SIEMENS

# SiMAP®-FIS

## Fahrgastinformationssystem

## Praxissemester-Bericht


Dipl.-Ing. Ralf Weiden

FH Köln

Fakultät für Informations-
Medien- und Elektrotechnik

Matr.-Nr. 11006584

Mail: mail*ralf-weiden.de (mit @ statt *)

Betreuer auf FH-Seite:

Prof.-Dr. Gregor Büchel,

Prof.-Dr. Georg Hartung,

FH Köln.



V1.1 24.02.2003 00:54h

# 1 Vorwort

Das diesem Praxissemester-Bericht zugrunde liegende IT-Projekt ist die Entwicklung des Fahrgastinformationssystems SiMAP®-FIS *(FIS)* bei der Siemens AG, Köln, Abteilung Information Technology Plant Solutions (I&S IT PS).

Die innerhalb dieses Projekts entstandene Applikations-Plattform wurde inzwischen zur namensgebenden *Siemens Modular Application Platform* (SiMAP®) ausgebaut.

Die Projektphasen Analyse, Spezifikation und Entwicklung des Basissystems wurden durchgeführt im Zeitraum Februar-November 2001. Das Projektteam bestand aus rund 10 festen Mitarbeitern:

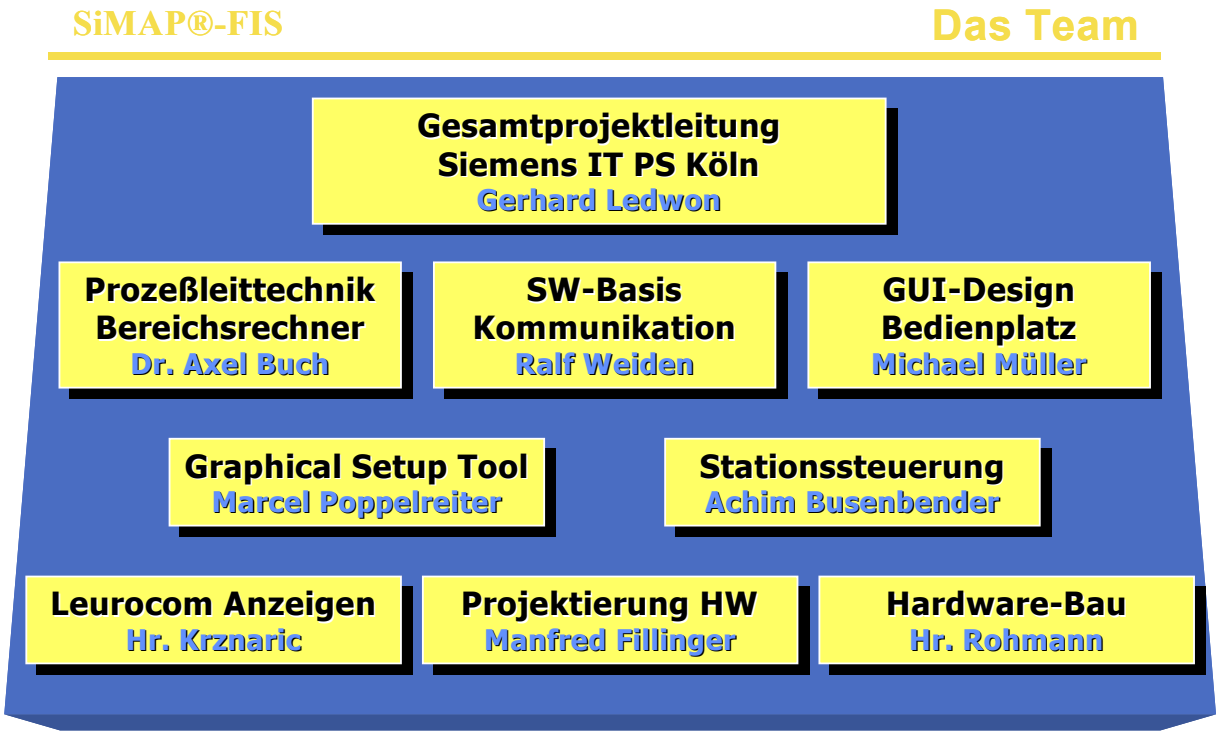



*Abbildung 1: Das Projektteam*

Die nachfolgenden Kapitel dieses Berichts geben eine Übersicht über das Systemkonzept und schildern insbesondere die von mir schwerpunktmäßig übernommenen Aufgaben in diesem Projekt. Dazu gehören:

- Analyse, Entwurf und Spezifikation des Gesamtsystems im Team mit meinen Kollegen Gerhard Ledwon und Axel Buch. Das resultierende Systemkonzept ist Thema der Kapitel 3-6 dieses Berichts.
- Entwurf und Implementierung des Software-Applikationsgerüsts (das „SiMAP®-Framework“) zur Bereitstellung und Kapselung der systemweit einheitlichen Basisfunktionalitäten als Windows-NT/2000-Anwendung. Dazu zählen Initialisierung, Laufzeit-Funktionalitäten wie Logging und Windows-spezifische Aufgaben (u.a. Fenster-Handling), sowie thread- und plattformübergreifende Kommunikation. Dieser von mir eigenverantwortlich durchgeführte Aufgabenkomplex ist Inhalt der Kapitel 7 und 8.
- Als Komponente des Kommunikations-Teilsystems entstand die Klasse *CTaggedString* (Kapitel 7.2) , die neben ihrer Aufgabe als „Telegramm-Container“ weitere Aufgaben wie Configfile-Handling übernimmt und damit eine wichtige Basisklasse innerhalb des SiMAP®-Systems darstellt.

Aufbauend auf „meinem“ Applikationsgerüst entstanden die eigentlichen SiMAP®-FIS-Software-Komponenten, insbesondere Bereichsrechner und Bedienplätze. Deren spezifische Anwendungs-Logiken wurden von meinen Kollegen (den „Benutzern“ des Frameworks) in das Framework eingebracht und sind daher nicht Bestandteil dieses Berichts.

Selbiges gilt für die Stationsrechner-Software meines Kollegen Achim Busenbender. Dessen Embedded-Linux-Anwendung ist eine auf die Belange dieser Plattform zugeschnittene Eigenentwicklung, die meine Klasse *CTaggedString* (Kapitel 7.2, „*Die Klasse CTaggedString*“) zum Telegrammaustausch mit den übrigen SiMAP®-Komponenten einbindet.

Getreu dem Motto „Klarheit vor Genauigkeit“ legt dieser Bericht den Schwerpunkt auf das Systemkonzept und die grundsätzlichen Vorgehensweisen und Techniken zur Realisierung. Die Implementierung mit C++ unter Benutzung der Microsoft Foundation Classes (MFC) für Windows koppelt die Software eng an das Windows-API bzw. die durch die MFC vorgegebene Struktur. Ein Eingehen auf die Besonderheiten dieser Umgebung würde den Blick auf das Wesentliche unnötig erschweren und den Rahmen eines solchen Berichts sprengen. Umfassende Informationen über fenster-basierte Windows-Applikationen allgemein sowie Multithreading und asynchrone Sockets im Speziellen findet der interessierte Leser in *[Kruglinski98]* und *[msdn].*

Köln, 24.02.2003 00:54h

Ralf Weiden

# Inhalt

# 2 Einleitung

Für die Akzeptanz eines Verkehrsunternehmens ist die dem Fahrgast gebotene Information von eminenter Bedeutung.

Ziel des Fahrgastinformationssystems SiMAP®-FIS ist die Bereitstellung von Technologie zur Information der Fahrgäste von Verkehrsunternehmen über Zugläufe, Ankunftszeiten und besondere Betriebssituationen.

Die Informationsdarstellung kann z.B. stationär über Anzeigen an den Bahnsteigen, im Internet oder mobil auf WAP-fähigen Endgeräten wie Mobile Phones oder PDA erfolgen.

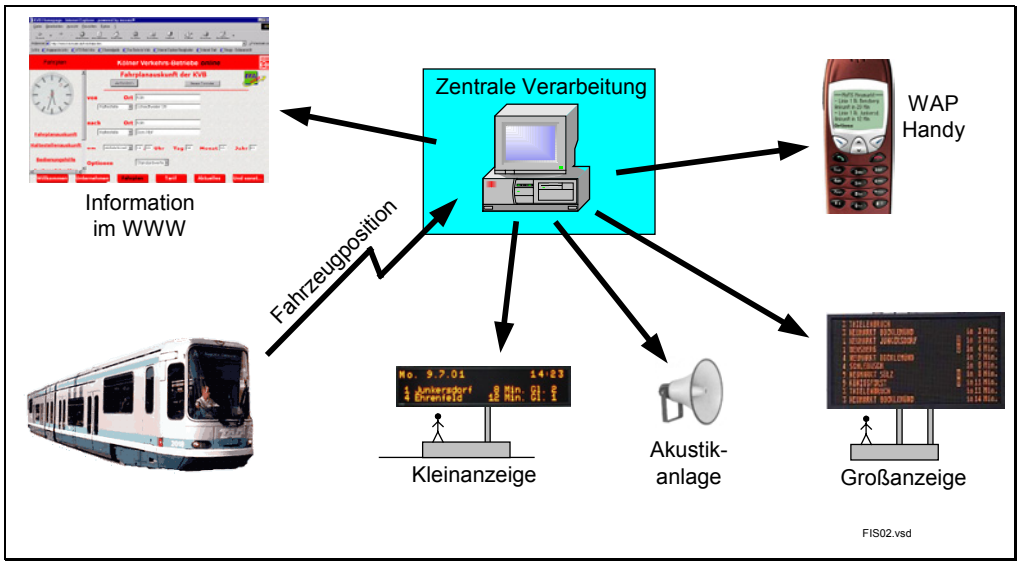



*Abbildung 2: Informationsangebot mit Simap®-FIS*


*Abbildung 3: Fahrgastinformationsanzeiger im Kompakt-Format*

Die Anzeigen informieren über die in nächster Zeit einfahrenden Züge inkl. Zugziel, Ankunftszeit und ggf. Einfahr-Gleis. Zusätzlich sind aktuelle Sondermeldungen oder Werbetexte darstellbar.

SiMAP®-FIS erlaubt die Ansteuerung von Anzeigen verschiedener Hersteller und Formate. Bei den stationären Anzeigen sind insbesondere Fahrgastinformationsanzeiger (FIA) im Kompakt-Format (2-4 Anzeige-Zeilen), sowie Großflächen-Anzeiger (GFA) zu nennen. Auf Großflächenanzeigern ist die Anzeigefläche beliebig aufteilbar. Dies ermöglicht beispielsweise die gleichzeitige Anzeige von Zugläufen verschiedener Bahnsteige oder Ebenen (unter-/oberirdisch) sowie Werbetexte.

