Erfassen und Auswerten von Störungen und Statusmeldungen im Rohbau eines Automobilwerkes

Alexander Stumpp

Curitiba, 20.03.2003

DIPLOMARBEIT

I. Inhaltsverzeichnis:

II. Abbildungsverzeichnis:

II. Listingverzeichnis:

IV. Abkürzungsverzeichnis:

AMIS	:	Alarm Management und Information System
DB	:	Datenbank
FH	:	Fachhochschule
FHKN	:	Fachhochschule Konstanz
ID	:	Identifikationsnummer
MIS	:	Management Information Systems
MFC	:	Microsoft Foundation Classes
SICALIS	:	SICALIS PMC
SPS	:	Speicherprogrammierbare Steuerungen
Symboltabellen	:	Symbolzuordnungstabellen
TI	:	Technische Informatik
VW/Audi	:	VW/Audi do Brasil

1. Vorwort

Die Diplomarbeit wurde bei VW/Audi do Brasil in einem Zeitraum von 6 Monaten durchgeführt. Das Besondere meiner Arbeit war nicht nur eine bisher unbekannte Arbeitsumgebung in einem fremden Land, sondern auch die Tatsache, dass ich einer der ersten Diplomanden bei VW/Audi war. Da es in der Firma keine Informatiker gab, welche vergleichbare Arbeiten wie die meine machten, stand mir daher auch keine dafür vorhandene Fachliteratur zur Verfügung. Somit war es auch eine besondere Herausforderung, alle notwendigen Quellen und Informationen alleinig aus dem Internet und digitalen Unterlagen, alter FH Vorlesungen, zusammen zu tragen.

Meine Diplomarbeit bei VW/Audi do Brasil bestand aus 2 Teilen.
Im ersten Teil galt es, für den geplanten Umbau des Rohbaus, alle 120 SPS Anlagen so umzustellen, dass die Symbolzuordnungstabellen auf einen Standard gebracht werden. Da diese Umstellung von Hand zu viel Zeit in Anspruch nehmen würde, sollte ein Programm bei der Aktualisierung der Symboltabellen assistieren, bzw. einen Großteil der Meldungen bereits selbstständig normieren.
Der zweite Teil der Diplomarbeit war die Erweiterung des lokalen Pager-Systems. Das vorhandene Pager-System ermöglicht lediglich den Personenruf über das Telefon und es übermittelt, bei einem Fehlerfall in der Produktion, eine Meldung an das Wartungspersonal. Dieses System sollte um eine manuelle Nachrichtenfunktion erweitert werden, mit der von einem Arbeitsplatzrechner aus, Meldungen an Pagergeräte verschickt werden können.

Als Entwicklungsumgebung wurde von der Firma VW/Audi ein Produkt der Visual Studio-Serie von Microsoft bereitgestellt. Da beide Programme für den Einsatz auf einem Microsoft Betriebssystem gedacht waren, konnte daher von der kompletten Funktionalität der "Visual Studio .NET" Programmierumgebung Gebrauch gemacht werden. So kommen in beiden Applikationen die MFC-Klassen zum Einsatz. Diese Klassen bieten für alle Bereiche der Programm-Entwicklung notwendigen Funktionen. So konnten beispielsweise für das erste Programm Klassen verwendet werden, die zur Bearbeitung von Texten gedacht sind. Beim zweiten Programm verwendete ich hingegen Funktionen der MFC-Klasse CSocket, womit die Netzwerkkommunikation zwischen Rechnern ermöglicht wird.

Beide Programme erweitern gemeinsam das Anlagen Management Information System bei der Firma VW/Audi do Brasil. Das dort bereits eingesetzte MIS stammt von der Siemens AG und nennt sich SICALIS PMC. Dieses MIS deckt bei VW/Audi do Brasil Bereiche des Managements von der Prozesswerterfassung, bis hin zum Werkskalender ab.

2. Aufgabenstellung

Für den Bereich des Rohbaus ist ein Programm zu entwickeln, durch das Meldungen der S5 Steuerungen für das Management und Informationssystem SICALIS PMC auf einen normierten Stand gebracht werden können. Außerdem ist eine Software zu entwickeln, mit der von einem beliebigen Arbeitplatzrechner aus Nachrichten an einen Pager versendet werden können.

Abbildung 2.1 zeigt nun, an welchen Stellen die während der Diplomarbeit entstandene Software in die Struktur des Rohbaus eingreift. Im Bereich der S5 Steuerungen kommt SIC-AU, welches die Symboltabellen der S5 Anlagen überarbeiten soll, zum Einsatz. Nach der Umstellung der Symboltabellen, haben die Meldungen an den zentralen SICALIS Server einen höheren Informationsgehalt. Diese Meldungen werden von SICALIS PMC ausgewertet und können dann zur Visualisierung an ANDON Anzeigetafeln und an Pagergeräte weitergeleitet werden. Hier können nun weiters über die SIMAplus-Software, noch zusätzliche manuelle Rufe an Pagergeräte versendet werden.

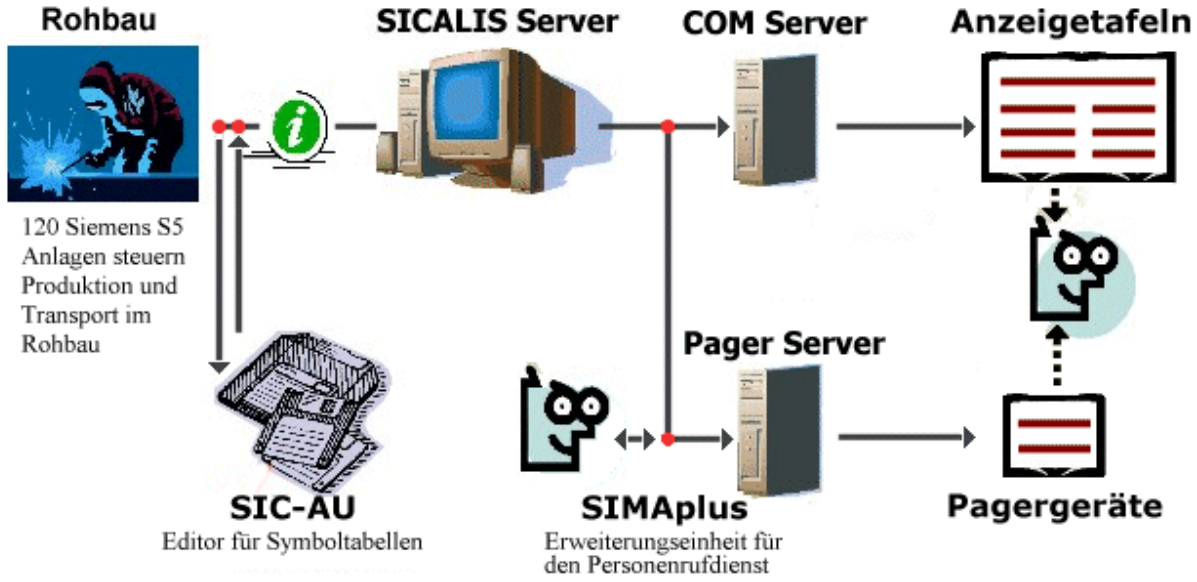

Abbildung 2.1: Übersicht über das Informationssystem für Störungen und Statusmeldungen im Rohbau

3. Management und Informationssystem

Der Begriff Management-Informationssystem ist nicht eindeutig und wird in der Literatur oft mit anderen Begriffen umschrieben. "MIS, EIS, CIS - Babylon ist mitten unter uns" titelt Kaske ihren Artikel (Kaske, 1992) und weist damit auf die begriffliche Vielfalt hin. Alle Begriffe stehen im Grunde für dasselbe. Es sind die Bezeichnungen für Computerunterstützte Informations- und Entscheidungssysteme für das Management.
Der Entwicklung von Entscheidungsunterstützungssystemen liegt die Idee zugrunde, reale Problemlösungsprozesse im Rahmen einer effektiven Managementarbeit zu unterstützen. Neben dem Begriff Management-Informationssystem finden sich die Bezeichnungen Computer Information System (CIS), Entscheidungsunterstützungssystem (EUS), Decision Support System (DSS), Management Support System (MSS), Executive Support Systems (ESS) Führungsinformationssystem (FIS) und Executive Information System (EIS).

3.1 Historie

Bereits in den 70er Jahren wurde der Versuch unternommen, Management-Informationssysteme als total integrierte Systeme für das gesamte Unternehmen einzuführen und sämtliche Führungsebenen des Unternehmens mit Informationen versorgen.

"Dieser Versuch war sowohl wegen des Fehlens leistungsfähiger Datenbanksysteme als auch wegen des instabilen Charakters der Organisationsstrukturen zum Scheitern verurteilt. Seitdem ist der Begriff MIS negativ besetzt." (Behme 1992, S. 179)

"[...] vielen IT-Verantwortlichen ist das Kürzel MIS (Management-Information-System) nach wie vor ein Dorn im Auge. Zu groß waren die Enttäuschungen in den 80er Jahren, denn weder Mainframegestützte Anwendungen noch die auf relationalen Datenbanken basierenden Client/Server-Architekturen waren dazu in der Lage, die in sie diesbezüglich gesetzten Erwartungen zu erfüllen." (Schmelz 1995, S. 72)

Seit den 90er Jahren sind MIS wieder im Kommen. Die Gründe dafür sind nach Untersuchungen des Fraunhofer-Instituts für Arbeitswirtschaft und Organisation (IAO) Fortschritte in Hard- und Software und ein neues Führungsverständnis. Auf der Hard- und Software-Seite ist die Bedienung benutzerfreundlicher geworden durch graphische Benutzeroberflächen und einheitliche Benutzerführung. Das neue Führungsverständnis sieht in der Informationstechnik ein Werkzeug zur Unterstützung strategischer Konzepte und flexiblerer Reaktion (Koll/Engstler, 1994, S. 41f).

3.2 Dienstleistung und Information

Unsere Gesellschaft entwickelt sich von der Dienstleistungsgesellschaft zur Informationsgesellschaft. Diese Entwicklung ist zum einen dadurch gekennzeichnet, dass immer mehr Menschen im Arbeitsprozess mit dem Beschaffen und Verwalten von Informationen befasst sind und zum anderen, dass Informationen immer umfassender und schneller verfügbar werden.

"Wir leben in einer informationsbasierten Gesellschaft, in der inzwischen mehr als 50% der gesamten Arbeitskräfte für Beschaffung, Verarbeitung und Verteilung von Informationen tätig sind." (Herget 1995, S. 26).

Diese Entwicklung wirkt sich natürlich auch auf Unternehmen aus. Das Informationsmanagement in diesen Organisationen steht vor einer grundlegenden Neuorientierung. Information ist zu einem überlebensnotwendigen Betriebskapital geworden, gleichzeitig drohen die Verantwortlichen in der Informationsflut zu ertrinken:

"Fortschritte in den Bereichen Telekommunikation, Medien, Computer- und Softwaretechnik, einhergehend mit zunehmender Integration, ermöglichen es heute prinzipiell, in Sekundenschnelle auf Informationen zurückzugreifen, deren Beschaffung früher Tage oder Wochen dauerte, soweit die Informationen überhaupt zugänglich waren. Der schnelle Zugriff auf eine Vielzahl von Informationen ist aber auch mit eine Ursache für die sich immer rasanter ändernde und komplexer werdende Umwelt. Gleichzeitig wirkt sich das turbulente Umfeld auf die Prozesse der Aufgabenerledigung in Verwaltungen und Unternehmen mit der Folge eines gestiegenen Informationsbedarfs aus." (Herget 1995, S. 26)

Die schnelle Verfügbarkeit von Informationen ist für das Unternehmen lebenswichtig geworden. Information ist ein Produktionsfaktor geworden, der über die Wettbewerbsfähigkeit entscheidet:

"Unternehmensweite Informationsverarbeitung ist für das Unternehmen zu einem entscheidenden Wettbewerbsfaktor geworden. Sowohl die grundsätzliche Gestaltung unternehmensweiter Informationssysteme als auch der innere Bau einzelner Komponenten dieser Informationssysteme sind von strategischer Bedeutung." (Bullinger/Fähnrich 1992, S. 62)

Dieser Bedarf an Informationen äußert sich naturgemäß besonders dort, wo Entscheidungen getroffen werden müssen, nämlich im Management. Dort wird eine besondere Form der Entscheidungsunterstützung gebraucht, ein Informationssystem, das Grundlage für Entscheidungen sein kann:

"Nur wer die richtige Information zur richtigen Zeit am richtigen Ort hat, kann auch Entscheidungen im Sinn eines Marktvorteils treffen. Die Information ist zum Produktionsfaktor geworden. Und hieraus gegründet sich der Ruf nach einem System, das Entscheidungsgrundlagen bietet und zwar einfach, schnell und aktuell - dem Executive-Information-System." (Kaske 1992 S. 46)

Die Verarbeitung von Status und Störungsmeldung, sowie die schnelle Auswertung und Bereitstellung der darin enthaltenen Informationen, stellt in der Fertigung von VW/Audi einen Wesentlichen Teil im MIS SICALIS PMC dar. So ist es also nach den Aussagen von Kaske, Bullinger, Fähnrich, etc. für ein Unternehmen wie VW/Audi do Brasil von entscheidender Wichtigkeit, dass das hier verwendete MIS, SICALIS PMC, so effektiv wie nur möglich arbeiten kann, damit eine weltweite Wettbewerbsfähigkeit gesichert werden kann.

