socket**
- string **msg**
- vector< **NetworkUser** *> **buddyList**

- int `maxsize`
- `ServerTCP * serverTcp`

8.1.1 Ausführliche Beschreibung

Netzwerk-Objekt.

In dieser Klasse wird das Netzwerk-Objekt implementiert. Es stellt einen UDP-Socket bereit, über denen Positionsupdates von anderen ARREAL-Benutzern empfangen und ausgewertet werden. Zudem stellt es einen TCP-Socket bereit, über den Text-Nachrichten empfangen werden können.

Autor:

Christoph Wellner <wellner@studcs.uni-sb.de>

8.1.2 Beschreibung der Konstruktoren und Destruktoren

8.1.2.1 `Network::Network ()`

Initialisiert das Netzwerk mit den Daten, die in den Dateien 'netconf' und 'buddyList' angegeben sind

8.1.3 Dokumentation der Elementfunktionen

8.1.3.1 `void Network::draw ()`

Zeichnet alle aktiven Benutzer in der buddyList. Ein User ist aktiv, sobald eine Positionsnachricht empfangen wurde

8.1.3.2 `string Network::getBuffer ()`

Funktion zum Auslesen des Nachrichten-Puffers, wenn eine TCP-Nachricht empfangen wurde

Rückgabewerte:

String Empfangene Nachricht

8.1.3.3 `bool Network::getFlag ()`

Zeigt an, ob eine TCP-Nachricht empfangen wurde

Rückgabewerte:

true eine TCP-Nachricht wurde empfangen

false es liegen keine aktuellen Nachrichten vor

8.1.3.4 NetworkUserIterator* Network::newNetworkUserIterator ()

Liefert einen NetzwerkUser-Iterator der eigenen buddyList

Rückgabewerte:

NetworkUserIterator Zeiger auf den Iterator

8.1.3.5 int Network::send (double *x*, double *y*)

Sendet die aktuelle Position, die über die Variablen *x* und *y* angegeben ist, an alle Benutzer in der buddyList

Parameter:

x x-Koordinate der aktuellen Position

y y-Koordinate der aktuellen Position

8.1.3.6 void Network::setBuffer (string *mes*)

Funktion zum Setzen des Nachrichten-Puffers für TCP-Nachrichten

Parameter:

mes String der zu setzenden Nachricht

8.1.3.7 void Network::start ()

Methode zum Starten des Netzwerks. Es werden 2 Threads initialisiert und gestartet, die dem Empfang von UDP- und TCP-Paketen dienen.

8.1.3.8 void Network::stop ()

Stoppen der Netzwerk-Unterstützung. Bei Aufruf dieser Funktion werden der UDP- und TCP-Socket geschlossen, zudem wird eine Logout-Nachricht an alle angeschlossenen Clients geschickt.

8.1.4 Dokumentation der Datenelemente

8.1.4.1 `vector<NetworkUser *> Network::buddyList`

Anzahl der Einträge in der buddy-List

8.1.4.2 `int Network::maxsize`

Zeiger auf den TCP-Socket zum Versenden von TCP-Nachrichten

8.1.4.3 `string Network::msg`

Die Buddy-Liste

8.1.4.4 `int Network::tcp_socket`

UDP-Socket, um Positions-Updates von anderen Benutzern zu empfangen

8.1.4.5 `int Network::udp_socket`

Nachrichten-Puffer für eingehende TCP-Nachrichten

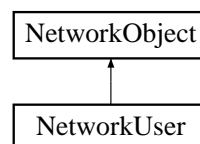
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- `network.h`

8.2 NetworkUser Klassenreferenz

```
#include <NetworkUser.h>
```

Klassendiagramm für NetworkUser::



Öffentliche Datenelemente

- `NetworkUser` (string name, struct sockaddr_in adr)

- `~NetworkUser ()`
- `void draw ()`

8.2.1 Ausführliche Beschreibung

Mit diesem Objekt werden andere ARREAL-Benutzer repräsentiert

Autor:

Christoph Wellner <wellner@stdcs.uni-sb.de>

8.2.2 Beschreibung der Konstruktoren und Destruktoren

8.2.2.1 NetworkUser::NetworkUser (string *name*, struct sockaddr_in *adr*)

Initialisiert das Objekt mit dem Namen des Benutzers im Netzwerk und dessen Adresse

Parameter:

name Name des Benutzers im Netzwerk

adr Netzwerk-Adresse des Benutzers

8.2.3 Dokumentation der Elementfunktionen

8.2.3.1 void NetworkUser::draw ()

Funktion zum Zeichnen des Benutzers

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- `NetworkUser.h`

8.3 NetworkUserIterator Klassenreferenz

```
#include <network.h>
```

Öffentliche Datenelemente

- `NetworkUserIterator (vector< NetworkUser *> network, int s)`
- `NetworkUser * nextNetworkUser ()`

8.3.1 Ausführliche Beschreibung

Autor:

Christoph Wellner <wellner@studcs.uni-sb.de>

Der NetzwerkUser-Iterator dient zum einfachen Zugriff auf die Buddy-Liste.