# 3 Design-Vorgaben für SiMAP®-FIS

Beim Entwurf des FIS stand insbesondere eine hohe Ausfallsicherheit im Vordergrund. Weitere Schwerpunkte waren offene Schnittstellen für zukünftige Erweiterungen und zur Integration bestehender Systeme wie Stellwerk-Kopplungen oder rechnergesteuerte Betriebsleitsysteme (RBL). Grafikdarstellung sowie WAP/Internetschnittstelle sollten vorgesehen werden. Die folgende Auflistung fasst die zu erfüllenden Vorgaben zusammen:

- **Betriebssystem aller Rechner in der Zentrale: Windows NT/2000**
- **Betriebssystem Stationsrechner (SR) entlang des Streckennetzes: Embedded Linux**
- **Software-Erstellung mit C++, MFC**
- **Modulare Software mit hoher Wiederverwendbarkeit, nach Möglichkeit Rückgriff auf vorhandene Komponenten**
- **durchgehend TCP/IP**
  - etablierter Standard, integriert sowohl in Windows als auch Linux
  - ermöglicht Anwendung von Standard-Tools für Netzüberwachung, Fernwartung und Fern-Updates: Telnet, FTP, PING usw.
  - große Datenübertragungsrate zwischen den Rechnern (100 Mbit/sek mit LWL) (z.B. Grafikdarstellung auf GFAs / Soundausgabe möglich)
  - einfache browserbasierte Diagnose (HTTP Server Einsatz für Logging-Auswertung)
- **Offene Schnittstellen zur Integration von zukünftigen Systemen bzw. Fremdsystemen**
  - XML als Mittel des Datenaustauschs
  - Standard-Telegramme als Klartext lesbar, damit leicht debugbar
  - Schnittstellen modular realisiert, flexibel kombinierbar
- **Keine Änderungen an evtl. bestehenden Fremdsystemen**
- **Einsatz beliebiger Schildertypen**
  - kundenspezifischer Anzeigenanschluß möglich -> flexibles Anzeigentreiber-Konzept
- **Unterschiedliche Datenquellen für die Fahrzeugpositionen möglich**
  - Infrarotbaken entlang der Strecke
  - Koppelspulen im Gleis
  - Rechnergesteuertes Betriebsleitsystem (RBL)
  - Rechnergestützte Zug-Überwachung (RZÜ)
  - Funk / GPS
  - Stellwerkdaten

# 4 Systemstruktur - FIS als dezentrales System

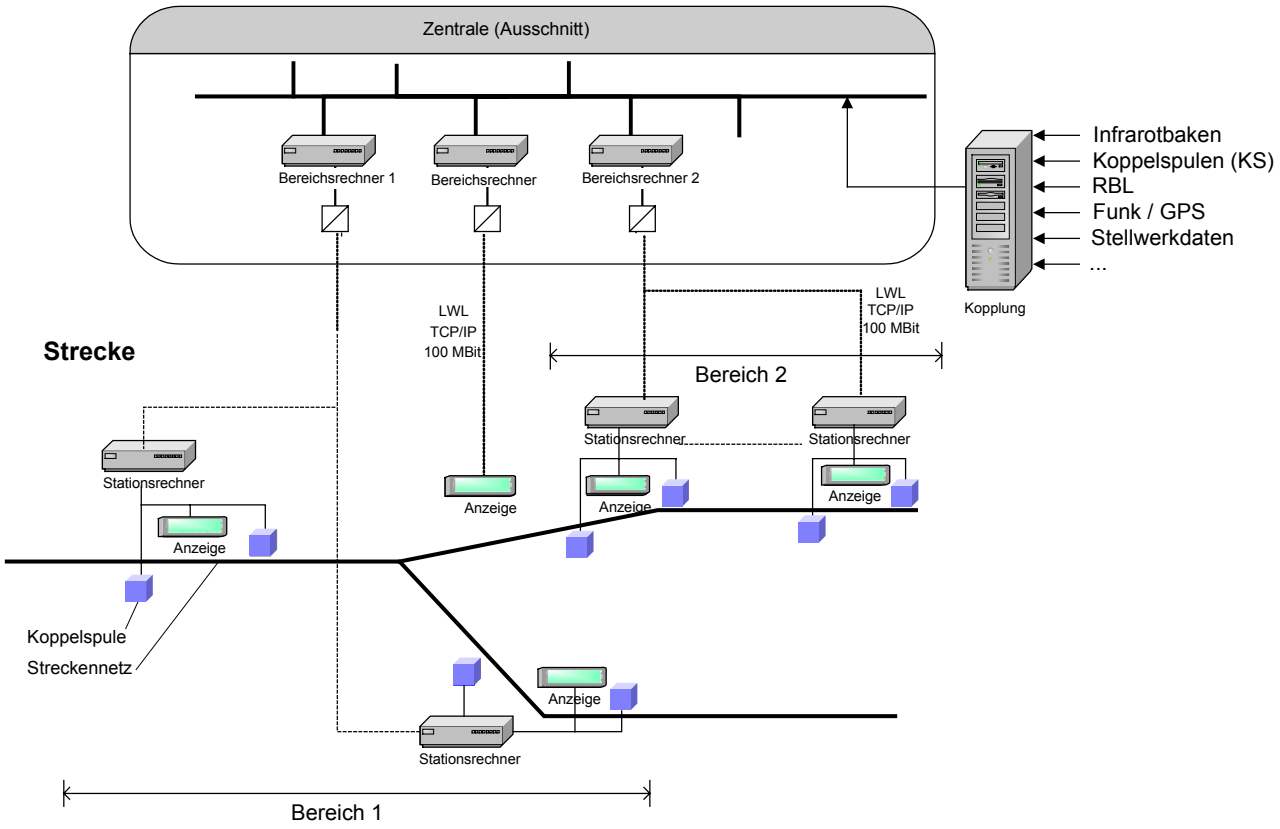



*Abbildung 4: FIS-Systemstruktur*

SiMAP®-FIS ist hierarchisch strukturiert und dezentral realisiert, d.h. es gibt keinen zentralen Server, über den alle Abläufe abgewickelt werden und bei dessen Ausfall das Gesamtsystem stünde.

Stattdessen unterteilt FIS das Streckennetz in mehrere logische Bereiche. Pro Bereich ist jeweils ein separater Bereichsrechner (BR) in der Zentrale vorgesehen.

Innerhalb eines Bereiches können mehrere Stationsrechner (SR) entlang der Strecke installiert sein. Stationsrechner bedienen Lesestellen wie Koppelspulen (KS) oder Infrarotbaken sowie Anzeigen im Stations- bzw. Streckenbereich. Für die Stationsrechner eines Bereiches ist jeweils der zugeordnete Bereichsrechner in der Zentrale verantwortlich. Bereits vorhandene Systeme, wie rechnergesteuerte Betriebsleitsysteme (RBL), Stellwerk-Rechner etc. können zur Fahrzeug-Positionsbestimmung über spezielle Koppelrechner eingebunden werden.

Bei Ausfall eines Bereichsrechners bleibt diese Störung auf den zugeordneten Bereich beschränkt, das Gesamtsystem bleibt weiter funktionsfähig. Analoges gilt für den Ausfall einzelner Stationsrechner.

In der Zentrale kann ein Backup-Rechner vorgesehen werden, der den Bereich eines ausgefallenen Bereichsrechners durch manuellen Eingriff übernehmen kann. Dazu ist der Backup-Rechner mit der zugehörigen IP-Adresse zu parametrieren, ggf. die Netzwerkverbindung zur Feldebene umzuschwenken und die (auf allen Bereichsrechnern identische) Software mit der Kennung des zu übernehmenden Bereiches (etwa „BR41“) zu starten. Daraufhin wird sich der neue Rechner selbsttätig im System anmelden.

# 5 IT-Komponenten des FIS

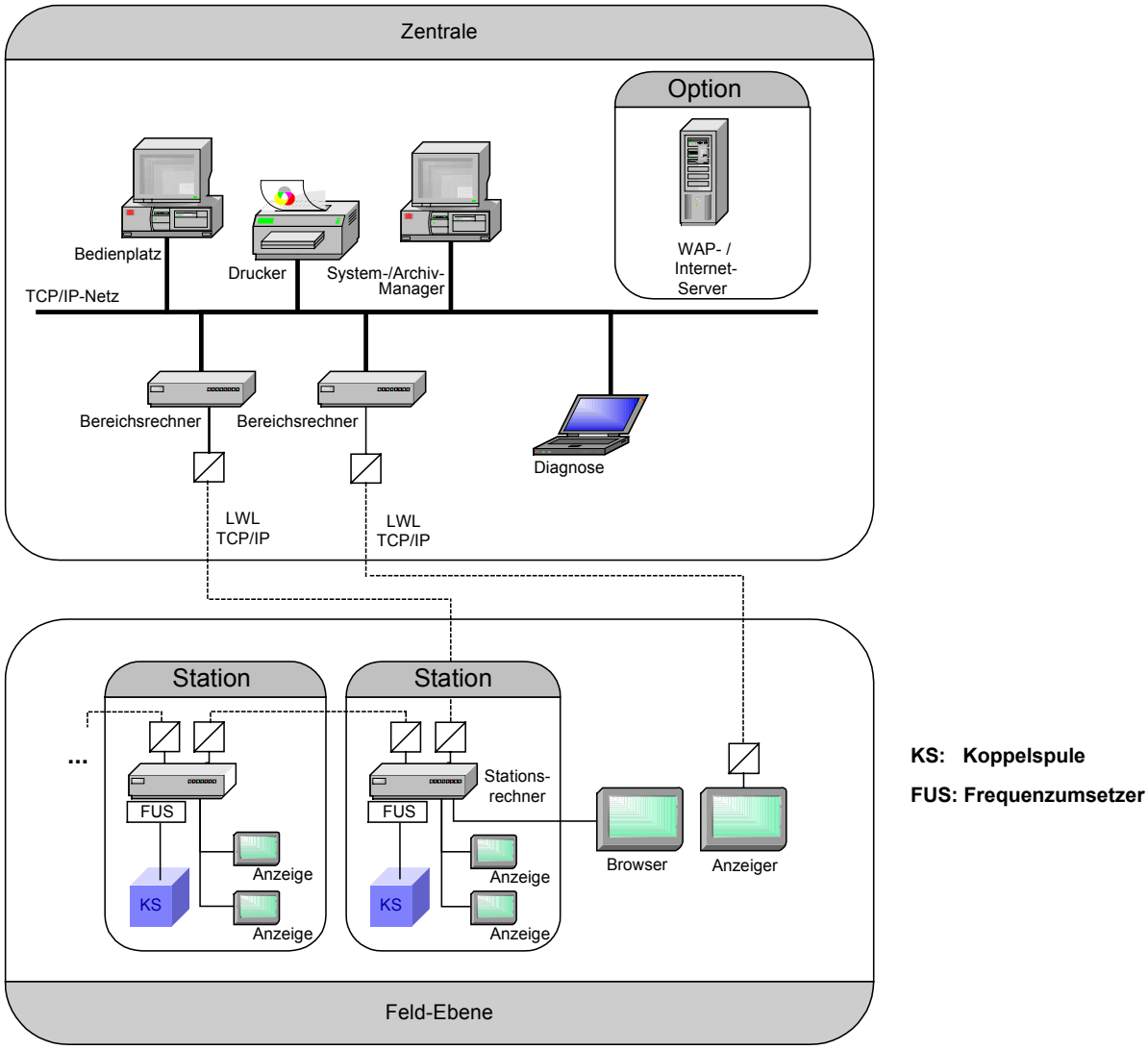



*Abbildung 5: IT-Komponenten des FIS*

| Komponente | Funktion |
|---|---|
| Bedienplatz BPL | Bedienen und Beobachten des Gesamtsystems<br>Manuelle Anzeigen-Steuerung und Anzeigen-Parametierung |
| System-/Archiv-manager | zentrale Archivierung von Betriebs-/Störmeldungen<br>zentrale Datenhaltung der Parametrierung aller Systemkomponenten |
| Bereichsrechner BR | Auswerten/Steuern aller Fahrzeugpositionen/Anzeigen im Bereich |
| Stationsrechner SR | Auswerten/Ansteuerung lokaler Meldestellen/Anzeigen im Streckennetz |
| Diagnose | Browser-basierter Zugriff auf Systeminformationen und Logdateien (z.B. Status von Bereichsrechnern, Auswertung von Meldestellen-Zuständen) |
| WAP-/Internet-Server | Bereitstellen von Informationen des FIS im Internet oder auf mobilen Endgeräten |

*Tabelle 1: Liste der IT-Komponenten*

# 5.1 Bedienplätze (BPL)

Die Bedienrechner stellen die Schnittstelle des FIS zum Bedienpersonal dar. Über die Bedienrechner kann das gesamte System beobachtet und gesteuert werden. Der Bedienrechner visualisiert ereignisgesteuert die Anzeigen der Schilder sowie ggf. in einem separaten Meldungsfenster anliegende Betriebsstörungen im System.