3.3 Weitere Management Information Systeme

Die Siemens AG bietet nicht alleinig ein MIS auf dem Weltmarkt an. So konkurriert SICALIS PMC mit vielen anderen Systemen. Ein Topseller auf dem Weltmarkt stammt beispielsweise von der Firma Rockwell, Milwaukee, USA. Unter dem Namen Rockwell Automation bietet die Company eine Komplettlösung für die gesamte Automobilindustrie an. Ihre Palette reicht hierbei von "Panel-Views" (eine Flüssigkeitsanzeige mit Bedienelementen, welche ein Interface zu den Maschinen der Produktion ist), bis hin zum "Information Flow". Wie auch SICALIS PMC, kann das System mit verschieden vielen Funktionalitäten modular in eine Firma implementiert werden.

Ein weiterer Anbieter eines MIS ist die Firma Intellution, Foxborough, USA. Ihr Tool für die Visualisierung von Produktionsdaten nennt sich iFIX. iFIX gehört zu einer Gesamtreihe von Produkten, welche Intellution für den Automatisierungsbereich dem Konsumenten zur Verfügung stellt. Um in etwa die gleichen Informationen und Funktionen bieten zu können wie SICALIS PMC, müsste zumindest noch das weitere Produkt iDownTime hinzugefügt werden. iDownTime ist eine Komponente, welche Kalkulationen über den Produktionsverlauf anhand der von den von der Produktion gelieferten Daten, anstellt. iDownTime liefert Effektivwerte und Verlaufsanalysen.

GE Fanuc Automation ist ebenfalls ein MIS-Anbieter aus den USA. Ihre Produktpalette CIMPLICITY besteht aus 8 verschiedenen Modulen, welche nach Wunsch des Abnehmers in den Betrieb eingebunden werden können. Wie alle anderen Anbieter auch, so ist CIMPLICITY ebenfalls im "Open System" -Design. "Open System" bedeutet, dass die Anwendungen nach dem offenen Industriestandard von Microsoft programmiert sind und somit mit Komponenten von Dritt-Anbietern kompatibel sind. So können also beispielsweise Komponenten, welche auf COM/DCOM, ActiveX oder VBA basieren, mit den Anwendungen kommunizieren.

Gefasoft bietet die modulare Lösung "Legato" als MIS an. Die deutsche Firma mit Hauptsitz in München, ist seit annähernd fast 20 Jahren im Automatisierungsbereich. Gefasoft hat auch einen Sitz in Sao José dos Pinhais / Parana Brasilien. Die Firma modelliert ihr MIS ebenfalls nach gängigen Standards und setzt daher beispielsweise auf die Leistungsfähigkeit von SQL und auf die Überschaubarkeit der IEC-1131 Scriptsprache.

4. SICALIS PMC

SICALIS PMC ist ein Anlagen-Informations-System, welches vor allem für die Stückfertigung konzipiert ist. Die Schwerpunkte liegen in der Berechnung von Statistiken und der visuellen Darstellung des Zustandes von Fertigungseinrichtungen. Statistiken können für einzelne Maschinen, Teile davon oder für größere Einheiten erstellt werden. In dem System ist eine Datenbank enthalten, welche auf dem Leitrechner installiert ist. Diese DB bietet eine Vielzahl von Daten an, mit denen tabellarische und grafische Auswertungen über Stückzahlen, Störmeldungen, Messwerte u. ä. erstellt werden können. SICALIS PMC bietet weiter eine komfortable Online-Projektierung, mit der die Struktur der Fertigungseinrichtungen beschrieben wird. Mit einer grafischen Oberfläche können Arbeitszeitmodelle mit Schichtwechsel- und Pausenzeiten erarbeitet werden.
Der Einsatz von SICALIS PMC erfolgt zusammen mit SIMATIC-Steuerungen und Windows-PC (Windows NT 4.0). Für die Automatisierungsgeräte bietet SICALIS Schnittstellen und Treiber, die bei geringem Projektierungsaufwand eine performante Übertragung von Daten aus der Steuerungsebene zum Rechner gewährleisten. Als Bussystem kommt Industrial Ethernet mit TCP/IP zum Einsatz.
SICALIS PMC benutzt die Daten aus dem Prozess (Prozesswerte), um ein internes Prozessabbild aufzubauen, außerdem werden Teile der Prozessdaten in eine relationale Datenbank eingetragen. Das interne Prozessabbild dient unter anderem dazu, statistische Daten über den Prozess zu gewinnen, sowie eine Prozessvisualisierung durchzuführen. Zur Abbildung der Produktionsanlage im Prozessabbild und der effektiven Weiterverarbeitung der Prozessdaten, werden diese in ein Adressierschema eingebettet.
Das SICALIS PMC-Adressierschema teilt Anlagen in Bereiche, Aggregate und Prozesswerte auf.
Das Management-Informationssystem für VW/Audi Curitiba ist auf der Basis des Management-Informationssystem SICALIS PMC mit Version V4 realisiert und bietet folgende Funktionen:

- Online-Projektierungssystem
- Prozesswerterfassung aus der Steuerungsebene
- Dynamische Visualisierung aus dem Prozessabbild
- Informationsverteilung an alle angeschlossenen User
- Visualisierungseditor (DV/Gipsy) zum Generieren und Ändern von Bildern
- Grafische Arbeitszeitmodellierung / Werkskalender
- Meldungsmanagement
- Anlagen - Logbuch mit Message-History
- Protokollierung von Datenbankabfragen (Reports)
- Wiederanlaufsteuerung und Schichtwechselkoordinierung
- Ansteuerung von Andon-Boards durch Prozessdaten
- Zuordnung von Reißleinen zum Team, Zuordnung der Prioritäten für die Andon-Boards
- Reißleinen-Meldungsverarbeitung und Auswertung
- Automatische Langzeitarchivierung von Datenbanktabellen

- Zyklische Aufzeichnung von Schichtzählern, Zählern und digitalen Werten als Basis für Kurvenauswertungen
- Parametrieren, Anzeige und Bedienung von Kurvenfenstern
- Kommunikationsprogramm für einen Steuerungstyp SIMATIC S5 U-Reihe
- Kommunikationslizenzen für 150 anzuschließende Steuerungen
- Externe Kommunikations-Überwachung
- Online-Verfügbarkeiten und Online-Anlagen
- Virtuelle Datenpunkte als Verknüpfung realer Prozessgrößen
- Automatische Schaltzeitpunkte zur Anlagensteuerung
- Anwenderprogrammschnittstelle
- Alarmwindow für wichtigste Meldungen mit akustischem Signal
- Message Spot zur Meldungsdarstellung in Anlagenbildern
- Administrations-Tool für Systembetreuer

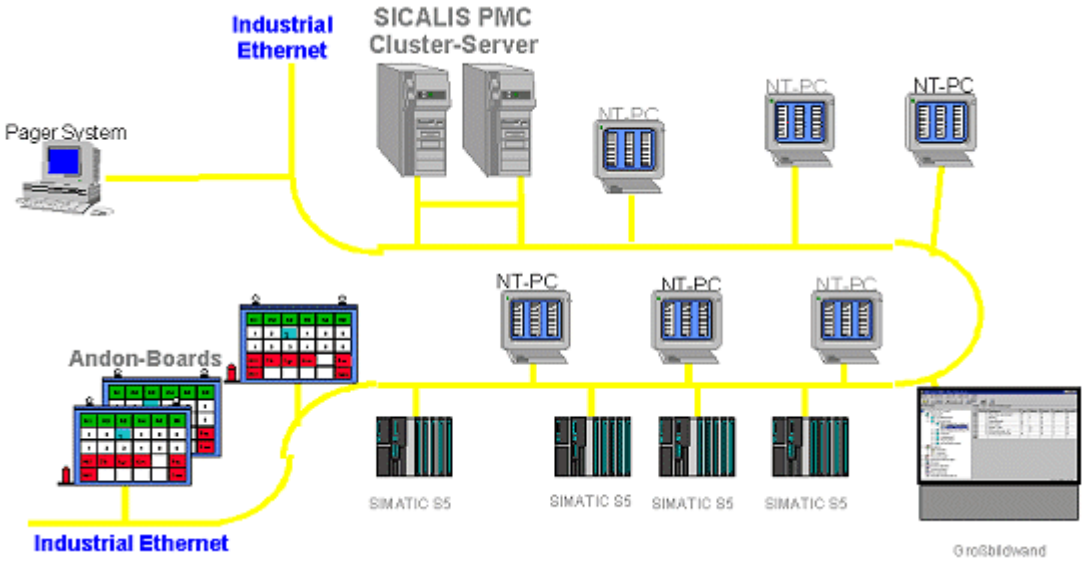

Abbildung 4.1: SICALIS Netzwerk bei VW/Audi do Brasil

Die Abbildungen 4.1 und 4.2 zeigen die Struktur des SICALIS PMC Netzes. Während die erste Abbildung verdeutlicht, auf welche Art und Weise sich SICALIS über ein Firmennetz verteilt, zeigt die zweite Abbildung, wie sich SICALIS im spezifischen Firmennetz von VW/Audi eingliedert.

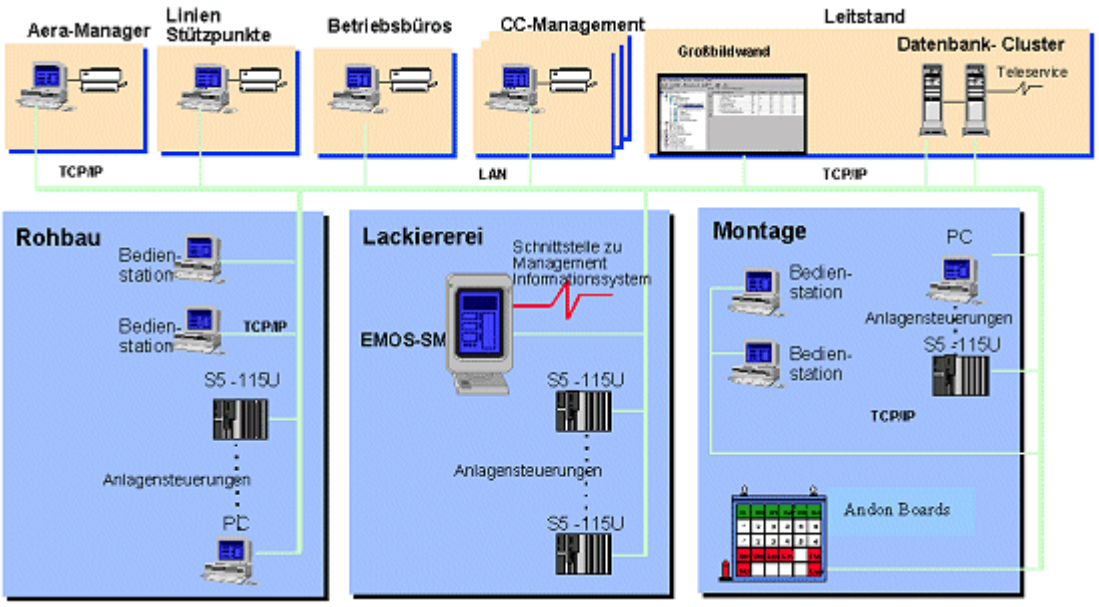

Abbildung 4.2: Struktur des SICALIS-Systems

Das 3 Stufen Modell von SICALIS, welches Abbildung 4.3 zeigt, wird in allen 3 Fertigungsbereichen von VW/Audi separat aufgebaut. Wie sich die 3 Stufen genauer definieren, wir auf den folgenden Seiten genauer erläutert.