8.3.2 Beschreibung der Konstruktoren und Destruktoren

8.3.2.1 `NetworkUserIterator::NetworkUserIterator (vector< NetworkUser * > network, int s)`

Der Konstruktor liefert einen NetzwerkUser-Iterator, der auf die `buddyList` des angegebenen Netzwerk-Objekts zugreift.

Parameter:

network Das Netzwerk-Objekt, aus dem die User gelesen werden sollen
s anzahl der User, die in der `buddyList` von `network` stehen

8.3.3 Dokumentation der Elementfunktionen

8.3.3.1 `NetworkUser* NetworkUserIterator::nextNetworkUser ()`

Liefert den nächsten NetzwerkUser in der `buddyList`

Rückgabewerte:

NetworkUser * Zeiger auf den nächsten Netzwerk-User, sofern einer vorhanden ist
NULL wenn kein Netzwerk-User vorhanden ist, oder das Ende der Liste erreicht ist

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- `network.h`

8.4 ServerTCP Klassenreferenz

```
#include <serverTCP.h>
```

Öffentliche Datenelemente

- **ServerTCP** (int port)
- **~ServerTCP** ()
- int **send** (const char buffer[], struct sockaddr_in client)

Öffentliche Attribute

- string **name**

8.4.1 Ausführliche Beschreibung

Diese Klasse stellt einen TCP-Socket bereit, über den Nachrichten an Clients gesendet werden können.

Autor:

Christoph Wellner <wellner@studcs.uni-sb.de>

8.4.2 Beschreibung der Konstruktoren und Destruktoren

8.4.2.1 ServerTCP::ServerTCP (int *port*)

Initialisiert den TCP-Socket mit dem angegebenen Port

Parameter:

port der TCP-Port, über den gesendet wird

8.4.3 Dokumentation der Elementfunktionen

8.4.3.1 int ServerTCP::send (const char *buffer*[], struct sockaddr_in *client*)

Funktion zum Senden von TCP-Nachrichten über den Socket der Klasse. Jeder Nachricht wird der Netzwerk-Name vorangestellt, getrennt durch einen Doppelpunkt.

Parameter:

buffer Nachricht, die gesendet werden soll

client Adresse des Empfängers der Nachricht

8.4.4 Dokumentation der Datenelemente

8.4.4.1 `string ServerTCP::name`

< Der eigene Netzwerk-Name. Dieser wird jeder Nachricht vorangestellt und muß deshalb nach der Initialisierung angegeben werden.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- `serverTCP.h`

8.5 ServerUDP Klassenreferenz

```
#include <serverUDP.h>
```

Öffentliche Datenelemente

- `ServerUDP` (`int port`)
- `~ServerUDP` ()
- `int send` (`double *xpos`, `double *ypos`, `string name`, `struct sockaddr_in client`)

8.5.1 Ausführliche Beschreibung

Implementierung eines UDP-Sockets zum verschicken von Positions-Updates.

Autor:

Christoph Wellner <wellner@studcs.uni-sb.de>

8.5.2 Beschreibung der Konstruktoren und Destruktoren

8.5.2.1 `ServerUDP::ServerUDP (int port)`

Initialisiert den UDP-Socket mit dem angegebenen Port.