Jeder beliebige PC, der mit dem TCP/IP-Netz in der Zentrale verbunden ist, kann nach dem Aufspielen der Bedienplatzapplikation als Bedienrechner genutzt werden. Durch eine Zugriffsverwaltung kann die Funktionalität bestimmter Bedienrechner eingeschränkt werden (z.B. nur Beobachten).

Auf dem Bildschirm erscheint das Bild des Streckennetzes, auf dem Streckennetz sind Objekte platziert. Die Objekte repräsentieren die Außenkomponenten des Systems (Außenanzeigen, Großanzeigen, Lesestellen etc.). Mit einem Zoom-Werkzeug ist das Visualisieren ausgewählter Ausschnitte in verschiedenen Detailliertheitsstufen von der Gesamtübersicht bis zur Detailansicht z.B. einzelner Anzeigen möglich.

Die Ereignissteuerung der Anzeigen erfolgt durch ein Publish/Subscribe-Verfahren (siehe [Gamma96] ): Alle aufgeblendeten Anzeigen werden beim zugehörigen Bereichsrechner angemeldet. Dieser wird daraufhin die Bedienplätze mit den angemeldeten Anzeigen per Telegramm über jede Änderung informieren. Die Update-Zeit ist < 1 Sekunde.

Die Bedienoberfläche kann weitgehend vom Kunden selber parametriert werden. Sowohl die Darstellung des Streckennetzes, der Anzeigen als auch die Lage und Anzahl der Objekte wird ohne Softwareänderung konfiguriert.

Die Bedienrechner sind Standard-PCs unter Windows NT/2000.

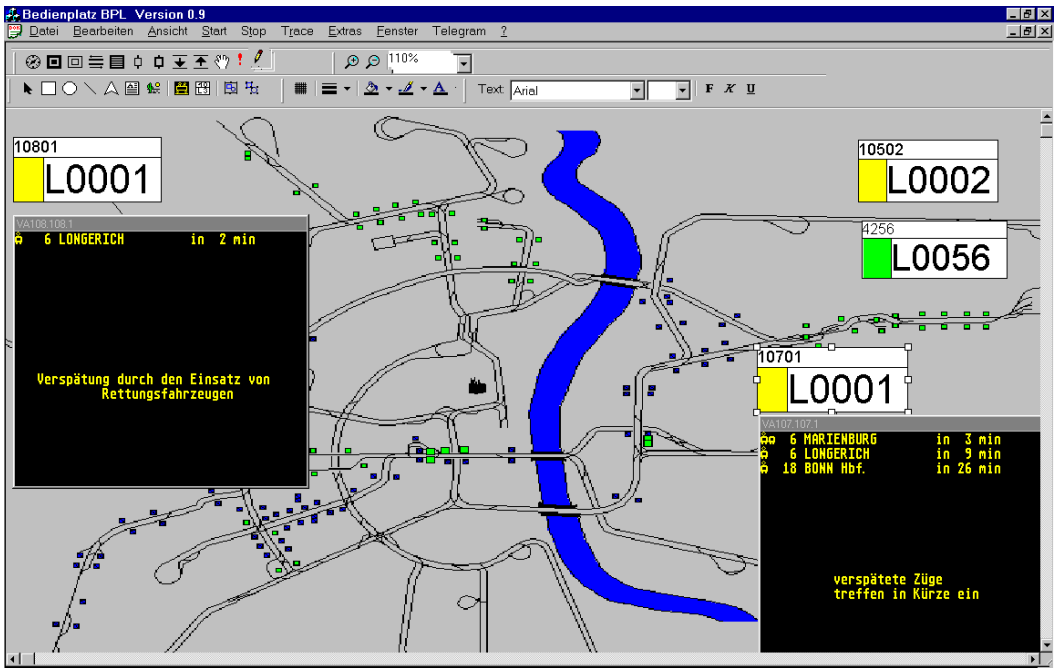



*Abbildung 6: Die grafische Oberfläche des FIS-Bedienplatzes*

## 5.2 System / Archivmanager (SM)

Der Systemmanager erfüllt im Wesentlichen zwei Aufgaben:

- Bereithalten der Konfigurationsdateien für alle Bereichsrechner sowie Bedienplätze
- Archivieren von Betriebs- und Störmeldungen

Auf dem Rechner liegen die Konfigurationsdateien für die Masken des Bediensystems. Außerdem werden hier die Konfigurationen für alle Bereichsrechner zentral gespeichert.

Nach dem Hochlauf eines Bereichsrechners oder Bedienplatzes werden Parametrierungen vom Systemmanager auf die entsprechenden Rechner übertragen. Bei einem Ausfall des Systemmanagers bzw. des Netzwerks arbeiten die Bereichsrechner und Bedienplätze mit lokalen Kopien der Konfigurationsfiles, so das auch in diesem Fall der Betrieb des FIS-Systems möglich ist.

Alle Systemkomponenten (Bereichsrechner, Bedienplätze) können zu archivierende Betriebs- und Störmeldungen an den Systemmanager senden, die dort zentral abgelegt werden.

Die Meldungsablage geschieht mindestens in Form von Dateien und wahlweise Ausgabe auf Druckern. Optional ist die Ablage in Melde- und Archivsystemen wie WinCC möglich. Im einfachsten Fall arbeitet der Systemmanager als Fileserver ohne spezifische Software.

Zusätzlich zu der Übertragung an den Systemmanager speichert jede Systemkomponente anfallende Betriebs- und Störmeldungen in lokalen Logfiles.

Der Systemmanager ist ein Industrie-PC mit dem Betriebssystem Windows2000.

## 5.3 Bereichsrechner (BR)

Ein Bereichsrechner hat folgende Funktionen:

- Empfang und Auswerten von Informationen über Züge im eigenen Bereich und anderen Bereichen
- Verteilen an andere Komponenten, i.d.R. Bereichsrechner und Bedienplätze
- Generierung von Anzeigetexten sowie Ansteuerung von Anzeigen und der Bedienplätze (abhängig von Lesestellensignalen, Fahrplan, externen Quellen)
- Archivierung von Betriebs-/Störmeldungen auf Archivserver
- Auswertung von Bedienereingaben
- Fahrplanüberwachung (an Endhaltestellen)
- Überwachung angeschlossener Komponenten

Zentralen-intern werden Informationen via TCP/IP über das Zentralen-LAN übertragen. Die Kopplung mit externen Komponenten wie direkt angesteuerten Großflächenanzeigern (GFA) oder Stationsrechnern kann sowohl über LWL-Switches als auch über 2-Draht-Kupferleitung erfolgen. In letzterem Fall übernimmt ein Async-Server das transparente Umsetzen/Multiplexing zwischen Ethernet und RS232 für bis zu 8 angeschlossene Modem-Strecken.

Der Bereichsrechner ist ein Industrie-PC unter Windows NT/2000.

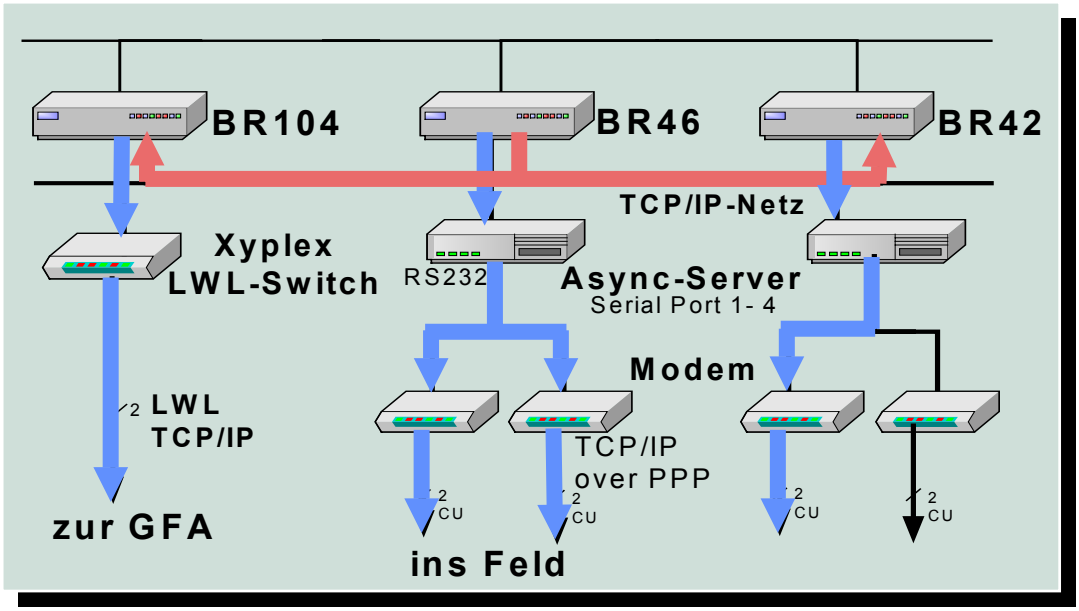



*Abbildung 7: Informationsflüsse in der Zentrale*

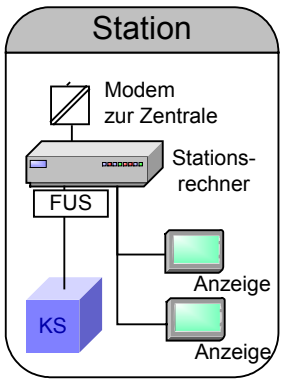
## 5.4 Stationsrechner (SR)

Die Stationsrechner sind entlang der Strecke installiert.

Stationsrechner haben folgende Funktionen:

- Empfangen und Weiterleiten der Informationen von Lesestellen
- Ansteuerung von Anzeigen

Sind im Bereich eines Stationsrechners mehrere Lesestellen installiert, kann ein Stationsrechner die Information mehrerer Lesestellen auswerten. Fahrzeug-Informationen aus Koppelspulen (KS) werden durch einen Frequenzumsetzer (FUS) aus dem analogen HF-Koppelspulensignal in ein serielles Digitalsignal umgewandelt. Der Stationsrechner erzeugt daraus ein Lesestellen-Telegramm und leitet es an „seinen“ Bereichsrechner weiter.