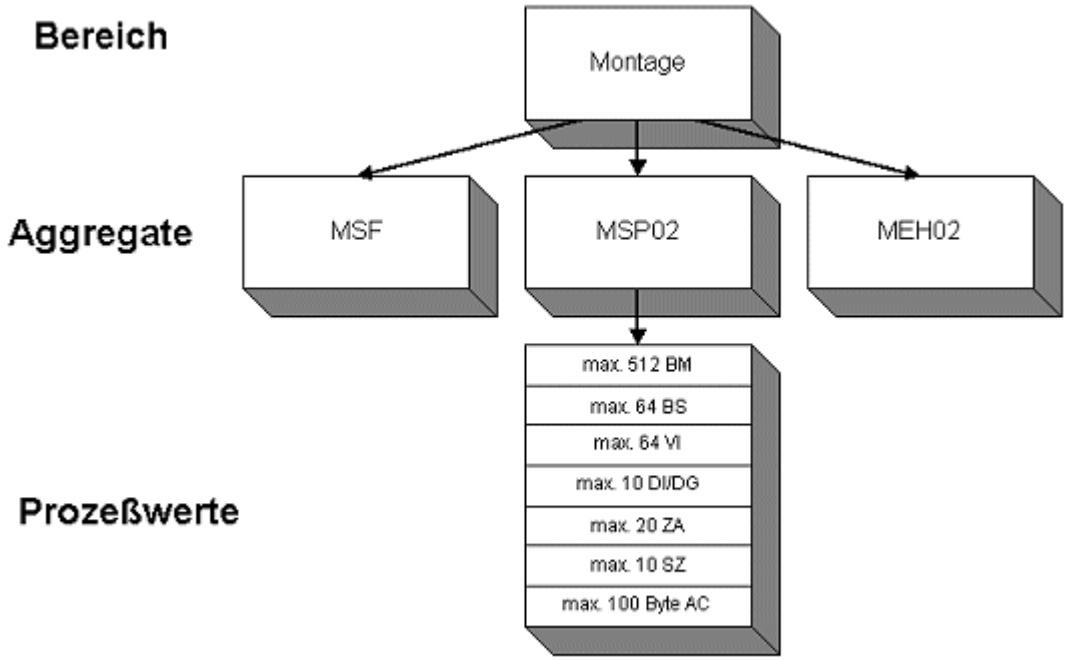

Abbildung 4.3: Stufenweiser Aufbau von SICALIS am Beispiel der Montage

4.1 Bereiche

Bereiche sind ein Hilfsmittel zur Strukturierung der Prozesswerte, die in SICALIS PMC projektiert werden können. Sie stehen in der Hierarchie am höchsten. Bereiche können z.B. einer Maschinengruppe entsprechen oder einen logisch oder räumlich zusammenhängenden Abschnitt (bspw. Instandhaltungsbereich) kennzeichnen. Jeder Bereich hat einen eindeutigen Namen. Bereiche sind ein wichtiges Selektionshilfmittel für die spätere Auswertung der Daten. Bei VW/Audi do Brasil sind die Bereiche mit den Sektionen Rohbau, Lackierung und Montage festgelegt.

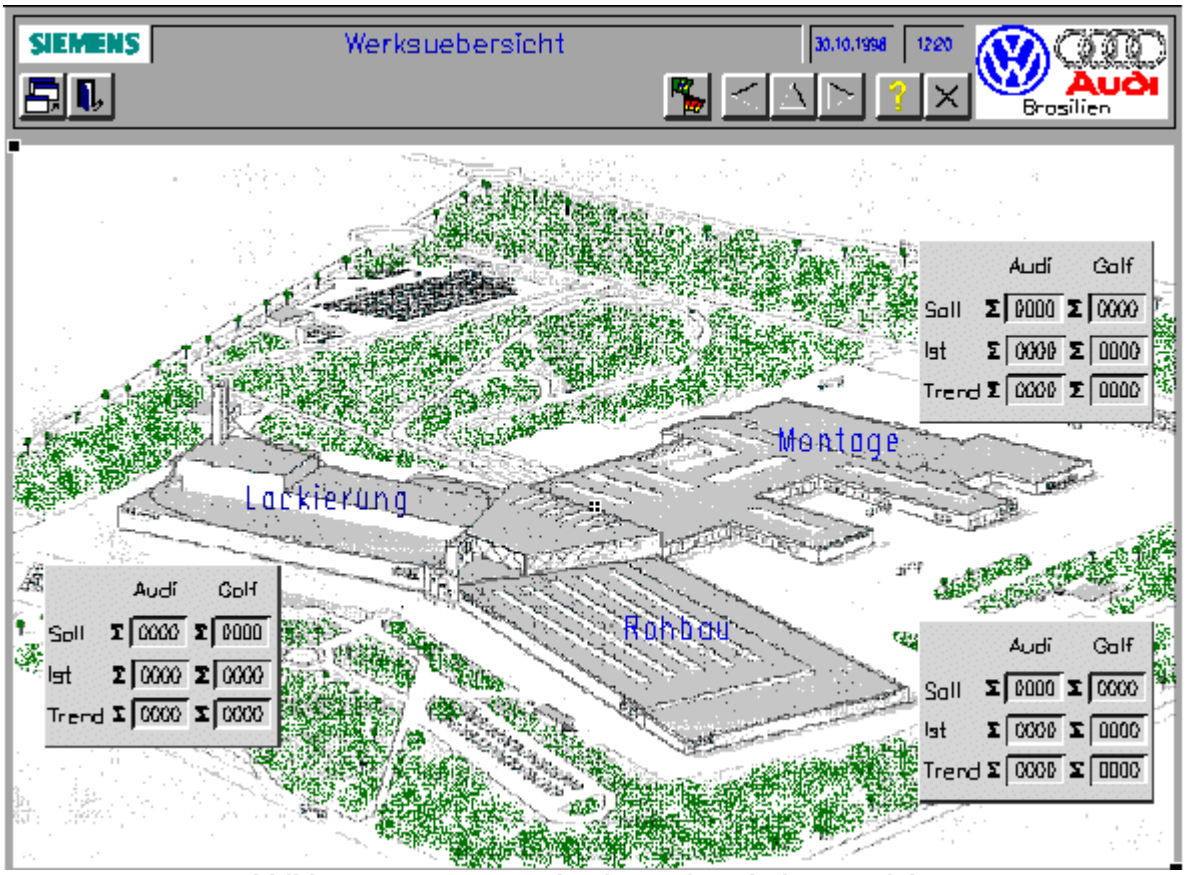

Abbildung 4.4: SICALIS Einstiegsseite mit den Bereichen

4.2 Aggregate

Die Ebene unter den Bereichen bilden die Aggregate. Ein Aggregat ist genau ein Teil einer Anlage. Jedes Aggregat gehört immer genau zu einem Bereich. Der Name eines Aggregates muss innerhalb des Systems eindeutig sein. Den Aggregaten direkt zugeordnet sind die Prozesswerte.
Für Aggregate werden unter anderem Summen von Stördauern und Verfügbarkeiten während einer Schicht berechnet. Daher sollte ein Aggregat immer einer selbstständigen Einheit in der überwachten Produktionsanlage entsprechen.

Einschränkungen bei der Festlegung der Aggregate ergeben sich zum einen aus den Obergrenzen für die Anzahl der Prozesswerte je Aggregat. Zum anderen müssen alle Prozesswerte eines Aggregates aus einer Steuerung kommen.

4.3 Prozesswerte

Ein Prozesswert gehört genau zu einem Aggregat. Innerhalb eines Aggregates werden Prozesswerte gleichen Typs durchnummeriert. Die Anzahl der Prozesswerte pro Aggregat ist beschränkt. Anhand dieser Prozesswerte errechnet SICALIS Produktionsverläufe, welche grafisch ausgegeben werden können.

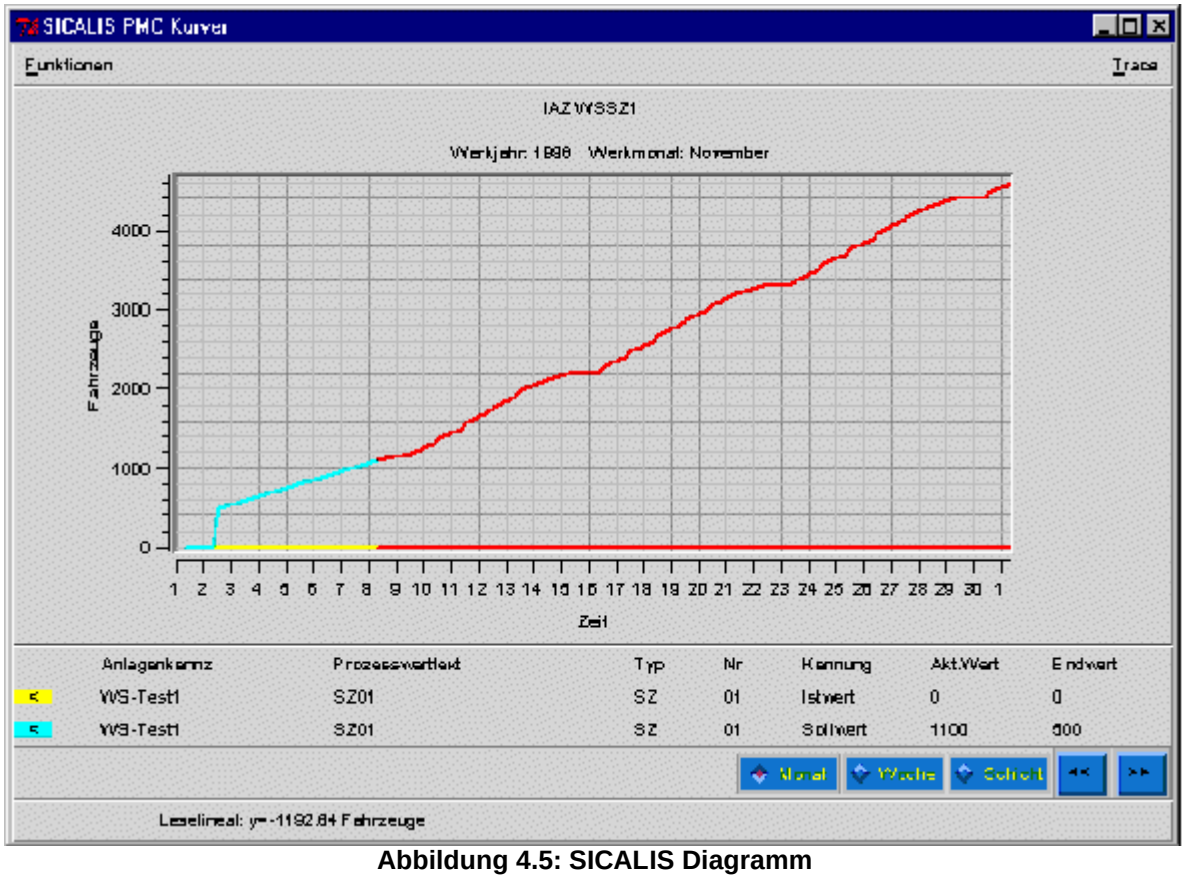

Abbildung 4.5: SICALIS Diagramm

4.4 Meldungen an SICALIS PMC

Um Meldungen an SICALIS PMC zu senden, sind Festlegungen für alle Anlagen, d.h. sowohl für Fördertechnik, Verfahrenstechnik als auch für die Applikationen notwendig. Sämtliche Stillstände und Störungen der Anlage gehen als binäre Meldungen an das Leitsystem; d.h. für jede Störung der Anlage ist eine Störmeldung vorgesehen. Meldungen werden von SICALIS PMC bei entsprechender Parametrierung in das Anlagenlogbuch eingetragen, das ein wichtiges Analysemittel für die Instandhaltung ist. Deshalb sind für alle Meldungen an SICALIS PMC aussagekräftige Meldetexte zu formulieren.

4.5 Alarmwindow

Das Alarmwindow hilft bei der gezielten Analyse von Störursachen. Es ermöglicht die Parametrierung von Meldungen zur ereignisgesteuerten Anzeige auf einem der SICALIS Client PC's. Bei der Parametrierung wird das entsprechende Störbit, sowie das Terminal für das Alarmwindow festgelegt. Bei Eintreffen der parametrierten Störmeldung wird auf dem festgelegten Terminal ein Window aufgeblendet, welches die Störmeldung, bzw. festgelegte Teile davon, anzeigt. Zusätzlich kann das Eintreffen der Störmeldung akustisch signalisiert werden. Das Alarmwindow wird vorrangig in den Fällen eingesetzt, in denen auf bestimmte Störungen zwecks Analyse gewartet werden muss.