Parameter:

port Port für dem UDP-Socket

8.5.3 Dokumentation der Elementfunktionen

8.5.3.1 `int ServerUDP::send (double * xpos, double * ypos, string name, struct sockaddr_in client)`

Sendet die Position, die durch die x- und y-Variable angegeben ist, an die Adresse *client*. Damit die Gegenstelle die Nachricht korrekt auswerten kann, muß der eigene Netzwerk-Name über die Variable *name* mitgegeben werden

Parameter:

xpos x-Koordinate der zu versendenden Position

ypos y-Koordinate der zu versendenden Position

name der eigen Netzwerk-Name

client die Adresse, an die die UDP-Nachricht geschickt werden soll

Rückgabewerte:

0 bei erfolgreichem Senden

-1 bei Misserfolg

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- `serverUDP.h`

8.6 Parser Class Reference

```
#include <Parser.h>
```

Public Methods

- `Parser` (vector< string > Eingabe1)
- `Parser` ()
- `~Parser` ()
- void `parse` (string Eingabe2)
- bool `returnError_flg` ()
- void `setError_flg` (bool error1)
- bool `returnName_flg` ()
- void `setName_flg` (bool error2)
- void `setMes` (string name)
- string `getMes` ()

Public Attributes

- `ServerTCP* server`

8.6.1 Detailed Description

Die Klasse `Parser` dient zum einfachen Parsen der vom Benutzer geschriebenen Nachrichten. Der Parser kontrolliert dabei, daß die richtige Form der Nachrichten eingehalten wird. Dazu zählt zum Beispiel, daß jeder Nachricht das Schlüsselwort "send" voran gestellt wird. Falls diese Konventionen nicht eingehalten werden, gibt der Parser eine Fehlermeldung im Messagewindow aus. Danach wird der Name des Benutzers, an den die jeweilige Nachricht geschickt werden soll, geparkt. Zwei generelle Möglichkeiten gibt es hierbei, wie schon erwähnt wurde, zu unterscheiden. Wenn man an alle angemeldeten Benutzern eine Nachricht schreiben will benützt man das Schlüsselwort "all", falls nur ein spezifischer Benutzer erreicht werden will, benützt man den Benutzernamen der jeweiligen Person. Anschließend sucht der Parser die entsprechende Netzwerkadresse, den `Tcp-Port` und den `Udp-Port` in einer extra fuer diesen Zweck angelegten Datei der jeweiligen Person, die dann an die `send-Methode` mit dem Nachrichtenrumpf uebergeben wird.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- `Parser.h`

Index

- ~Network
 - Network, 27
- ~NetworkUser
 - NetworkUser, 31
- ~Parser
 - Parser, 35
- ~ServerTCP
 - ServerTCP, 33
- ~ServerUDP
 - ServerUDP, 34
- Anzeigefenster über die angemeldeten Benutzer (Buddy Window), 23
- Anzeigefenster von erstellten Nachrichten (Compose Message Window), 24
- Anzeigefenster von Nachrichten (Message Window), 24
- ARREAL, 7
- AutoCAD, 13
- buddyList
 - Network, 30
- draw
 - Network, 28
 - NetworkUser, 31
- draw(), 21
- Flatpanel-Display, 11
- Fußgängernavigationssystem, 7
- getBuffer
 - Network, 28
- getFlag
 - Network, 28
- getMes
 - Parser, 35
- getName
 - Network, 27
- Grafikkarte, 11
- Head-Mounted-Display, 11
- Informationsfenster, 23
- Installation, 10
- Kernelmodule, 12
- Klasse serverTCP, 25
- Konfigurationsdateien, 17
- main.cc, 20
- Mainwindow, 23
- Maus, 10
- maxsize
 - Network, 30
- msg
 - Network, 30
- name
 - ServerTCP, 34
- Network, 27
 - ~Network, 27
 - buddyList, 30
 - draw, 28
 - getBuffer, 28
 - getFlag, 28
 - getName, 27
 - maxsize, 30
 - msg, 30
 - Network, 28
 - newNetworkUserIterator, 29
 - send, 29
 - serverTcp, 28
 - setBuffer, 29