Züge werden über eine Linien- und eine Kurs-Nummer (jeweils 0..99) identifiziert. Meist reicht die Linien-Nr. aus, um sowohl die Linie, als auch das Zugziel zu identifizieren.
Über die Kursnummer kann eine Zuordnung z.B. zu Fahrplan-Informationen erfolgen. Dies ist notwendig, wenn manche Linien tageszeitabhängig nicht die gesamte Strecke befahren, sondern verschiedene Endhaltestellen ansteuern. Die Kombination aus "Linie" + "Zugnummer" ist im FIS-System eindeutig.

Die Logik zur Ansteuerung angeschlossener Kleinanzeigen liegt komplett in den Bereichsrechnern. Die Aufgabe des Stationsrechners besteht nur darin, die via TCP/IP eintreffenden Anzeige-Telegramme über die serielle Schnittstelle an die richtige Anzeige weiterzureichen.

Stationsrechner sind PC/104 Einplatinen-Computer mit Intel Pentium-CPU. Betriebssystem ist Embedded Linux. Entsprechend liegt der Software nicht das SiMAP®-Framework zugrunde, sondern eine spezielle Embedded-Linux-Entwicklung.

# 6 Telegramm-Formate

## 6.1 Einleitung

Ein wesentliches Merkmal von SiMAP®-FIS ist der umfangreiche rechnerübergreifende Datenverkehr. Dabei sind Daten sowohl zwischen Windows- als auch zwischen Linux-Rechnern (z.B. den Stationsrechnern) auszutauschen.

Früh in der Entwurfsphase fiel die Entscheidung, alle auszutauschenden Telegramme in Klartext-lesbarer Form zu spezifizieren. Das Format der von uns definierten Telegramme folgt dem bekannten XML-Stil mit jeweils durch ein Start- und Endtag eingerahmten Daten.

Die Klartext-Lesbarkeit erleichtert die Nachvollziehbarkeit ausgetauschter Informationen sowohl während der Entwicklungsphase als auch im laufenden Betrieb. Ferner ist durch den festgelegten Telegramm-Aufbau die einheitliche Generierung und Auswertung zu sendender bzw. empfangener Telegramme möglich. Dies und die einfache Einführung neuer Telegramme wird durch die Klasse „CTaggedString“ und ihrer Unterklassen (siehe „7.2 Die Klasse *CTaggedString*“, Seite 15) erreicht, über welche Telegramme generiert und ausgewertet werden. Weitere Informationen über das XML-Format finden sich in *[selfxml]*, *[w3c]* und *[Nussbaumer02]*.

## 6.2 Telegrammstrukturen

### 6.2.1 Der Telegramm-Header

Telegramme bestehen immer aus einem Header und den eigentlichen Telegramm-Daten. Das Datenformat des Headers wurde wie folgt definiert:

| Bedeutung | Beispiel | Bemerkung |
|---|---|---|
| Telegramm-Start | `<TELEGRAM>` | |
| Header-Start | `  <HEAD>` | |
| Telegramm-Länge | `    <LEN>nnnnnnnnnn</LEN>` | Über-alles-Länge (immer 10 Ziffern) (derzeit nicht ausgewertet) |
| Telegramm-Typ | `<TEL_TYPE>`*`tel.type`*`</TEL_TYPE>` | Typ des Telegramms zum Dispatching in der Empfangsroutine |
| Absender | `    <SENDER>SM01</SENDER>` | |
| Empfänger | `    <RECEIVER>BR31</RECEIVER>` | |
| Header-Ende | `  </HEAD>` | |
| Tel. Body Start | `  <BODY>` | |
| Telegramm-Body | `    :` | Telegramm-Rumpf wie unten |
| Tel. Body Ende | `  </BODY>` | |
| Telegramm-Ende | `</TELEGRAM>` | |

*Tabelle 2: Der Telegramm-Header*

## 6.2.2 Der Telegramm-Rumpf

Die eigentlichen Telegramm-Daten sind innerhalb des <BODY> Tags eingeschlossen. Es können mehrere Datensätze gleichen Typs innerhalb des Bodys hintereinander folgen. Sie sind jeweils in ein <*tel.type*> Tag eingeschlossen, wobei „*tel.type*“ mit dem im Header angegebenen Typ übereinstimmen muss. Dies ist z.B. sinnvoll bei der Übertragung von Koppelspulen-Zuständen: Ein Stationsrechner kann den Status aller „seiner“ Koppelspulen in einem einzigen Telegramm an den Bereichsrechner melden.

| Bedeutung | Beispiel | Bemerkung |
|---|---|---|
| Tel. Inhalt Start | <BODY> | |
| Telegramm-Daten | {<*tel.type*><br>:<br></*tel.type*>} | Dateninhalt des Telegramms, abhängig vom Telegramm-Typ. Der Abschnitt kann ggf. mehrfach wiederholt werden |
| Tel. Inhalt Ende | </BODY> | |

*Tabelle 3: Der Telegramm-Rumpf*

## 6.2.3 Telegramm-Beispiel: Status-Telegramm

Exemplarisch sei der Telegramm-Aufbau am Status-Telegramm (Telegramm-Typ: *SEND_STATUS*) erläutert. Dieses Telegramm wird von Stationsrechnern zur Zustandsübermittlung angeschlossener Komponenten an den Bereichsrechner benutzt. Der Bereichsrechner seinerseits meldet mit dem gleichen Telegramm Zustände zur Anzeige an die Bedienplätze weiter.

Das Status-Telegramm hat insgesamt 4 optionale „INFO“Parameter. Zur Zeit wird nur vom SR der erste optionale Parameter benutzt um die aktuelle SW-Version zu übertragen.

Beispiel:

```
<TELEGRAM>
   <HEAD>
      <LEN>000000002644</LEN>
      <SENDER>SR42.2</SENDER>
      <RECEIVER>BR42</RECEIVER>
      <TELTYPE>SEND_STATUS</TELTYPE>
   </HEAD>
   <BODY>
     <SEND_STATUS>                  // beliebig viele SEND_STATUS-Abschnitte
       <NAME>LST42.12</NAME>        // Status zu Lesestelle LST42.12
       <STATUS>error</STATUS>
       <INFO1>SR Version 0.9.0.2</INFO1>
     </SEND_STATUS>
     <SEND_STATUS>
        <NAME>LST42.14</NAME>       // Status zu Lesestelle LST42.14
        <STATUS>error</STATUS>
        <INFO1>SR Version 0.9.0.2</INFO1>
     </SEND_STATUS>
  </BODY>
</TELEGRAM>
```

# 7 Die Software-Basis: Das SiMAP®-Framework

## 7.1 Allgemeines

In IT-Projekten wiederholen sich vielerlei Aufgaben, wie Logging, Datenaustausch, System-Anlauf und –stopp immer wieder. Daher liegt es nahe, diese Basis-Aufgaben in Modulen zu kapseln und für die spätere Wiederverwendung vorzusehen. Dies trifft insbesondere auch auf die Bestandteile des FIS-Systems zu: Alle Komponenten, wie Bereichsrechner, Bedienplätze, Archiv-System etc. müssen in einheitlicher Weise miteinander kommunizieren und sollten eine in weiten Teilen einheitliche Software-Basis besitzen.

Die Windows-Komponenten des SiMAP®-FIS basieren im Kern auf einer vorab bereits mehrfach benutzten Entwicklungs-Plattform der Siemens AG Köln zur Lösung von Integrations-Aufgaben in der Industrie. Diese Plattform wurde innerhalb des FIS-Projekts zur *Siemens Modular Application Platform (SiMAP®)* ausgebaut. Inzwischen dient SiMAP® unter eigenständigem Markennamen als Basis zur Realisierung integrationsfähiger kundenspezifischer Modularsoftware zur Lösung mannigfaltiger Kommunikationsaufgaben (siehe www.simap.de).

SiMAP® ist anzusehen als Framework, welches die Initialisierung und Darstellung des Haupt-Applikationsfensters, sowie Log- und diverse Hilfsfunktionen übernimmt. In dieses Framework können mittels bereitgestellter Steuer- und Schnittstellenfunktionen verschiedene, anwendungsspezifische Module (Kommunikationsobjekte, kurz ComObjekte) eingebracht werden. Vom SiMAP®-Framework werden diese Module zu einem Gesamtsystem zusammengefügt und koordiniert.

Die einzelnen Module übernehmen Kommunikations- und Steuerungsaufgaben, wie:

- Kommunikation via TCP/IP- und Novell-IPX-Sockets
- Grafische Benutzerschnittstelle
- Logik/Steuerungsmodul
- Weitere Aufgaben, die im FIS-spezifischen Basissystem nicht benutzt werden

Die SiMAP®-Module kommunizieren untereinander asynchron über Messages des Windows-Betriebssystems. Jedes Modul läuft in einem eigenen Thread und besitzt ein eigenes Fenster zur Meldungsausgabe und Interaktion mit dem Bediener.

Die Verarbeitungs- und Steuerlogik der zu erstellenden Applikation wird in einem spezialisierten Modul zentralisiert. Dadurch erfolgt die Verarbeitung entkoppelt von den Kommunikationsmodulen. Dies vermeidet ein Blockieren der Verarbeitung bei stockender Kommunikation.

Als Basis zur Entwicklung eigener Module dient die SiMAP®-Klassenbibliothek. Hierin enthalten sind Klassen, die die Basisfunktionalitäten eines Kommunikationsobjektes kapseln und verschiedene, immer wieder vorkommende Funktionalitäten bereitstellen.

Für das Projekt SiMAP®-FIS wurde ein FIS-spezifisches Standard-Framework mit folgenden Modulen entwickelt:

- TCP/IP-Server zum Empfang von Telegrammen im XML-Format (Kap. 8.4)
- TCP/IP-Client zum Versenden dieser Telegramme (Kap. 8.3)
- Ein im Hintergrund laufender TCP/IP-Client zur Netzüberwachung
- Ein Steuerungs-Objekt zur Einbringung der jeweiligen Applikationslogik (Kap. 8.5)

Alle Windows-Applikationen des FIS-Systems basieren auf diesem Standard-Framework.