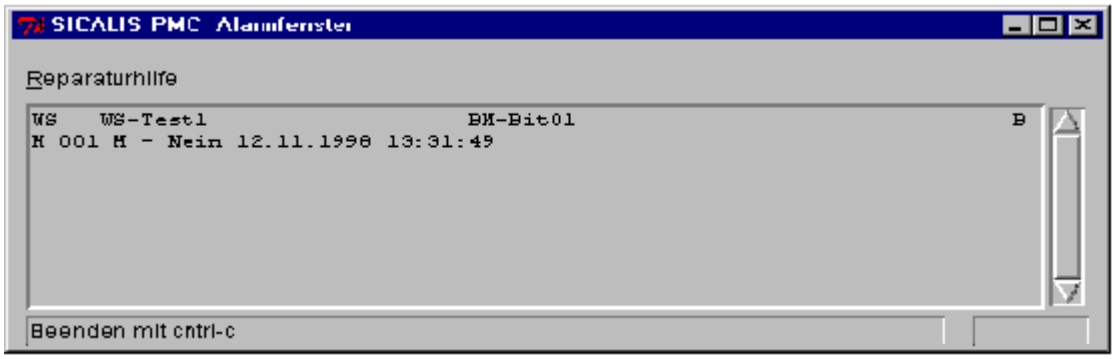

Abbildung 4.6: SICALIS Alarmwindow

SICALIS PMC bildet zusammen mit zwei weiteren Systemen (Abbildung 4.7) ein gesamtheitliches Verwaltungssystem für alle Abläufe der Produktion und trägt somit wesentlich zu einem stabilen Produktionsablauf bei. Die Systeme helfen ebenfalls mit, die Qualität in der gesamten Produktion zu sichern und machen dank SICALIS eine Analyse zur fortlaufenden Qualitätsverbesserung, möglich.

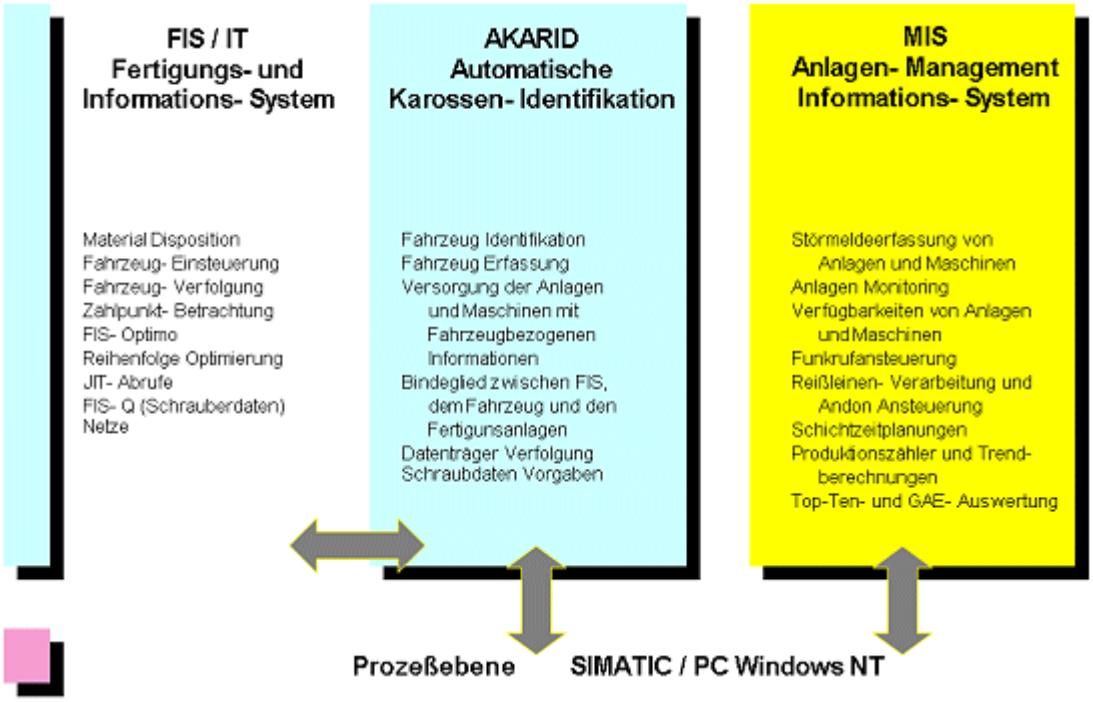

Abbildung 4.7: Kommunikation zwischen SICALIS und weiteren Systemen

5. SIC-AU – Editor für Symboltabellen

Im Rohbau bei VW/Audi do Brasil werden sämtliche Roboter und Förderstrassen von Siemens S5 SPS Anlagen gesteuert. Diese SPS-Anlagen übergeben Informationen an das Management System Siemens SICALIS PMC. SICALIS wertet sämtliche erhaltene Informationen aus, um sie in Grafiken und Hochrechnungen darstellen zu können. So kann mit Hilfe von SICALIS PMC der Verlauf der Produktion analysiert und vorhergesagt werden.
Einige Meldungen jedoch, die an SICALIS weitergeleitet werden, enthalten nur wenige bzw. ungenaue Informationen. Deshalb sind nicht alle erhaltenen Informationen für SICALIS auswertbar. Der Ursprung der Informationen liegt im Kommentarteil der Symboltabellen einer jeden S5 Einheit. Da es bei der Programmierung der S5 Anlagen noch keine Normvorschrift gab, war es den Programmierern überlassen, möglichst sinnvolle Kommentare zu verfassen. Daraufhin wurde in einer kleinen Arbeitsgruppe ein Standard verfasst, welcher nun in eine jede S5 Anlage zu integrieren ist. Durch diesen Standard sollen die Analysemöglichkeiten von SICALIS PMC verbessert und erweitert werden.

5.1 Analyse der S5 Anlagen

Nach einer 2-wöchigen Analyse der S5 Programme und der dazugehörigen Symboltabellen war es eindeutig, dass eine Normierung der Kommentare in den Symboltabellen, nicht anhand des Programmcodes möglich war. Bei Versuchen, gewisse Algorithmen und Schaltregeln aus den Programmen so zu analysieren, dass man sie hätte bestimmten Kommentaren hätte zuweisen können, wurde deutlich, dass gerade die Einfachheit eines S5-Codes, zum Problem wird. Da ein S5 Code mehrheitlich nur aus UND und ODER-Verknüpfungen besteht, waren die Schaltregeln eines Kommentars oftmals identisch mit den Regeln eines anderen Kommentars. So ist vor allem das Beispiel von "Akarid" sehr markant. Bei solch verschiedenen Algorithmen, wie die nachfolgende Tabelle es zeigt, ist es nicht mehr möglich, eine Standardmeldung einer Schaltfolge im SPS-Programm zuzuweisen.

Agua de Resfriamento	M & E → ((S | S) & M) & E
Agua RIP	A & M
AKARID	(M | (M & (S | S))) & (M | M | °E) & (E | E) & °T & E & E E | °E | M | S | S (E | E) & E | M E & E & E & E E & E (E | E) & E | M (E | E) & E & E & °E & °E & M E & E & E & M & M (M | M) & (S | S) & E

Cola	E & (E & E | T) E & E & E E °M S | S E & E S | S | M
Interbus	M | M | S | S
Manual	M
Automatico	(M & M | M) & °M | S | S (E & E | M) & °E | S | S (E | M) & °E | S | S E & E | M | S | S M
Scanner	E | S | S
Solda	(°S & °S | °M) & E M
Tensao	(S | S) & °E
Transportador	(M | (M & (S | S))) & (M | M | °E) & (E | E) & °T & E & E

Zeichenerklärung für Tabelle:

M	= Merker	E	= Eingang	A	= Ausgang
&	= UND	|	= ODER	°	= NICHT
T	= Timer	S	= Setzer		

Jedoch war während der Analyse der Programme auffallend, dass die Kommentare der Symboltabellen immer die in etwa gleichen Worte verwendete. Daher bot sich als Lösung des Problems ein Editor an, welcher anhand der Wortkombinationen eines Kommentars die Entscheidung darüber trifft, ob dieser Kommentar normierbar ist, oder nicht.

5.2 Umsetzung

Es wurde also eine Applikation benötigt, welche einen funktionierenden Algorithmus, zum Ersetzen der Kommentare, beinhaltet. Um die oftmals recht unterschiedlich verfassten, aber dennoch gleichen Kommentare zu erkennen, muss der Algorithmus über eine gewisse Unschärfe verfügen. Diese Unschärfe war gerade so einzustellen, dass keine Kommentare ersetzt werden, die nicht ersetzt werden durften.

Diese Applikation gliedert sich bei der Funktionalität in mehrere Teile:

- Öffnen der, über das Firmennetzwerk zugänglichen, Symboltabellen
- Vergleichen und Filtern von Kommentaren mit einer speziellen, für die Umstellung generierte, Datenbank
- Ersetzen der Kommentare, durch die standardisierten Kommentare innerhalb einer neu erzeugten Datei
- Datenbankeditor

5.3 Funktion und Layout von SIC-AU

Die Applikation besteht aus 2 Teilen. Zum einen gibt es also den Algorithmus, welcher dafür verantwortlich ist, Kommentare auszumachen, welche standardisiert werden können. Zum anderen gibt es eine Datenbankoberfläche, mit der die in der Datenbank befindlichen Eintragungen bearbeitet werden können. Eine Eintragung in der Datenbank - im Weiteren auch Rekord genannt - enthält einen normierten Kommentar und die dazugehörigen Suchworte.
Um die Symboltabellen in der Anwendungsoberfläche gut lesbar darzustellen, ist der linke Frame, in welchen eine Symboltabelle geladen wird, mit einer Mindestgröße initialisiert. Bei einer größeren Monitorauflösung als 800*600, vergrößert sich auch automatisch dieser Frame. Um nach der Überarbeitung der Symboltabelle, die erneuerten, normierten Zeilen, mit den Ursprünglichen vergleichen zu können, werden die ursprünglichen Kommentare in den rechten Frame in der Oberfläche geladen.