- start, 29
- stop, 29
- tcp_socket, 30
- udp_socket, 30
- NetworkObject, 18
- NetworkUser
 - ~NetworkUser, 31
 - NetworkUser, 31
- NetworkUser, 30
 - draw, 31
 - NetworkUser, 31
- NetworkUserIterator
 - NetworkUserIterator, 32
- NetworkUserIterator, 31
 - NetworkUserIterator, 32
 - nextNetworkUser, 32
- Netzwerkanbindung, 17
- NetzwerkObjekt, 17
- newNetworkUserIterator
 - Network, 29
- nextNetworkUser
 - NetworkUserIterator, 32
- OpenGL, 23
- parse
 - Parser, 35
- Parser, 35
 - ~Parser, 35
 - getMes, 35
 - parse, 35
 - Parser, 35
 - returnError_flg, 35
 - returnName_flg, 35
 - server, 36
 - setError_flg, 35
 - setMes, 35
 - setName_flg, 35
- Protokoll für das Senden von Nachrichten, 25
- Remote-Objekt, 19
- returnError_flg
 - Parser, 35
- returnName_flg
 - Parser, 35
- Schlüsselwörter, 24
- send
 - Network, 29
 - ServerTCP, 33
 - ServerUDP, 35
- send(), 21
- send-Funktion, UDP, 18
- Senden von Nachrichten, 24
- server
 - Parser, 36
- ServerObjectUDP, 17
- ServerObjektTCP, 17
- ServerTCP
 - ~ServerTCP, 33
 - ServerTCP, 33
- ServerTCP, 32
 - name, 34
 - send, 33
 - ServerTCP, 33
- serverTcp
 - Network, 28
- ServerUDP
 - ~ServerUDP, 34
 - ServerUDP, 34
- ServerUDP, 34
 - send, 35
 - ServerUDP, 34
- setBuffer
 - Network, 29
- setError_flg
 - Parser, 35
- setMes
 - Parser, 35
- setName_flg
 - Parser, 35
- Sound, 12
- Soundconfiguration, 12
- start
 - Network, 29
- start(), 19

stop

 Network, 29

stop(), 21

Subwindow, 23

SuSE Linux, 10

TCP-Protokoll, 17

tcp_socket

 Network, 30

Textnachrichten, 23

Treiber, 11

UDP-Protokoll, 17

udp_socket

 Network, 30

Visual_GL.cc, 20

Visual_GL.h, 20

VisualBasic, 13

WLAN, 12

XOF, 13, 15

XServer, 11

Xybernaut, 7

Abbildungsverzeichnis

2.1	Xybernaut samt Headset	9
3.1	AutoCAD-Plan der CeBit mit allen Details	14
3.2	AutoCAD-Plan der CeBit mit wenigen Details	14
3.3	Benutzeroberfläche	15
3.4	Darstellung eines rechteckigen Gebäudes in der XOF-Datei	16
3.5	Darstellung der Karte in ARREAL	17
4.1	Perspektivenwechsel	22
4.2	Kalibrierungsmodus	23
5.1	Kommunikation der Client's untereinander	28
5.2	Beispiel einer Netzwerk-Konfigurations-Datei	29
5.3	Beispiel einer BuddyList	29
5.4	UML-Klassendiagramm	33
6.1	Die Karte: Hier sieht man die Umgebung in der sich der Benutzer befindet (in diesem Fall, Gebäude der Universität der Saarlandes), sowie die drei Fenster die zur Kommunikation zwischen den Benutzern dient.	36
6.2	Protokoll für das Senden von Nachrichten	38

Literaturverzeichnis

- [1] Dateidefinition XOF : <http://w5.cs.uni-sb.de/aura/xof.html>
- [2] Homepage ARREAL : <http://w5.cs.uni-sb.de/arreal>
- [3] Homepage REAL-Projekt : <http://w5.cs.uni-sb.de/real.htm>
- [4] Xybernaut-Homepage : <http://www.xybernaut.de>
- [5] SuSE - Homepage: <http://www.suse.de>
- [6] SuSE-Patch-Script : <ftp://ftp.suse.com/pub/people/tiwai/7.3-i386/>
- [7] Wireless-HowTo <http://ldp.kernelnotes.de/HOWTO/Wireless-HOWTO.html>
- [8] CNN-Bericht zu Xybernaut <http://www.cnn.com/1999/TECH/computing/12/16/wearable.linux.i>
- [9] Eyeled <http://www.eyeled.de>
- [10] QT3-Library von Trolltech <http://www.trolltech.com>
- [11] SuSE Linux AG (2001) *SuSE Linux 7.3 Installationshandbuch*
- [12] Foley, vanDam, Feiner, Hughes (1996) *Computer Graphics, Principles and Practice*
Addison-Wesley

Abkürzungsverzeichnis

ADO	ActiveX Data Object
ALSA	Advanced Linux Sound Architecture
CAD	Computer Aided Design
FPD	Flat-panel Display
GPS	Global Positioning System
HMD	Head-mounted Display
ID	Identification-Number
IP	Internet Protocol
IR	Infrarot
LAN	Local Area Network
PC	Personal Computer
TCP	Transmission Control Protokoll
UDP	User Datagram Protocol
VB	VisualBasic
VBA	VisualBasic for Applications
Win	Microsoft Windows
WLAN	Wireless LAN
YAST	Yet another Setup Tool