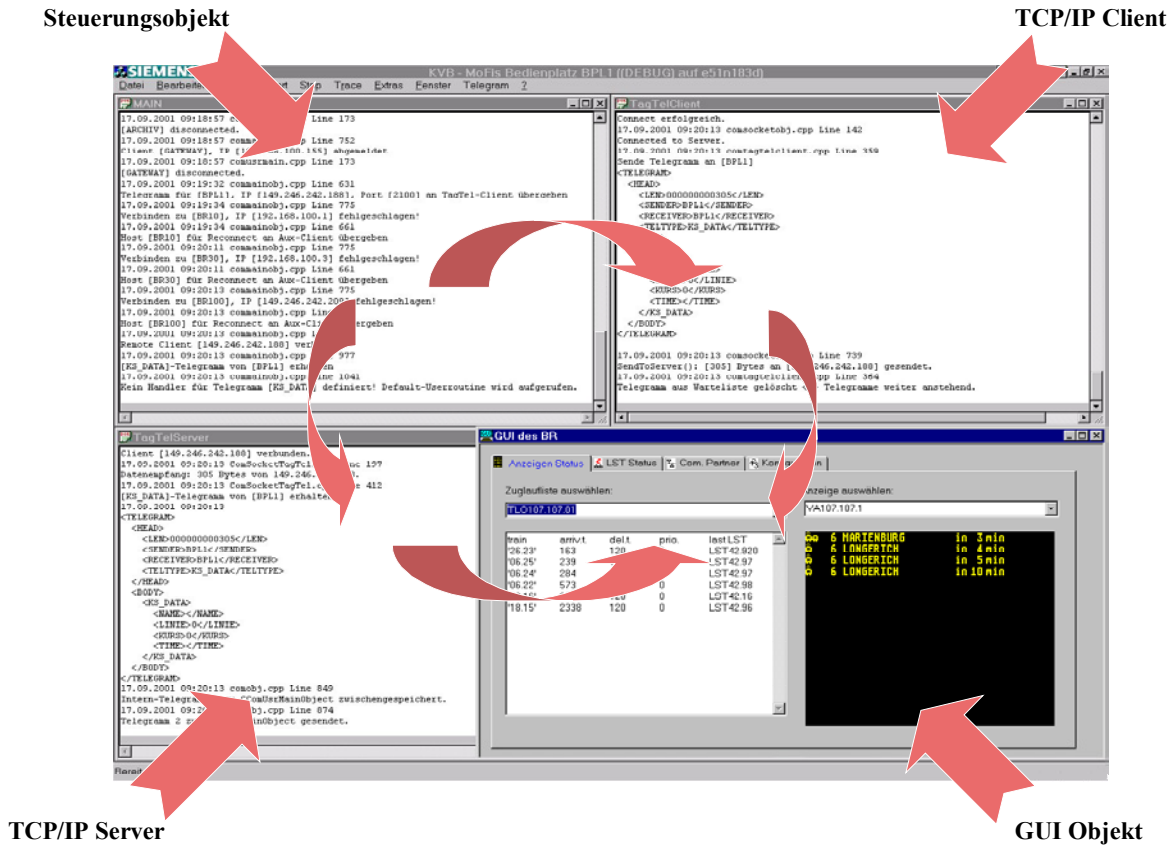



*Abbildung 8: Oberfläche des SiMAP-Frameworks (Beispiel: Bereichsrechner)*

# 7.2 Die Klasse *CTaggedString*

Durch die Entwicklung der Klasse *CTaggedString* stehen Basis-Funktionalitäten zur Handhabung von XML-artigen, „getaggten“ Strings zur Verfügung.

Beim System-Entwurf wurde bewusst auf die Verwendung existierender Bibliotheken bzw. DLLs zum XML-Handling von Drittanbietern verzichtet. Hauptgrund hierfür ist die notwendige Portierbarkeit zwischen Windows- und Linux-Systemen sowie die sehr spezielle Benutzung als Telegramm-Container und weiterer Aufgaben innerhalb des SiMAP®-Systems. Infolgedessen wurden auch nicht alle im XML-Standard definierten Strukturmöglichkeiten (siehe *[selfxml]*) realisiert. Insbesondere ist die Verwendung von separaten DTD-Dateien (Data Type Description-Dateien) zur automatischen Korrektheitsüberprüfung durch einen XML-Parser nicht vorgesehen.

Als „getaggter String” wird in der SiMAP®-Nomenklatur ein String verstanden, welcher jede Art von benutzerdefinierbaren Daten im ASCII-Format innerhalb eines Start-Tags und eines End-Tags einschließt:

```
<TAGNAME>freely defineable user data</TAGNAME>.
```

Wie in XML ist das Schachteln von Tags und damit das Bilden von Gruppen innerhalb sogenannter „Sections“ möglich:

```
<SECTION1>
      <TAG1>data1</TAG1>
      <TAG2>data2</TAG2>
</SECTION1>
<ITERATIONS_SECTION>
  <SECTION2>
    <TAG1>data1</TAG1>
  </SECTION2>
  <SECTION2>
    <TAG1>data2</TAG1>
  </SECTION2>
  <SECTION2>
    <TAG1>data3</TAG1>
  </SECTION2>
</ITERATIONS_SECTION>
```

Die Klasse *CTaggedString* bietet Memberfunktionen für die Erstellung von Sections (*BeginSection(), EndSection()*), Einfügen von „getaggten“ Daten (*AddTagString()*) und, umgekehrt, das Auslesen von Daten aus spezifizierten Tags in spezifizierten Sections (*GetProfileString()*).

Ferner ist das Iterieren über eine Abfolge von mehreren gleichartigen Sections möglich. Dies ermöglicht die *CTaggedString*-Methode *SetSection(),* die als Übergabe den Namen der zu iterierenden Section (oben etwa „SECTION2“) sowie ggf. einen Startindex für die Suche nach dem Section-Starttag erwartet. Rückgabewert ist der Index des ersten gefundenen Section-Starttags ab der Startindex-Position. Dieser Rückgabewert kann bei weiteren Aufrufen als neuer Such-Startindex benutzt werden.
Nach *SetSection()* findet der Aufruf der Datensatz-Abfragemethode *GetProfileString()* nur noch Daten innerhalb der festgesetzten Section, bis mit *ReleaseSection()* die zuletzt gesetzte Section wieder freigegeben wird. Ein Schachteln verschiedener Sections ist bis zu einer Tiefe von 20 Sections möglich.

Das folgende Klassendiagramm gibt einen Überblick über *CTaggedString*. Die Klasse ist Basis weiterer von ihr abgeleiteter Klassen. Zu den abgeleiteten Klassen gehören Klassen wie *CConfigFile* zum Zugriff auf Konfigurationsdateien mit XML-artigem Aufbau sowie die Klasse *CTelegram* als abstrakte Basisklasse für alle von ihr abgeleiteten User-Telegramm-Klassen des FIS-Systems.

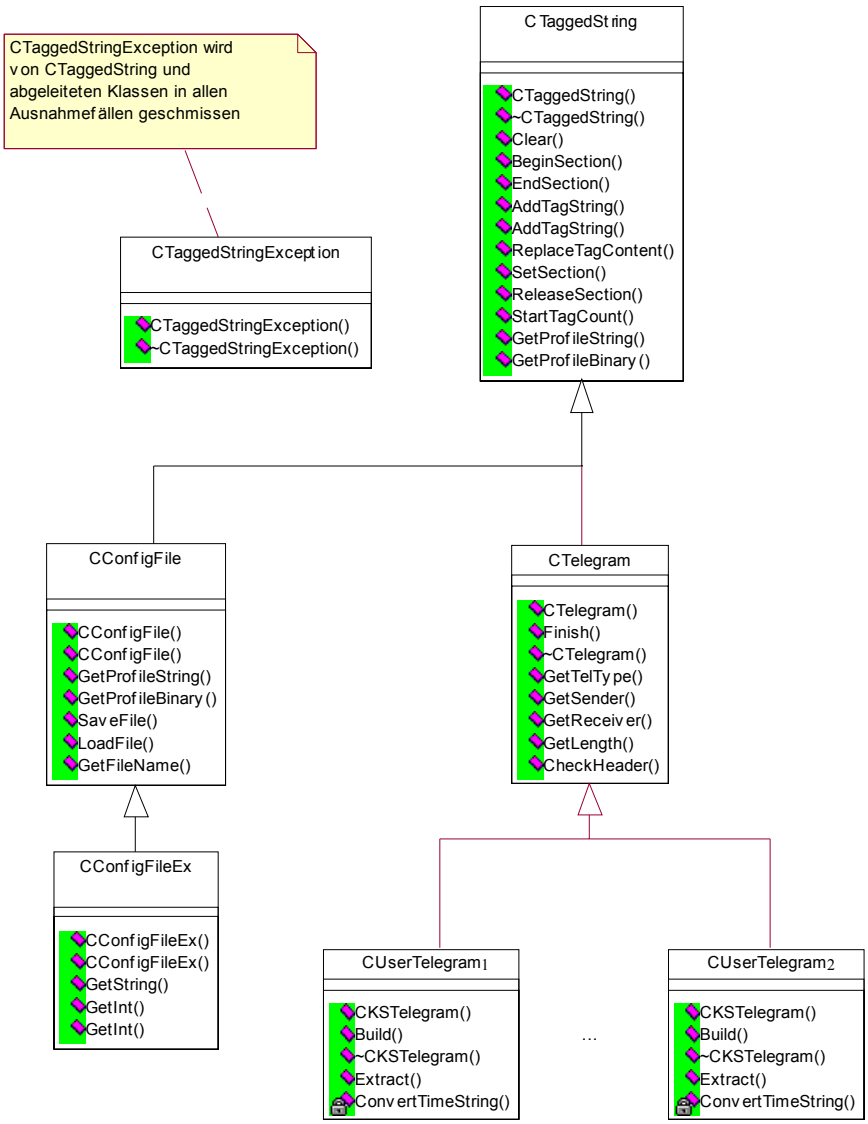


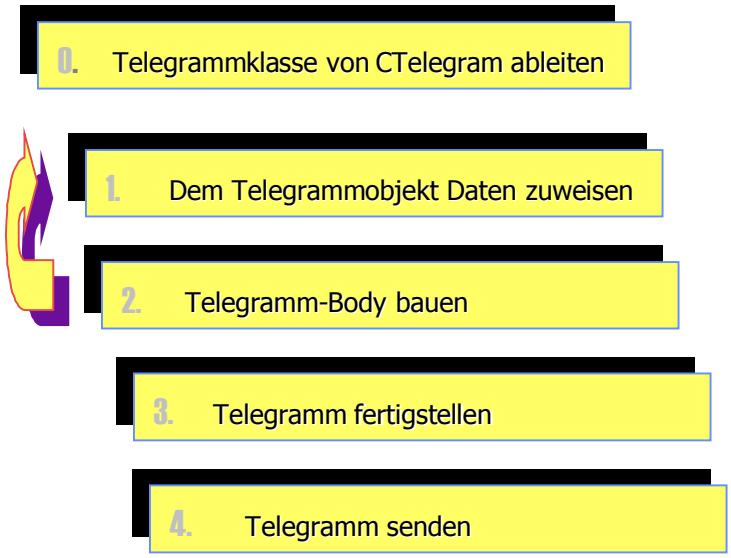

*Abbildung 9: Klassendiagramm von CTaggedString mit abgeleiteten Klassen*

# 7.3 Telegramm-Handling mit *CTelegram*

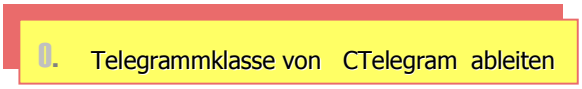
Das Erstellen von Datentelegrammen im XML-Format wird durch die Telegrammklasse *CTelegram* abgewickelt. Der Benutzer kann von dieser Basisklasse eigene Telegrammklassen ableiten und Objekte davon als Container zur Daten-Übertragung an das SiMAP®-Framework übergeben bzw. von diesem entgegennehmen. Das Framework übernimmt das Versenden an den Telegramm-Empfänger sowie umgekehrt den Aufruf der zu eingehenden Telegrammen definierten Behandlungsfunktionen.