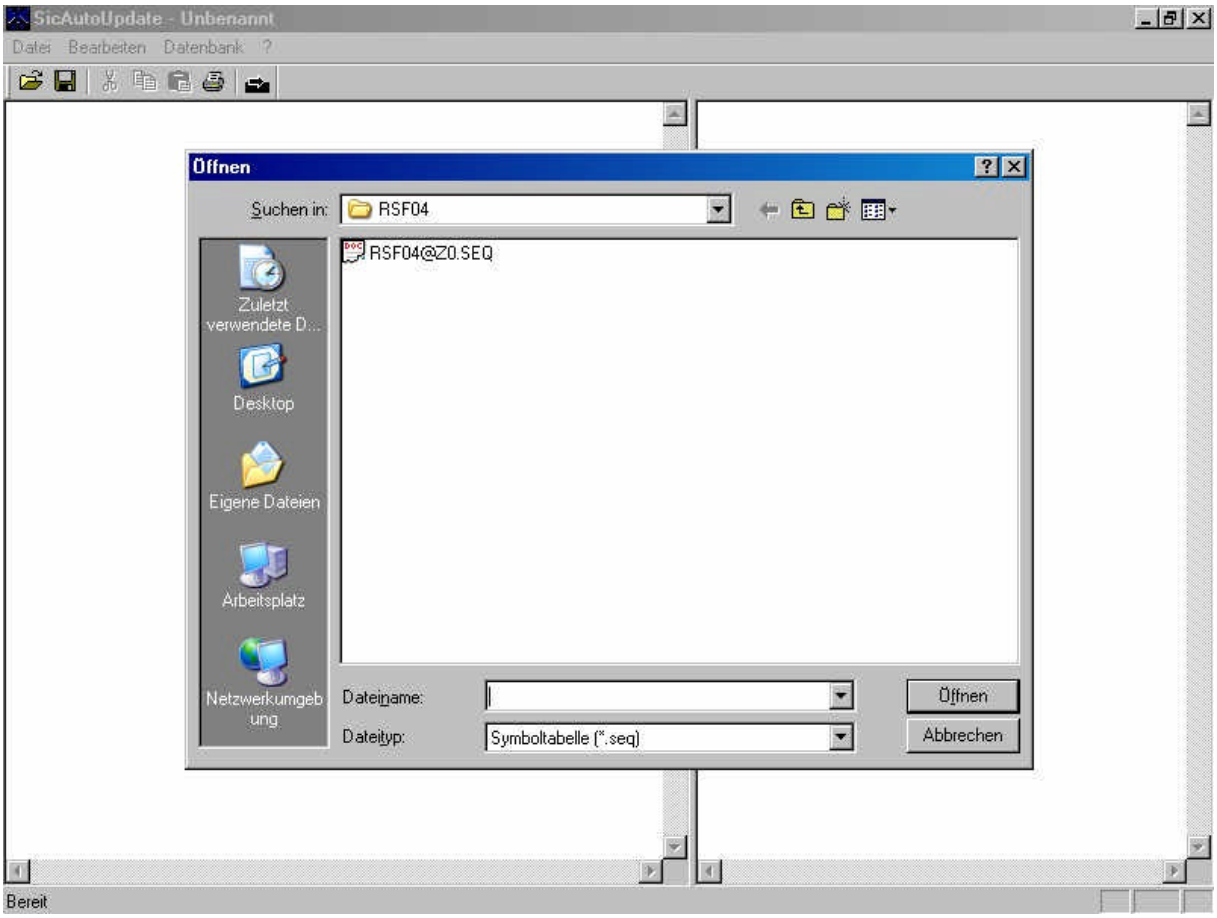

Abbildung 5.1: SIC-AU mit Dialog

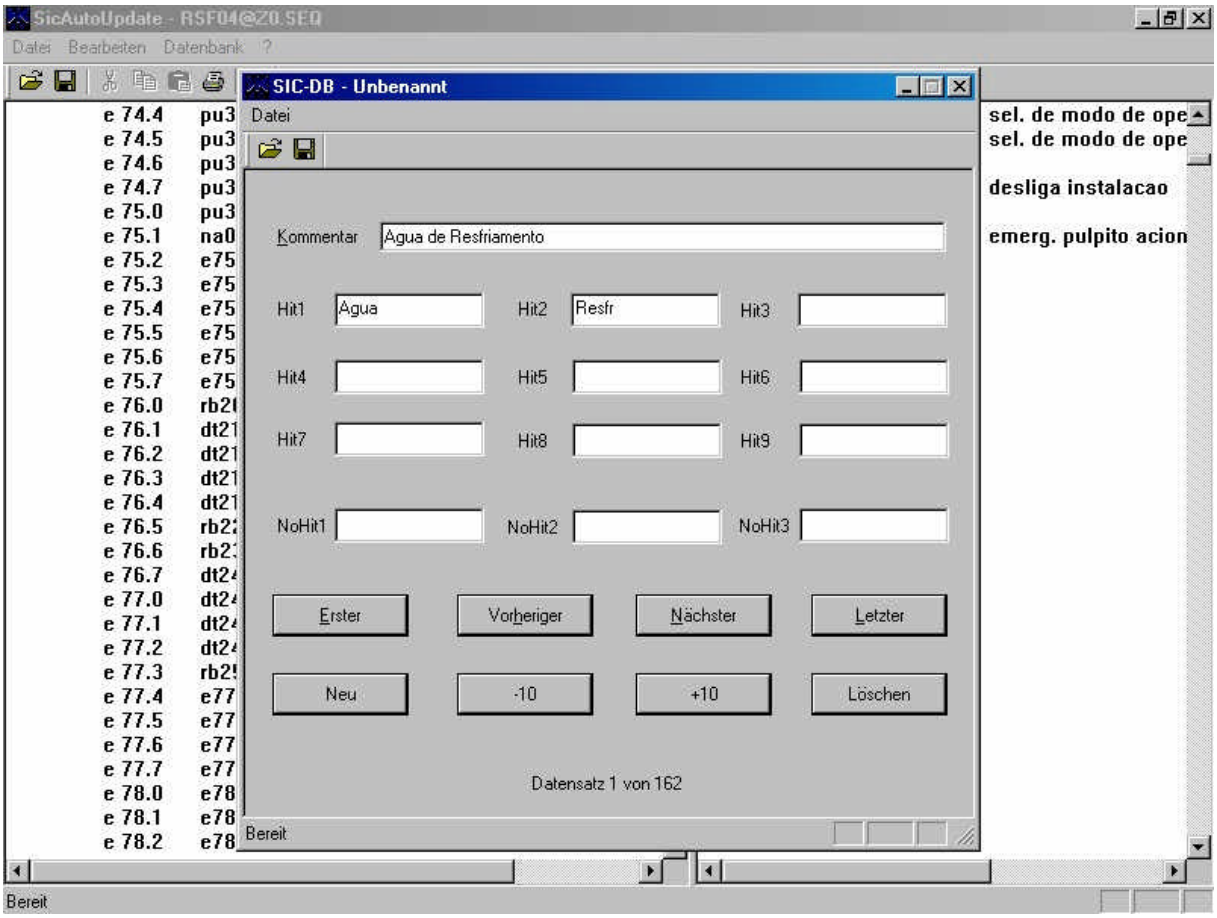

Abbildung 5.2: SIC-AU Datenbankoberfläche

Da die Anwendung im Wesentlichen ein Editor für ein speziell formatiertes Dateiformat, mit zusätzlicher, integrierter Parserfunktion ist, gibt es keine besonderen Funktionen, welche in der Oberfläche anzuwählen wären. So besteht die Menüleiste lediglich aus den Elementen "Datei", "Bearbeiten", "Datenbank" und "?"-Hilfe.

Abbildung 5.3: SIC-AU Menü

In der Toolbar ist neben den Standardfunktionen "Datei öffnen", "Datei speichern", "Ausschneiden, "Kopieren", "Einfügen" und "Drucken" auch ein Button zum Aktivieren der Parserfunktion "Go" enthalten. Der Aufruf der Parserfunktion, also der Überarbeitung der Symboltabelle, ist ebenfalls über den Menüteil "Bearbeiten"→"Go" zu erreichen.

Abbildung 5.4: SIC-AU Toolbar

Da die Anwendung recht einfach zu bedienen ist, ist für die Hilfe eine einfache Textbeschreibung ausreichend. Wird die Hilfe aufgerufen, so wird ein neues Dialogfenster erzeugt, in welchem die Informationen zur Programmbedienung als statischer Text positioniert sind.

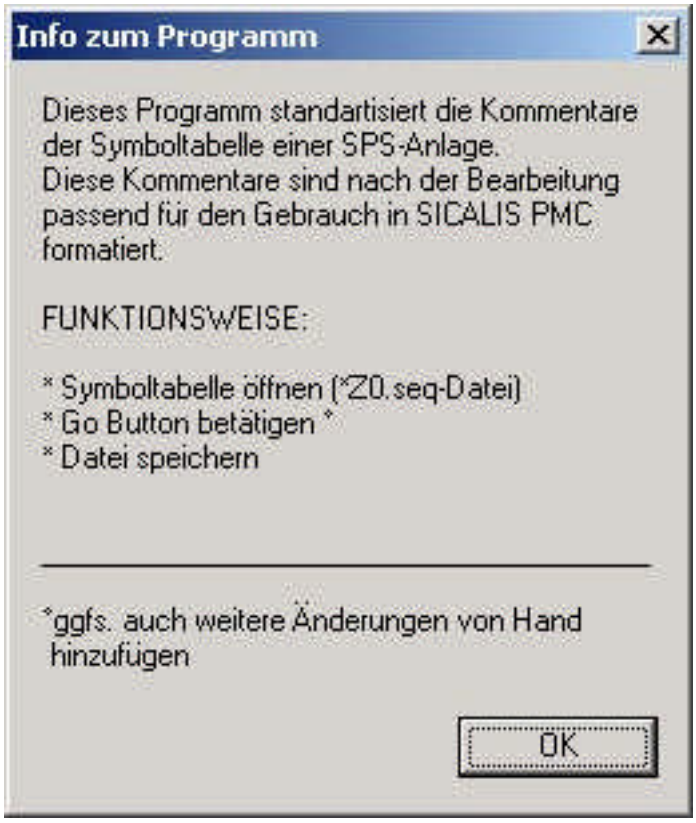

Abbildung 5.5: Hilfedialog zu SIC-AU

5.4 Datenbank für normierte Kommentare

Wenn die Daten der Anwendung (Datenbank als auch Symboltabellen) auf der Festplatte in Form einer Datei gespeichert werden, spricht man von Serialisierung. Wird der Anwendungszustand aus der Datei wiederhergestellt, bezeichnet man diesen Vorgang als Deserialisierung. Die Kombination der beiden Teile ergibt die Serialisierung von Anwendungsobjekten in Visual C++. Die Serialisierung in Visual C++-Anwendungen läuft über die Klasse CArchive. Diese Klasse agiert als Eingabe-/Ausgabe- (E/A) Stream für ein CFile-Objekt, wie es Abbildung 5.6 zeigt. Die C++-Streams gewährleisten einen effizienten Datenfluss in und aus einer Datei, die als Speicher der Anwendungsdaten dient.

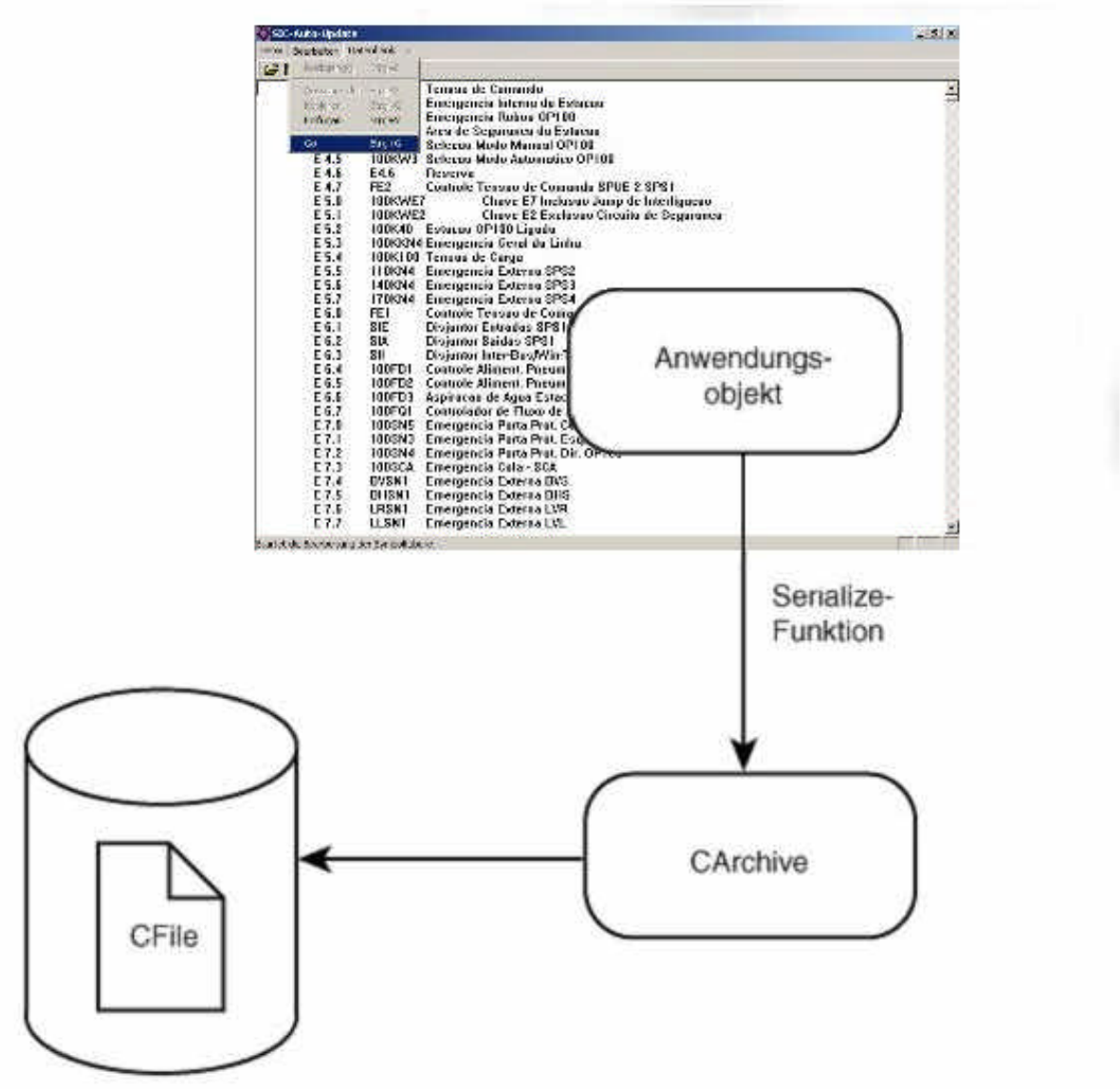

Abbildung 5.6: Struktur der Datenbank

Die Werte (also die Strings: Kommentar, Hit1- Hit9, NoHit1-NoHit3) der DB werden über einen einzigen Datenstrom in das CArchive-Objekt geschrieben bzw. daraus wiederhergestellt.

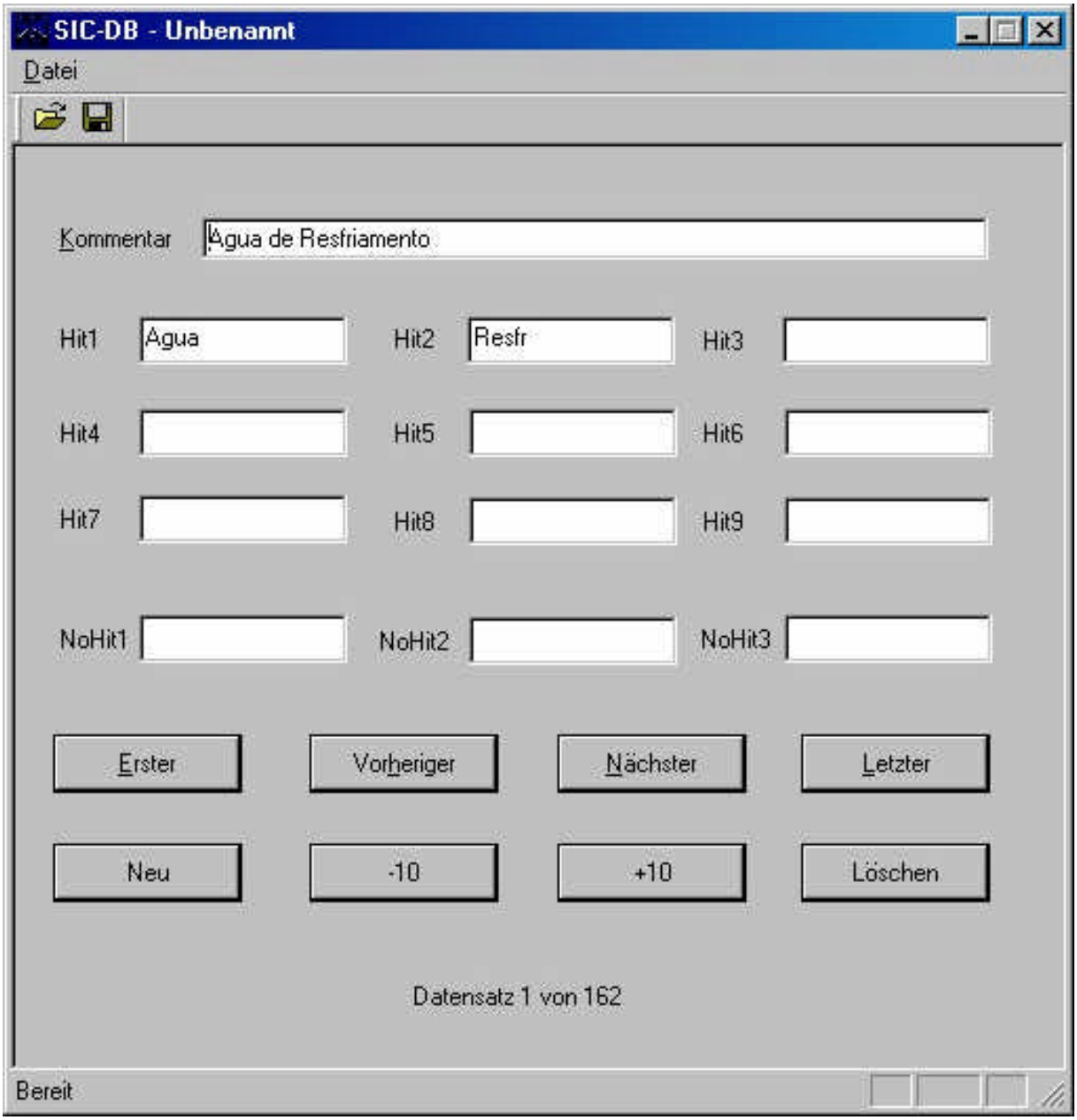

Abbildung 5.7: Dialog zum Verwalten der Datenbank

5.4.1 Aufbau der Datenbank

Bei der Datenbank handelt es sich um eine flache, lineare Datenbankanwendung. In der Datenbank sind normierte Kommentare gespeichert. Zusätzlich zu einem normierten Kommentar, sind weitere Informationen abgespeichert. Diese Informationen sind Worte und/oder Wortkürzel, welche in den bisher vorhandenen

Symboltabellen Anwendung gefunden hatten. Die Kürzel zu den jeweiligen Kommentaren wurden durch eine zweiwöchige Analyse von S5 Programmen ermittelt.
Die Datenbankanwendung zeigt nun einzelne Datenfelder an, die als Strings mit variabler Länge angelegt sind. Vor einem jeden Datenfeld, steht ein Bezeichner, welcher über die Art des Datenfelds Aufschluss gibt. Des Weiteren sind im unteren Bereich der Datenbankoberfläche Funktionsbuttons implementiert, über welche man durch die Datenbank navigieren, oder diese auch bearbeiten kann.
Auf die genaue Struktur und Anwendung der CStrings wird unter Abschnitt 5.5 Suchalgorithmus, genauer eingegangen.

5.4.2 Speichern von Daten in der Datenbank

Microsoft stellt in seiner Entwicklungsumgebung die Funktion Serialize mit den beiden Makros DECLARE_SERIAL und IMPLEMENT_SERIAL zum Speichern von Daten auf Datenträgern zur Verfügung.
Das Makro IMPLEMENT_SERIAL übernimmt hierbei drei Argumente. Das erste gibt wie beim Makro DECLARE_SERIAL den Klassennamen an, das zweite Argument übergibt den Namen der Basisklasse, von der ihre Klasse abgeleitet ist. Das dritte Argument bezeichnet eine Versionsnummer, mit der man bestimmen kann, ob eine in die Anwendung einzulesende Datei in der richtigen Version vorliegt.
Um Variablen mit den Steuerelementen der Datenbank zu verbinden, wurden der Ansichtsklasse (in diesem Fall CTeslaView) Variablen für die Steuerelemente hinzugefügt. Variablen für die angegebenen Steuerelemente:

Tabelle 5.1: Variablen für die Steuerelemente

ObjektName	**Kategorie**	**Typ**
IDC_EKOMMENTAR	m_sName	CString
IDC_SPOSITION	m_sPosition	CString
IDC_EHIT1	m_eHit1	CString
IDC_EHIT2	m_eHit2	CString
IDC_EHIT3	m_eHit3	CString
IDC_EHIT4	m_eHit4	CString
IDC_EHIT5	m_eHit5	CString
IDC_EHIT6	m_eHit6	CString
IDC_EHIT7	m_eHit7	CString
IDC_EHIT8	m_eHit8	CString
IDC_EHIT9	m_eHit9	CString
IDC_ENOHIT1	m_eNoHit1	CString
IDC_ENOHIT2	m_eNoHit2	CString
IDC_ENOHIT3	m_eNoHit3	CString

5.4.3 Eine Klasse für den Zugriff auf die Datenbank erstellen

Damit der Benutzer durch die Datensätze navigieren, Änderungen vornehmen, zusätzliche Datensätze einfügen oder Datensätze aus dem Rekordset entfernen kann, sind einige Methoden und Funktionen implementiert um auf die Klassenvariablen zugreifen zu können.

Tabelle 5.2: Klassenvariablen für die Klasse CNormiert

Name	Typ
m_eHit1	CString
m_eHit2	CString
m_eHit3	CString
m_eHit4	CString
m_eHit5	CString
m_eHit6	CString
m_eHit7	CString
m_eHit8	CString
m_eHit9	CString
m_eNoHit1	CString
m_eNoHit2	CString
m_eNoHit3	CString
m_eKommentar	CString
m_sPosition	CString

Listing 5.1: Der Konstruktor der Klasse CNormiert

```
CNormiert::CNormiert()
{
	m_eKommentar = "";
	m_eHit1 = "";
	m_eHit2 = "";
	m_eHit3 = "";
	m_eHit4 = "";
	m_eHit5 = "";
	m_eHit6 = "";
	m_eHit7 = "";
	m_eHit8 = "";
	m_eHit9 = "";
	m_eNoHit1 = "";
	m_eNoHit2 = "";
	m_eNoHit3 = "";
	m_sPosition = "";
}
```

5.4.4 Die Klasse CNormiert speichern

Als erstes ist hierfür in der Funktion Serialize die gleichnamige Funktion der Basisklasse aufzurufen. Hierbei werden alle gespeicherten Basisinformationen zuerst wiederhergestellt. Damit steht die erforderliche Unterstützung für die Klasse bereit, bevor die Variablen in Ihrer Klasse wiederhergestellt werden. Nach dem Aufruf der Funktion der Basisklasse wird angegeben, ob die Klassenvariablen zu lesen oder zu schreiben sind. Dazu dient die Methode IsStoring der Klasse CArchive. Diese Funktion liefert TRUE zurück, wenn das Archiv zu schreiben ist, und FALSE, wenn es zu lesen ist. Wenn die Funktion IsStoring den Wert TRUE zurückgibt, werden mit C++-E/A-Streams alle Klassenvariablen in das Archiv geschrieben. Liefert die Funktion FALSE, wird in Form von C++-Streams aus dem Archiv gelesen. Insbesondere ist darauf zu achten, dass die Variablen sowohl beim Lesen als auch beim Schreiben in derselben Reihenfolge erscheinen, da es ansonsten zu einer Inkonsistenz der DB und damit der gesamten Anwendung führen würde.

Listing 5.2: Die Funktion Serialize der Klasse CNormiert

```
void CNormiert::Serialize(CArchive &ar)
{
	//Funktion der Basisklasse aufrufen
	CObject::Serialize(ar);

	//Wird geschrieben?
	if (ar.IsStoring())
	{	//Alle Variablen in der richtigen Reihenfolge schreiben
		ar << m_eKommentar << m_eHit1<< m_eHit2<< m_eHit3<< m_eHit4<< m_eHit5<<
		m_eHit6<< m_eHit7<< m_eHit8<< m_eHit9<< m_eNoHit1<< m_eNoHit2<<
		m_eNoHit3<< m_sPosition;
	}
	else
	{
		//Alle Variablen in der richtigen Reihenfolge lesen
		ar >> m_eKommentar >> m_eHit1>> m_eHit2>> m_eHit3>> m_eHit4>> m_eHit5>>
		m_eHit6>> m_eHit7>> m_eHit8>> m_eHit9>> m_eNoHit1>> m_eNoHit2>>
		m_eNoHit3>> m_sPosition;
	}
}
```

Listing 5.3: Die Funktion Serialize der Klasse CSicAUDoc

```
void CNewTestDoc::Serialize(CArchive& ar)
{
	m_oaNormiert.Serialize(ar);
}
```

Um die Datensätze zwischenzuspeichern und die Navigation durch den Rekordset zu ermöglichen, wird ein Objektarray als Variable in der Dokumentklasse verwendet. Damit kann man bei Bedarf zusätzliche Datensatzobjekte hinzufügen. Die Navigation

implementiert man als Funktionen, die das erste, letzte, nächste oder vorherige Datensatzobjekt abrufen.
Schließlich kommt noch die Funktionalität, um bestimmen zu können, welchen Datensatz im Rekordset der Benutzer gerade bearbeitet. Zur Unterstützung dieser Funktionalität braucht die Dokumentklasse zunächst einmal zwei Variablen für das Objektarray und die aktuelle Datensatznummer im Array. Diese beiden Variablen bieten die erforderliche Unterstützung, um den Rekordset zu speichern und die Navigation zu ermöglichen.