## 7.3.1 Senden von Telegrammen

Das Versenden von Telegrammen erfolgt in 5 Schritten, wobei Schritt 0 nur ein einziges Mal auszuführen ist:




**Schritt 0:**




Vor der ersten Benutzung eines Telegramms ist eine entsprechende Telegrammklasse anzulegen. Diese wird von der Basisklasse *CTelegram* abgeleitet. Hinzuzufügen sind Membervariablen für jedes zu übertragende Datum sowie eine *Build()*- und eine *Extract()*-Methode zum Aufbau bzw. Auswertung des Telegramms.

```
class CKSTelegram : public CTelegram
{
public:

  T_LONG Build();
  T_BOOL Extract ();

// data members
  T_STRING  m_strName;  // name of the LST
  T_LONG    m_nLinie;
  T_LONG    m_nKurs;
  T_STRING  m_strTime;  // event time
};
```

## 1. Dem Telegrammobjekt Daten zuweisen

**Schritt 1:**

Von der in Schritt 0 angelegten Telegrammklasse können nun Objekte instanziiert werden. Den Daten-Members des Objekts werden die zu übertragenden Daten zugewiesen.

```
CKSTelegram KSTelegram;

KSTelegram.m_strName ="KS30.11";
KSTelegram.m_nLinie = "15";
KSTelegra.m_nKurs   = "48";
KSTelegram.m_strTime = "20011011094815";
```

**Schritt 2:**

## 2. Telegramm-Body bauen

Durch Aufruf der *Build()*-Methode des Telegrammobjektes werden die Daten aus den Membervariablen in einen XML-Telegrammbody eingetragen:

```
KSTelegram.Build ();
```

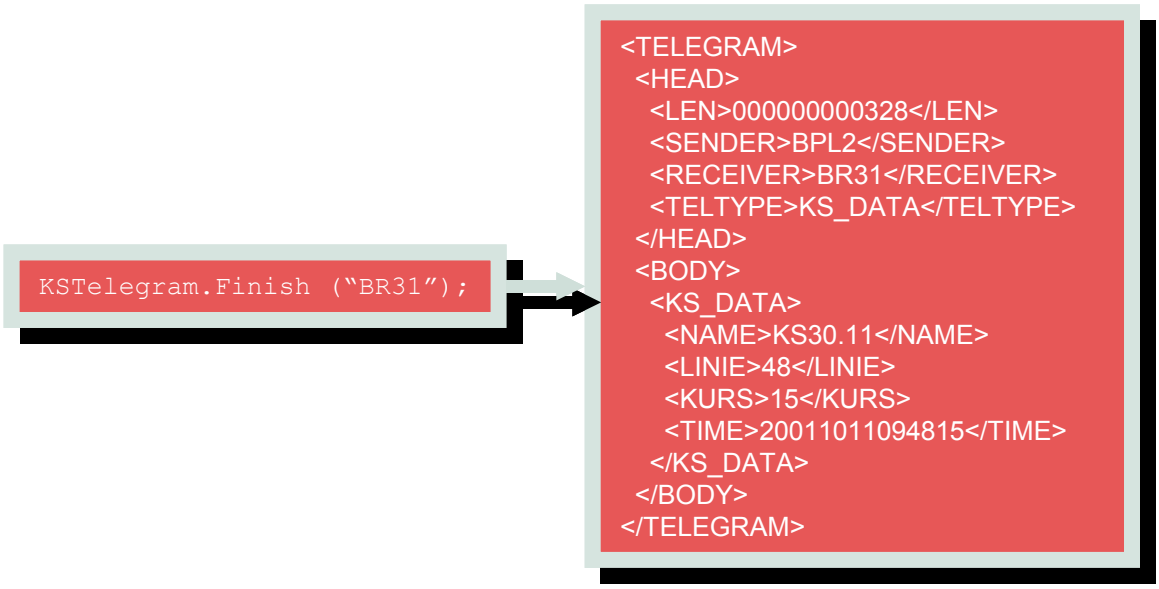
```
<KS_DATA>
   <NAME>KS30.11</NAME>
   <LINIE>48</LINIE>
   <KURS>15</KURS>
   <TIME>20011011094815</TIME>
</KS_DATA>
```

Die Schritte 1. und 2. können beliebig wiederholt werden, um mehrere Datensätze in einem einzelnen Telegramm zu übertragen.

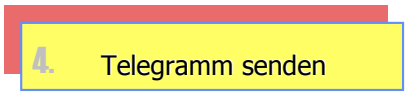
**Schritt 3:**

## 3. Telegramm fertigstellen

Durch Aufruf der *Finish()*-Funktion (in der *CTelegram*-Basisklasse) wird das Telegramm durch den Telegramm-Header ergänzt und komplett fertig gestellt. *Finish()* erwartet als Übergabeparameter den Namen des Empfängers:




**Schritt 4:**




Als letzter Schritt kann das fertige Telegramm über die Framework-Funktion *SendTagTel()* zum Versand an das Framework übergeben werden.

```
SendTagTel (KSTelegram);
```

## 7.3.2 Empfang von Telegrammen

Der Empfang und die Auswertung von Telegrammen erfolgt ähnlich den Schritten zum Telegramm-Versand, mit den Abläufen in umgekehrter Reihenfolge. Statt *Build()* ist die Methode *Extract()* aufzurufen, die die Daten aus dem XML-Telegramm in die Membervariablen des Telegramm-Objekts extrahiert.

Bei Anwendungs-Start ist dem Framework für jeden zu empfangenden Telegrammtyp eine Callback-Funktion bekannt zu geben, welche durch das Framework bei Empfang des zugehörigen Telegramms aufgerufen wird.

# 8 Kommunikation

## 8.1 Grundsätzliche Überlegungen

Als hauptsächliche Technologie zur rechner/plattformübergreifenden Kommunikation greift SiMAP®-FIS auf TCP/IP zurück. Auf die Benutzung weiter von der Übertragungstechnik abstrahierender Modelle wie Microsoft DCOM wurde bewusst verzichtet. Die Gründe für TCP/IP waren u.a.:

- etablierter Standard, ähnliche Implementierung in Windows und Linux
- Anzukoppelnde Fremdsysteme wie Anzeigen etc. benutzen meist ebenfalls TCP/IP
- Verbindungsorientierte, zuverlässige Übertragung. Ausgefallene Systemteilnehmer werden sicher erkannt.
- Mit dem Debugger gut nachverfolgbar
- Bereits gemachte gute Erfahrungen in verschiedenen Projekten

### 8.1.1 Ein erstes Modell

Im ersten Schritt sah der Entwurf des FIS-Basisframeworks drei Module vor:

- Einen TCP/IP-Client für das Senden von XML-Telegrammen, der genau eine Verbindung zu einem entfernten Server zur Zeit aufbauen kann
- Einen Server für den Empfang von XML-Telegrammen, der mehreren entfernten Clients gleichzeitig das Verbinden ermöglicht
- Das Logik/Steuermodul mit der zentralen Applikations-Logik.

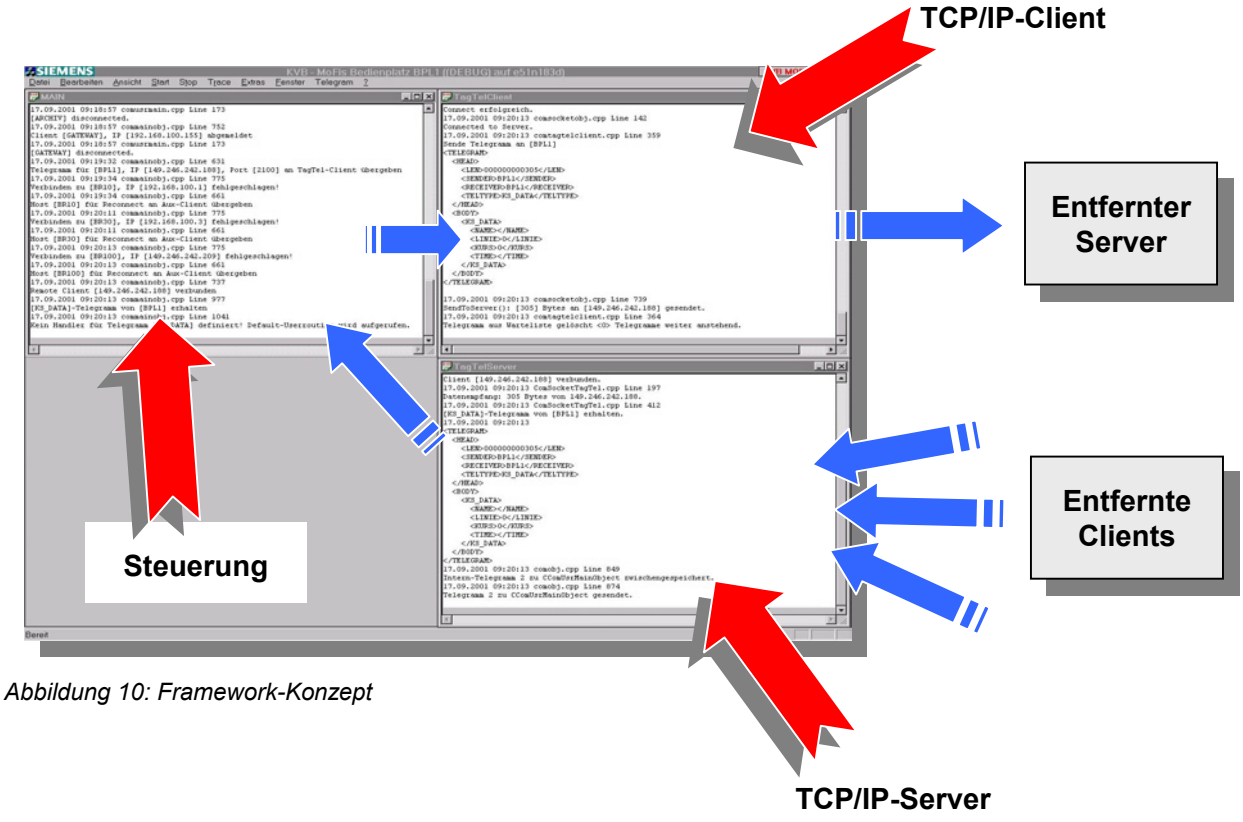



*Abbildung 10: Framework-Konzept*

Client und Server sind spezialisiert auf die Kommunikation mit XML-Telegrammen via TCP/IP und besitzen keinerlei darüber hinaus gehende Intelligenz.

Vollständig erhaltene Telegramme werden vom Server-Modul unmittelbar zur Verarbeitung an das Steuerungs-Modul weitergeleitet. Dort existiert pro zu verarbeitendem Telegramm-Typ eine eigene Callback-Methode, die nach Telegramm-Empfang durch den Framework-Telegramm-Dispatcher aufgerufen wird.

Zu versendende Telegramme übergibt das Steuerungs-Modul an den TCP/IP-Client. Falls die Verbindung mit dem Empfänger-Rechner noch nicht besteht, wird diese vom Client aufgebaut und das Telegramm daraufhin versendet. Der Client hält die Verbindung aufrecht, bis Telegramme an ein anderes Ziel zu versenden sind. Jede SiMAP®-FIS-Applikation ist also sowohl Client als auch Server.