Variablen für die Unterstützung des Rekordsets von CNormiert-Objekten sind:

Tabelle 5.3: Variablen der Dokumentklasse

Name	**Typ**
m_iCurPosition	int
m_oaNormiert	CObArray

5.4.5 Neue Datensätze in die Datenbank aufnehmen

Bevor durch den Rekordset navigiert werden kann, müssen neue Datensätze in das Objektarray aufgenommen werden. Nachdem ein Datensatz angelegt wurde, muss ein Zeiger darauf zurückgegeben werden, sodass die Ansichtsklasse direkt die Variablen im Datensatzobjekt aktualisieren kann. Auf diese Weise lässt sich die aktuelle Datensatznummer leicht anhand des Positionszählers leicht bestimmen.

Listing 5.4: Die Funktion AddNewRecord der Klasse CSICAUDoc

```
CNormiert* CSicAUDoc::AddNewRecord()
{
	CNormiert *pNormiert = new CNormiert();		//Ein neues CNormiert-Objekt erzeugen
	try
	{
		m_oaNormiert.Add(pNormiert);		//Neues Normiert in Objektarray hinzufügen
		SetModifiedFlag();			//Dokument als bearbeitet markieren
		m_iCurPosition = m_iCurPosition + 1;	//Neue Position festhalten
	}
	catch (CMemoryException* perr)		//Speicherausnahme aufgetreten?
	{
		AfxMessageBox			//Benutzer über schlechte Neuigkeiten informieren
		("Speichermangel", MB_ICONSTOP | MB_OK);
		if (pNormiert)			//Wurde Normiert-Objekt erzeugt?
		{
			delete pNormiert;	//Objekt löschen
			pNormiert = NULL;
		}
		perr->Delete();			//Ausnahmeobjekt löschen
	}
	return pNormiert;
}
```

5.4.6 Positionsermittlung

Um dem Benutzer die Navigation durch den Rekordset zu erleichtern, wird dem Benutzer mitgeteilt, wo er sich im Rekordset befindet. Zur Bereitstellung dieser Informationen müssen die aktuelle Datensatznummer und die Gesamtzahl der Datensätze aus dem Dokument ermittelt werden. Die Gesamtzahl der Datensätze im Objektarray wird durch die Größe des Arrays ermittelt.

Listing 5.5: Die Funktion GetTotalRecords der Klasse CSicAUDoc

```
int CSicAUDoc::GetTotalRecords()
{
	return m_oaNormiert.GetSize();		//Anzahl der Datensätze im Array zurückgeben
}
```

Die aktuelle Datensatznummer lässt sich auf die gleiche Weise ermitteln. Ein Positionszähler enthält die Nummer des Datensatzes, den der Benutzer gerade bearbeitet. Der Wert dieser Variablen wird an den Aufrufer zurückgegeben.

Listing 5.6: Die Funktion GetCurRecordNbr der Klasse CSicAUDoc

```
int CSicAUDoc::GetCurRecordNbr()
{
	return (m_iCurPosition + 1);		//Aktuelle Position zurückgeben
}
```

Nun zu der Funktion, welche den aktuellen Datensatz zurückgibt. Diese Funktion muss den Wert im Datensatzzeiger untersuchen, um sicherzustellen, dass der aktuelle Datensatz eine gültige Arrayposition ist. Nachdem geprüft wurde, ob die aktuelle Position gültig ist, kann die Funktion einen Zeiger auf den aktuellen Datensatz im Array zurückgeben.

Listing 5.7: Die Funktion GetCurRecord der Klasse CSicAUDoc.

```
CNormiert* CSicAUDoc::GetCurRecord()
{
	if (m_iCurPosition >= 0)			//Ist Datensatznummer gültig?
	{
		return (CNormiert*)
		m_oaNormiert[m_iCurPosition]; //Ja, aktuellen Datensatz zurückgeben
	}
	else
		return NULL;				//Nein, NULL zurückgeben
}
```

Um den ersten Datensatz im Array zurückzugeben, muss in dieser Funktion geprüft werden, ob das Array überhaupt Datensätze enthält. Sind Datensätze im Array vorhanden, wird der Datensatzzeiger auf 0 gesetzt und ein Zeiger auf den ersten Datensatz im Array zurückgeliefert.

Listing 5.8: Die Funktion GetFirstRecord der Klasse CSicAUDoc

```
CNormiert* CSicAUDoc::GetFirstRecord()
{
        if (m_oaNormiert.GetSize() > 0)                 //Enthält das Array Datensätze?
        {
                m_iCurPosition = 0;                     //Ja, zur Position 0 gehen
                return (CNormiert*)m_oaNormiert[0];     //Datensatz bei Position 0 zurückgeben
        }
        else
         return NULL;                                   //Keine Datensätze, NULL zurückgeben
}
```

Um zum nächsten Datensatz im Rekordset zu navigieren, wird der Datensatzzeiger inkrementiert und geprüft, ob das Ende des Arrays noch nicht überschritten ist. Wenn der Zeiger innerhalb der Arraygrenzen liegt, kann ein Zeiger auf den aktuellen Datensatz im Array zurückgegeben werden. Ist das Ende des Arrays überschritten, wird ein neuer Datensatz an das Array angehängt.

Listing 5.9: Die Funktion GetNextRecord der Klasse CSicAUDoc

```
CNormiert* CSicAUDoc::GetNextRecord()
{
        if(++m_iCurPosition<m_oaNormiert.GetSize())  //Arraygrenze überschr. nach Inkrementieren
        {                                            //des Positionszählers?
                return (CNormiert*)
                m_oaNormiert[m_iCurPosition];        //Nein, Datensatz an der neuen aktuellen
        }                                            //Position zurückgeben
        else
        {       m_iCurPosition = (m_oaNormiert.GetSize() - 1);
                return (CNormiert*)
                m_oaNormiert[m_iCurPosition];        //Wenn kein weiterer Eintrag in DB, letztes
        }                                            //Element anzeigen
}
```

In der Funktion, mit der zum vorherigen Datensatz im Array navigiert wird, sind verschiedene Prüfungen auszuführen. Erstens ist zu testen, ob das Array über Datensätze verfügt. Ist das der Fall, muss der Datensatzzeiger dekrementiert werden. Wird dieser Zeiger kleiner als null, wird der Datensatzzeiger auf 0 gesetzt. Somit wird auf den ersten Datensatz im Array gezeigt. Nun kann ein Zeiger auf den aktuellen Datensatz im Array zurückgegeben werden.

Listing 5.10: Die Funktion GetPrevRecord der Klasse CSicAUDoc

```
CNormiert* CSicAUDoc::GetPrevRecord()
{
	if (m_oaNormiert.GetSize() > 0)		//Enthält das Array Datensätze?
	{
		if (--m_iCurPosition < 0)	//Position nach Dekrementieren der aktuellen
						//Position kleiner als 0?
		m_iCurPosition = 0;		//Ja, Datensatzzeiger auf 0 setzen
		return (CNormiert*)
		m_oaNormiert[m_iCurPosition];	//Datensatz an der neuen aktuellen Position
						//zurückgeben
	}
	else
		return NULL;			//Keine Datensätze, NULL zurückgeben
}
```

Für die Funktion, die zum letzten Datensatz im Array navigiert, muss ebenfalls geprüft werden, ob Datensätze im Array vorhanden sind. Ist das der Fall, wird die aktuelle Größe des Arrays ermittelt und der Datensatzzeiger auf die um 1 verminderte Zahl der Datensätze im Array gesetzt. Dabei handelt es sich tatsächlich um den letzten Datensatz im Array, da die Zählung der Datensätze im Array bei 0 beginnt. Nachdem der Datensatzzeiger gesetzt ist, wird der Zeiger auf den letzten Datensatz im Array zurückgegeben.

Listing 5.11: Die Funktion GetLastRecord der Klasse CSicAUDoc

```
CNormiert* CSicAUDoc::GetLastRecord()
{
	if (m_oaNormiert.GetSize() > 0)
	{
		m_iCurPosition = (m_oaNormiert.GetSize() - 1);//Zur letzten Position im Array gehen
		return (CNormiert*)
		m_oaNormiert[m_iCurPosition];	//Datensatz auf dieser Position zurückgeben
	}
	else
	 return NULL;				//Keine Datensätze, NULL zurückgeben
}
```

Wird ein Rekord aus der Datenbank gelöscht, so muss darauf geachtet werden, dass danach wieder auf eine gültige (existente) Position im gesamten Rekordset gezeigt wird.

Listing 5.12: Die Funktion DelCurRecord der Klasse CSicAUDoc

```
bool CSicAUDoc::DelCurRecord()
{
	int anzRec = m_oaNormiert.GetSize();		//Anzahl der Einträge zwischenspeichern
	m_oaNormiert.RemoveAt( m_iCurPosition );	//Löschen eines Eintrages in der DB
	if ( m_iCurPosition == 0)			//auf valide Position im Recordset zeigen
		m_iCurPosition = 0;
	else
		m_iCurPosition--;
	if ( m_oaNormiert.GetSize() == (anzRec - 1))	//Anzahl der Einträge berichtigen
	{
		return true;
	}
	return false;
}
```

Wenn die Datenbankansicht geschlossen wird, muss der Speicher bereinigt werden. Dazu sind alle Objekte im Objektarray zu durchlaufen und zu löschen.

Listing 5.13: Die Funktion DeleteContents der Klasse CSicAUDoc

```
void CSicAUDoc::DeleteContents()
{
	CDocument::DeleteContents();
}
```

Wird in die Datenbankansicht gewechselt, so soll gleich der erste Eintrag der Datenbank angezeigt werden. Dazu muss der Datensatz gleich beim Wechseln der View aktualisiert werden.

Listing 5.14: Die Funktion OnOpenDocument der Klasse CSicAUDoc

```
BOOL CSicAUDoc::OnOpenDocument(LPCSTR lpszPathName)
{
	if (!CDocument::OnOpenDocument(lpszPathName))
		return FALSE;

	POSITION pos = GetFirstViewPosition();
	while (pos != NULL)
	{
		UpdateAllViews(NULL);
		CEditView* pView = (CEditView*)GetNextView(pos);
	}
	return TRUE;
}
```

Navigation und Bearbeitung sind ebenfalls in der Ansichtsklasse zu unterstützen. Die Unterstützung für den Rekordset ist in die Dokumentenklasse aufgenommen. Diese Funktionalität ist auch in der Ansichtsklasse zu realisieren, um durch die Datensätze zu navigieren, diese anzuzeigen und zu aktualisieren zu können.

5.4.7 Aktuellen Datensatz anzeigen

Zunächst ist der Ansichtsklasse die Funktionalität hinzugefügt, um den aktuellen Datensatz anzuzeigen. Da verschiedene Stellen in der Ansichtsklasse auf diese Funktionalität zurückgreifen, habe ich hierfür eine eigene Funktion erstellt. In dieser Funktion werden die aktuellen Werte aller Variablen im Datensatzobjekt geholt und in die Steuerelemente geschrieben, welche mit dem Fenster verbunden sind. Weiterhin werden die aktuelle Datensatznummer und die Gesamtzahl der Datensätze im Rekordset ermittelt und angezeigt.

Listing 5.15: Die Funktion PopulateView der Klasse CTeslaView

```
void CTeslaView::PopulateView()
{
	//Einen Zeiger auf das aktuelle Dokument holen
	CSicAUDoc* pDoc = GetDocument();
	if (pDoc)
		{
		//Aktuelle Datensatzposition in der Menge anzeigen
		m_sPosition.Format("Datensatz %d von %d", pDoc->GetCurRecordNbr(),
		pDoc->GetTotalRecords());
		}
	//Ist Datensatzobjekt gültig?
	if (m_pCurNormiert)
		{
		//Ja, alle Werte des Datensatzes holen
		m_eKommentar = m_pCurNormiert->GetKommentar();
		m_eHit1= m_pCurNormiert->GetHit1();
		m_eHit2= m_pCurNormiert->GetHit2();
		m_eHit3= m_pCurNormiert->GetHit3();
		m_eHit4= m_pCurNormiert->GetHit4();
		m_eHit5= m_pCurNormiert->GetHit5();
		m_eHit6= m_pCurNormiert->GetHit6();
		m_eHit7= m_pCurNormiert->GetHit7();
		m_eHit8= m_pCurNormiert->GetHit8();
		m_eHit9= m_pCurNormiert->GetHit9();
		m_eNoHit1= m_pCurNormiert->GetNoHit1();
		m_eNoHit2= m_pCurNormiert->GetNoHit2();
		m_eNoHit3= m_pCurNormiert->GetNoHit3();
		}
	//Anzeige aktualisieren
	UpdateData(FALSE);
}
```

5.4.8 Navigieren durch die Datenbank

Zunächst übernehmen Behandlungsroutinen für alle Navigationsschaltflächen die Navigation durch die Datenbank. Nachdem das Dokument zum entsprechenden Datensatz im Rekordset gelangt ist, wird die Funktion zur Anzeige des aktuellen Datensatzes aufgerufen.