Dieses Modell arbeitet sehr zuverlässig auch bei hohem Telegramm-Aufkommen. Allerdings wird die Verbindungsorientierung von TCP/IP zu einem Nachteil, sobald ein Netzwerkabschnitt nicht erreichbar ist:

Kann die Verbindung zu einem Kommunikationspartner nicht aufgebaut werden, blockiert der Client im *connect()*-Aufruf des TCP/IP-Stacks, bis der Verbindungsversuch vom Betriebssystem abgebrochen wird. Dabei sind zwei Fälle zu unterscheiden:

1. **Die Netzwerkverbindung zum Zielrechner ist unterbrochen**
   In diesem (seltenen) Fall wird TCP wiederholt versuchen, eine Verbindung zum Zielsystem aufzubauen. Nach jedem erfolglosen Versuch erhöht TCP die Timeout-Zeit für den nächsten Versuch, bis die maximale Versuchsanzahl erreicht wurde. Erst dann kommt der *connect()*-Aufruf mit Fehlermeldung zurück und gibt den TCP/IP-Client frei.
   Insbesondere, wenn weitere Telegramme zum Versand an andere Rechner anstehen, ist die damit verbundene Zeitverzögerung unakzeptabel.
2. **Die Ziel-Applikation auf dem Remote-Rechner läuft nicht**
   Ist der entfernte Rechner erreichbar und nur die Ziel-Applikation nicht laufend, quittiert das Betriebssystem des Zielrechners den Verbindungsaufbauwunsch negativ. Dieser Vorgang ist mit keiner nennenswerten Zeitverzögerung verbunden

### 8.1.2 Das realisierte Modell

Zur Vermeidung des oben unter 1. „Die Netzwerkverbindung zum Zielrechner ist unterbrochen“ beschriebenen Nachteils wurde dem FIS-Framework ein viertes Modul hinzugefügt: Die Netzüberwachung.

Die Netzüberwachung ist ein zyklisch im Hintergrund laufender TCP/IP-Client, der periodisch Lebenszeichen-(„ALIVE“)-Telegramme an alle bekannten Kommunikationspartner sendet. Tritt dabei ein Fehler auf, wird der zugehörige Partner in einer internen Liste als „nicht erreichbar“ markiert und eine Meldung an das Archiv sowie alle angemeldeten Bedienplätze gesendet. Dieser Zustand bleibt erhalten, bis in einem folgenden Durchlauf der Partner wieder erreicht werden kann.

Der „Arbeits-TCP/IP-Client“ wird nun nur mit als „erreichbar“ markierten Partnern Verbindungen aufbauen.

Durch dieses Konzept wissen alle im System vorhandenen Rechner stets über den Zustand ihrer Kommunikationspartner bescheid. Ein zentraler Rechner zur Systemüberwachung wird eingespart.

# 8.2 Kurzer Überblick über einige SiMAP®-Klassen

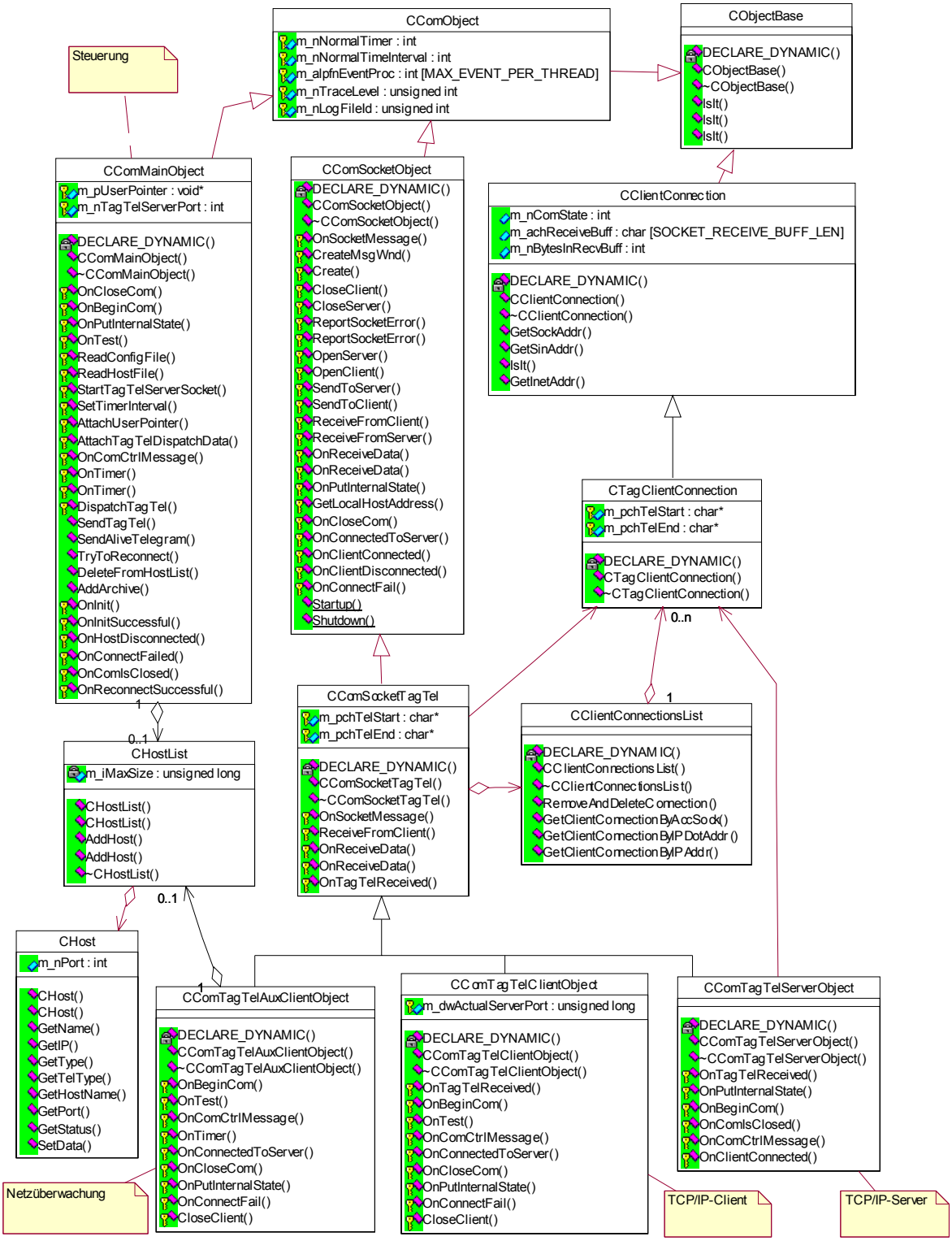



*Abbildung 11: Kommunikations-relevante SiMAP®-Klassen (Ausschnitt)*

Das obige Klassendiagramm zeigt einen Ausschnit der kommunikationsrelevanten Klassen der SiMAP®-FIS-Bibliothek. Dazu gehören:

- **CComObject** ist die Basis aller SiMAP®-Module (Kommunikationsobjekte). Diese Klasse bietet grundlegende Funktionen, wie Initialisierung, Logging und (Windows-) Message-Handling. Zusätzlich wird ein Timer zur Verfügung gestellt. Die Klasse kann unmittelbar zur Erstellung eigener Module benutzt werden. Für spezielle Aufgaben existieren weitere, hiervon abgeleitete Klassen, wie die nachfolgend beschriebenen.
- Zur Kommunikation über TCP/IP wird die Klasse **CComSocketObject** von *CComObject* abgeleitet. *CComSocketObject* kapselt die Socket-Funktionalitäten der Microsoft-Winsock-Bibliothek und bietet so die Basis für Kommunikation über Sockets.
- **CComSocketTagTel** ergänzt *CComSocketObject* um die Verwendung von „getaggten Telegrammen“ im XML-Stil. Dazu gehört die Empfangsbuffer-Verwaltung für getaggte Telegramme unter Zuhilfenahme eines Objekts der Klasse *CTagClientConnection* pro verbundenem Client. Mehrere Methoden von *CComSocketTagTel* sind abstrakt und müssen von abgeleiteten Klassen implementiert werden.
- **CTagClientConnection** verwaltet neben dem erwähnten Empfangsbuffer weitere Informationen pro verbundenem Client, darunter die Client-IP-Adresse. Ein Objekt dieser Klasse wird pro verbundenem Client durch *CComSocketTagTel* angelegt. Objekte der Klasse *CTagClientConnection* werden über die Klasse **CClientConnectionsList** verwaltet.
- **CComTagTelServerObject** bildet das TCP/IP-Server-Modul unter Benutzung der von *CComSocketTagTel* geerbten Funktionalitäten und implementiert u.a. die geerbte abstrakte Methode *OnTagTelReceived()* zur Weiterverarbeitung empfangener getaggter Telegramme.
- **CComTagTelClientObject** bildet entsprechend das TCP/IP-Client-Modul.
- **CComTagTelAuxClient** ist ähnlich dem *CComTagTelClientObject* und bildet unter Hinzunahme weiterer Funktionalität das Netzüberwachungs-Modul des SiMAP®-Frameworks.
- Die Klasse **CComMainObject** bildet durch Definition mehrerer abstrakter Methoden das Gerüst für das Steuerungs-Modul des Frameworks. Der Applikations-Entwickler (der „Benutzer“ des Frameworks) hat diese Methoden zu implementieren und die jeweilige Applikations-Logik in dieses Modul zu integrieren.
- **CHostList** verwaltet Objekte der Klasse **CHost**, die von *CComTagTelAuxClient* und *CComMainObject* benutzt werden. Ein einzelnes CHost-Objekt speichert jeweils Informationen eines einzelnen Kommunikationspartners, wie Hostname, IP-Adresse, Port, sowie den Erreichbarkeitsstatus des Host („erreichbar“, „nicht erreichbar“, „unbekannt“). Die Host-Liste wird beim Framework-Start über eine Konfigurationsdatei („FIS_HOSTS“) initialisiert.

# 8.3 Der TCP/IP-Client

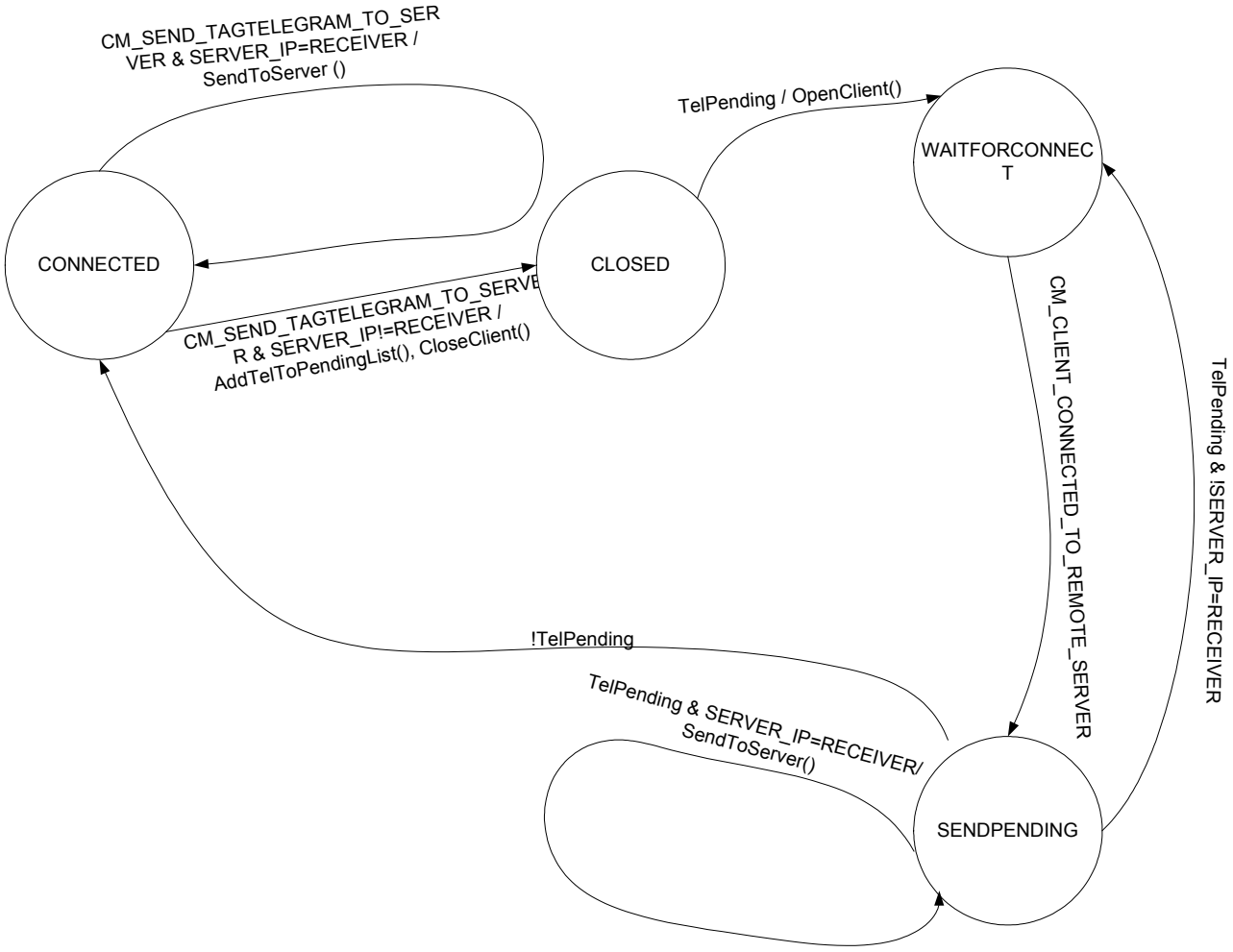



*Abbildung 12: Zustandsdiagramm TCP/IP-Client*

Der TCP/IP-Client verhält sich im wesentlichen nach dem obigen Zustandsdiagramm. Dabei sind 4 Zustände einnehmbar:

1. **CLOSED:** Dies ist der Ruhezustand, wenn keine Verbindung besteht. Sobald ein Telegramm zum Versand in der Sende-Queue ansteht („pending“), setzt der Client innerhalb des *CComSocketObject.OpenClient()*-Aufrufs eine asynchrone *connect()*-Anforderung an das Betriebssystem ab und wechselt in den Zustand **WAITFORCONNECT.**
2. **WAITFORCONNECT:** In diesem Zustand wartet der Client auf die Benachrichtigung durch das Betriebsystem nach einem *connect()*-Aufruf.
3. **SENDPENDING:** War der Verbindungsaufbau erfolgreich, werden in diesem Zustand alle in der Sende-Queue anstehenden Telegramme mit gleichem Ziel gesendet. Stehen nur noch Telegramme für andere Ziele in der Queue, wird die Verbindung abgebaut und mit neuem Ziel wieder aufgebaut. Der Vorgang wiederholt sich ab **WAITFORCONNECT.**
   Ist die Sende-Queue leer, bleibt die zuletzt hergestellte Verbindung bestehen. Der Client wechselt in den Zustand **CONNECTED.**

4. **CONNECTED:** In diesem Zustand wird auf neue zu versendende Telegramme gewartet. Trifft ein solches ein und stimmt dessen Ziel mit der bestehenden Verbindung überein, wird das Telegramm unmittelbar versendet. Andernfalls wird es in die Sende-Queue eingereiht, die bestehende Verbindung wird geschlossen und der Vorgang im Status **CLOSED** fortgesetzt.

## 8.4 Der TCP/IP-Server

Der TCP/IP-Server wartet nach dem Öffnen eines Server-Sockets auf Verbindungen durch Remote-Clients. Pro sich verbindendem Client wird ein Objekt von *CTagClientConnection* angelegt und damit pro Verbindung ein Empfangsbuffer initialisiert.

Innerhalb der Methode *OnReceiveData()* der Klasse *CComSocketTagTel* wird auf eingehende getaggte Telegramme, eingeschlossen durch die Tags <TELEGRAM>...</TELEGRAM> gewartet.

Sobald das Telegramm-Start-Tag erkannt wurde, werden alle zu diesem Telegramm gehörigen Bytes aus dem Empfangsbuffer in ein Objekt der Klasse *CTelegram* verschoben.

Bei Erkennen des Telegramm-Ende-Tags wird die Methode *OnTagTelReceived(),* implementiert in der Klasse *CComTagTelServerObject*, aufgerufen. Diese prüft durch *CTelegram.CheckHeader()* den Header des erhaltenen Telegramms auf Konsistenz. Hier wird insbesondere überprüft, ob der im Telegramm-Header eingetragene „Soll-Empfänger“ mit dem empfangenden „Ist-Empfänger“, also dem Namen der eigenen Applikation, übereinstimmt. Im OK-Fall wird das Telegramm per Windows-Message an das Steuerungsmodul zur Methode *DispatchTagTel()* weitergeleitet.

## 8.5 Das Steuerungs-Modul

Die eigentliche Applikations-Logik wird im Steuerungs-Modul zentralisiert.

Durch die Ableitung von ***CComMainObject*** steht eine Anzahl von Methoden zur Verfügung, die teilweise durch den Applikations-Entwickler zu implementieren sind und teilweise durch diesen aufgerufen werden können. Einige dieser Methoden werden nun kurz vorgestellt.

### 8.5.1 Callback-Funktionen

Der Benutzer wird vom Framework durch vordefinierte Callback-Funktionen über aufgetretene Ereignisse informiert. Dazu gehören beispielsweise Initialisierungs-, Timer- und Kommunikationsstatus-Funktionen. Innerhalb dieser Funktionen hat der Benutzer die Möglichkeit, auf aufgetretene Ereignisse spezifisch zu reagieren.

- **OnInit()**
  *OnInit()* wird bei jedem Start der Kommunikation aufgerufen. Eigene Initialisierungen sowie das Anlegen der Callback-Methoden für das Telegramm-Dispatching sollten hier erfolgen

- **OnInitSuccessful()**
  *OnInitSuccessful()* wird durch das Framework aufgerufen, wenn die gesamte Applikation erfolgreich initialisiert und gestartet werden konnte. Hier können weitere User-Initialisierungen eingebracht werden, die ein laufendes Framework voraussetzen.

- **OnTimer()**
  *OnTimer()* wird bei jedem Ablauf des modul-eigenen Timers aufgerufen. Das Zeitintervall der Aufrufe kann (z. B. in *OnInit()*) in Millisekunden mittels *SetTimerInterval()* festgelegt werden.

- **OnHostDisconnected()**
  *OnHostDisconnected* wird nach jedem regulären Verbindungsabbau zu einem vormals verbundenen Host aufgerufen. Dies ist z. B. der Fall, wenn sich die Remote-Seite abmeldet oder der lokale Client eine Verbindung zu einem anderen Host aufbaut.

- **OnConnectFailed()**
  *OnConnectFailed()* wird bei jedem fehlgeschlagenen Verbindungsversuch aufgerufen. Der Status des nicht erreichbaren Hosts wird innerhalb des zugehörigen *CHost*-Objekts auf „*nicht erreichbar*“gesetzt. Alle weiteren Sendeversuche an diesen Host werden verworfen, bis der Status des Hosts durch das Netzüberwachungs-Modul wieder auf „erreichbar“ gesetzt wurde.

- **OnComIsClosed()**
  *OnComIsClosed()* wird aufgerufen, wenn die Kommunikation angehalten wurde. Dies kann sowohl bei Beendigung der Applikation als auch durch vom Benutzer ausgelöste Ereignisse erfolgen.

## 8.5.2 Aufrufbare Funktionen

In diese Kategorie fallen Funktionen, die das Framework bereitstellt und die vom Applikations-Entwickler zu Initialisierungs- und Steuerzwecken aufgerufen werden können.

- **AttachTagTelDispatchData()**
  Für jedes zu empfangende XML-Telegramm hat der Applikations-Entwickler eine eigene Auswerte-Callback-Methode anzulegen.
  Mit *AttachTagTelDispatchData()* wird dem Framework eine Struktur mit den Telegramm-Typen zu empfangender Telegramme sowie je ein Pointer auf die telegramm-spezifische Callback-Funktion bekanntgegeben.
  Empfangene Telegramme nimmt die Methode *DispatchTagTel()* entgegen und leitet diese an die zugehörigen Auswerte-Callback-Methoden weiter.

- **SetTimerInterval()**
  Setzt das Zeitintervall in Millisekunden für zyklische Aufrufe der *OnTimer()*-Funktion.

- **SendTagTel()**
  Sendet ein getaggtes Telegram über den TCP/IP-Client an einen entfernten Server. Der Funktion ist das vollständige Telegramm als Objekt einer von *CTelegram* abgeleiteten Telegramm-Klasse zu übergeben.

# 9 Verzeichnisse

## 9.1 Literatur

### 9.1.1 Weblinks & Online-Dokumente

[msdn] Microsoft Developer Network Library Visual Studio 6.0-Release, Online-Hilfe zu Microsoft Visual Studio, Stand Januar 2001. Siehe auch http://msdn.microsoft.com

[selfxml] *http://www.netzwelt.com/selfhtml/xml/index.htm*
Münz, Stefan: Einführung in XML auf den bekannten Seiten www.selfhtml.de

[W3C] http://www.w3.org/XML
Die XML-Spezifikationen des W3C-Konsortiums

[simap] http://www.simap.de
Die Webseite zur *Siemens Modular Application Platform*

### 9.1.2 Gedrucktes

[Weiden99] Weiden, Ralf: „Erweiterung einer PID-Digitalreglersoftware einschließlich TCP/IP-Kopplung an einen PC“, Kapitel 4: „Kommunikation mittels TCP/IP“. Diplomarbeit 1999, FH Köln, Institut für Nachrichtenverarbeitung und Prozessautomatisierung

[Hilf00] Kunze, J.; Boehm, A.: „Visual C++: Windows-Programmierung mit den Microsoft Foundation Classes (MFC)“, 2000, Kursunterlagen der Hilf AG

[Hilf99] Kunze, J.; Boehm, A.: „ANSI C++: Objektorientierte Software-Entwicklung“, 1999/2000, Kursunterlagen der Hilf AG

[Kruglinski98] Kruglinski, D., Wingo, S., Shepherd, G.: „Inside Visual C++ 6.0”, Mircosoft Press Deutschland, 1998

[Nussbaumer02] Nussbaumer, A., Mistelbacher, A.: „XML Ge-Packt – Die Ge-Packte XML-Referenz“, mitp 2002

[Oesterreich01] Oesterreich, Bernd: „Objektorientierte Software-Entwicklung. Analyse und Design mit der Unified Modeling Language“, Oldenbourg 2001

[Gamma96] Gamma, Helm, Johnson, Vlissides: „Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software“, Addison-Wesley 1996

## 9.2 Abbildungen



## 9.3 Tabellen