Listing 5.16: Die Funktion OnBfirst der Klasse CTeslaView

```
void CTeslaView::OnBfirst()
{
	//Zeiger auf das aktuelle Dokument holen
	CSicAUDoc * pDoc = GetDocument();
	if (pDoc)
		{
		//Den ersten Datensatz aus dem Dokument holen
		m_pCurNormiert = pDoc->GetFirstRecord();
		if (m_pCurNormiert)
			{
			//Aktuellen Datensatz anzeigen
			PopulateView();
			}
		}
}
```

Für die Schaltfläche Letzter werden die gleichen Schritte ausgeführt, wie für die Schaltfläche Erster, jedoch wird aber die Funktion GetLastRecord des Dokumentobjekts aufgerufen.

Listing 5.17: Die Funktion OnBlast der Klasse CTeslaView

```
void CTeslaView::OnBlast()
{
	//Zeiger auf das aktuelle Dokument holen
	CSicAUDoc * pDoc = GetDocument();
	if (pDoc)
		{
		//Letzten Datensatz vom Dokument holen
		m_pCurNormiert = pDoc->GetLastRecord();
		if (m_pCurNormiert)
			{
			//Aktuellen Datensatz anzeigen
			PopulateView();
			}
		}
}
```

Die gleichen Schritte wiederholen sich für die Schaltflächen Vorheriger und Nächster, wobei die Funktionen GetPrevRecord bzw. GetNextRecord des Dokumentobjekts aufgerufen werden. Für die Schaltflächen +10 bzw. -10 werden ebenfalls GetPrevRecord bzw. GetNextRecord, wie Listing 4.18 und 4.19 es zeigen, aufgerufen.

Listing 5.18: Die Funktion OnBmais der Klasse CTeslaView

```
void CTeslaView::OnBmais()
{
	int counter = 0;
	while ( counter <= 9 )
	{
		counter = counter + 1;
		//Zeiger auf das aktuelle Dokument holen
		CSicAUDoc * pDoc = GetDocument();
		if (pDoc)
			{
				//Letzten Datensatz vom Dokument holen
				m_pCurNormiert = pDoc->GetNextRecord();
				if (m_pCurNormiert)
				{
				//Aktuellen Datensatz anzeigen
				PopulateView();
				}
			}
	}
}
```

Listing 5.19: Die Funktion OnBmenos der Klasse CTeslaView

```
void CTeslaView::OnBmenos()
{
	int counter = 0;
	while ( counter <=9 )
	{
		counter = counter + 1;
		//Zeiger auf das aktuelle Dokument holen
		CSicAUDoc * pDoc = GetDocument();
		if (pDoc)
			{
			//Letzten Datensatz vom Dokument holen
			m_pCurNormiert = pDoc->GetPrevRecord();
			if (m_pCurNormiert)
				{
				//Aktuellen Datensatz anzeigen
				PopulateView();
				}
			}
	}
}
```

5.5 Suchalgorithmus zum Finden normierbarer Kommentare

Der Suchalgorithmus ist im Prinzip sehr einfach aufgebaut. Während Zeilen von der Original-Symboltabelle eingelesen werden können, wird nach jeder eingelesenen Zeile ein Vergleich mit den einzelnen Rekords der Datenbank gestartet. Bei Treffern, also Übereinstimmungen mit den Hits bzw. den NoHits, werden Flags gesetzt. In einer Abfrage am Ende eines Vergleichs wird überprüft, ob alle Flags aktiviert wurden, damit der Vergleich einen wirklichen Treffer gemacht hat. Liefert diese Abfrage TRUE zurück, so wird der Kommentar der entsprechenden Zeile durch den normierten Kommentar in der Datenbank ersetzt. Gibt es keinen Treffer, wird der nächste Rekord der Datenbank eingelesen und mit dem Kommentar verglichen. Sollten nach sämtlichen Vergleichen mit den Rekords der Datenbank, keine gültigen Treffer erzielt worden sein, so wird der Kommentar der Original-Symboltabelle beibehalten.

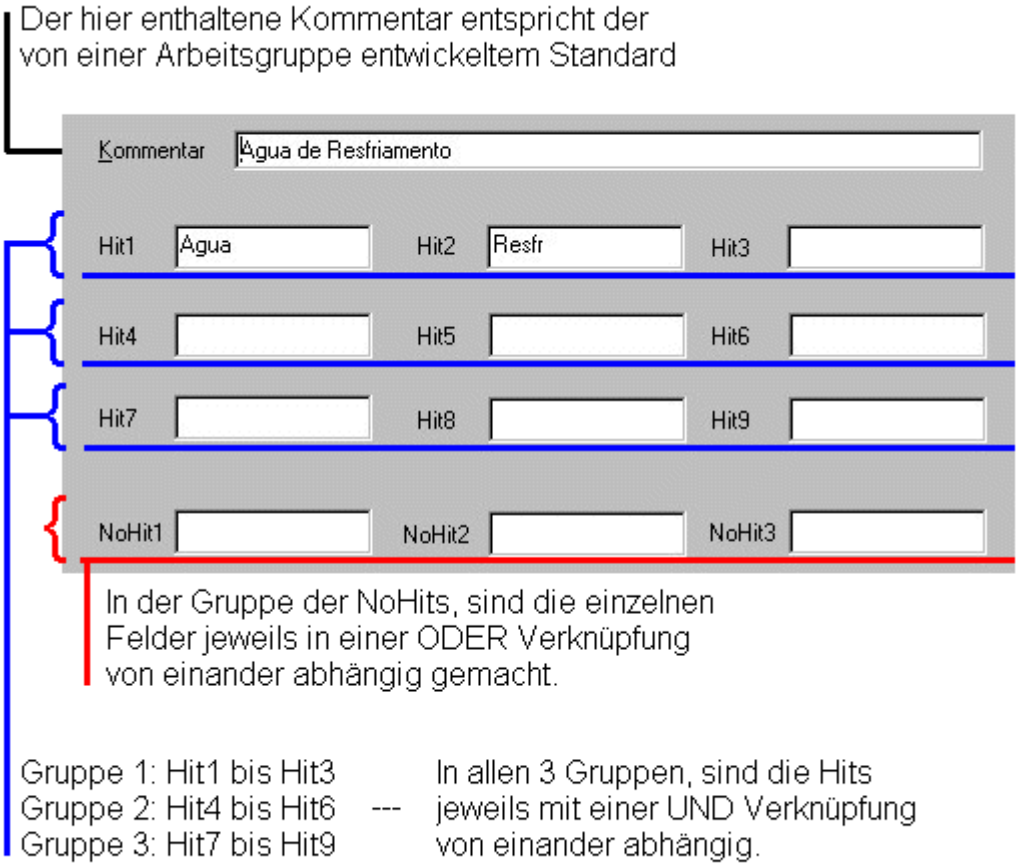

Abbildung 5.8: Gruppenbildung der Suchwörter

An dieser Stelle möchte ich nun die Struktur des Vergleiches eines Kommentars mit einem Rekord darstellen:
Ein Rekord besteht aus einem normierten Kommentar, 3 Hit-Gruppen und 1 NoHit-Gruppe (Abbildung 4.9). Dabei setzt sich sowohl eine Hit-Gruppe, als auch eine

NoHit-Gruppe aus je drei CStrings zusammen. Diese CStrings enthalten Wörter bzw. Wortkürzel welche signifikant für bestimmte Kommentare sind. Das Wortkürzel "Resfr" steht beispielsweise für "Resfriamento". Da es allerdings noch keine Richtlinien für die SPS-Programmierer beim Programmieren der S5-Anlagen gab, kann es sein, dass in einem Kommentar nicht "Resfriamento" sondern nur "Resfriam." steht. Das Wortkürzel "Resfr" ist also sensitiv genug, um bei einer Suche das Wort "Resifram." genau so zu finden, wie das Wort "Resfriamento".
Wird nun bei der Überarbeitung der originalen Symboltabelle der Kommentar "Agua de Resfriamento RIP 1" gefunden, so geschieht folgendes:
Im Vergleichsstring steht der originale Kommentar, also "Agua de Resfriamento RIP 1". Der Suchalgorithmus liest als nächstes den ersten Rekord ein. In der Datenbank, welche ich für die Anwendung erstellt habe, hätte dieses Set die Struktur:

Kommentar	:	"Agua de Resfriamento"
Hit 1	:	"Agua"
Hit 2	:	"Resfr"
Hit 3	:	leer
Hit 4	:	leer
:		
:		
Hit 9	:	leer
NoHit 1	:	leer
:		
NoHit 3	:	leer

Nun wird der Vergleichsstring mit Hit 1 verglichen. Hit 1 "Agua" kommt in "Agua de Resfriamento" vor und somit wird, das zu Hit 1 zugehörige, Hit1Flag gesetzt. Als nächstes vergleicht der Algorithmus den String mit Hit 2. "Resfr" wird in "Agua de Resfriamento" gefunden, und somit wird auch Hit2Flag gesetzt. Da Hit4 - NoHit 3 leer sind, werden diese Flags automatisch passend für den Entscheidungsvergleich, ob oder ob nicht die Bedingung zum Ersetzen erfüllt ist, gesetzt. Wie die Bezeichner der Suchwörter bzw. Suchkürzel bereits suggerieren, müssen Hit-Bedingungen erfüllt werden und bei NoHit Treffern ist die Erfüllung des Entscheidungsvergleiches gescheitert. Die Abfrage des Entscheidungsvergleiches ist durch eine UND/ODER Verknüpfung aller Flags realisiert.

```
*************************************************************************
*       Abfrage:                                                        *
*                                                                       *
*       if ( (  (Hit1Flag && Hit2Flag && Hit3Flag)      ||              *
*               (Hit4Flag && Hit5Flag && Hit6Flag)      ||              *
*               (Hit7Flag && Hit8Flag && Hit9Flag)      ) &&            *
*               (NoHit1Flag && NoHit2Flag && NoHit3Flag)    )           *
*                                                                       *
*************************************************************************
```

Für die Überarbeitung der Symboltabellen muss Text durchsucht und überarbeitet werden, deshalb basiert dieser Teil des Programms auf der MFC-Klasse CEdit. Deren Aufgabe ist es, Textdateien in einem Window darzustellen. Des Weiteren kann dient ein CEdit Window Text auch der Hinzufügung und Bearbeitung eines Textes. Nachdem der Suchalgorithmus die Symboltabelle überarbeitet hat, wird die neue Symboltabelle in einem Fenster angezeigt. Da dieses Fenster ein CEdit Fenster ist, sind bereits alle Möglichkeiten geboten, um den Text nun auch von Hand manuell nachzubearbeiten.

5.5.1 Der Code zum Suchalgorithmus

Listing 5.20: SicAU.cpp

```
while(strlen(p) != 0)                   //Symboltabelle solange einlesen, bis Ende erreicht
{
        index = strcspn(p, "\n");       //|Zeilenweise Symboltabelle in
        strncpy(buf, p, index);         //|buf zum Vergleich mit der DB speichern
        buf[index] = '\0';              //|ersetze \n durch \0 -> Endemarkierung
        anzZeile += index;              //|Dialogfenster aktualisieren
        myStatusDlg.m_sStatus.Format
         ("%d von %d Bytes überprüft",anzZeile,num);
        myStatusDlg.m_ctrlStatus.SetPos(anzZeile);
        myStatusDlg.m_sAnzahl.Format("Anzahl der Treffer: %d", t_zaehler);
        myStatusDlg.UpdateData(FALSE);
        if(strstr (buf, ";") != NULL)   //|Wenn Zeile mit ";" beginnt, Bearbeitung
        {                               //|überspringen und auf nächste Zeile zeigen
                neueST += buf;          //|Zeile bereits in neue Symboltabelle kopieren
                neueST += "\n";         //|NL
                p += index + 1;         //|
        }
        else
        {


                ////////////////////////////////////////////////////////////////////////////////////////////
                //-------------------------------------------------------------------------        ///
                //--------- Bearbeitung der Symboltabelle ---------------------------------        ///
                //-------------------------------------------------------------------------        ///
                // Eine in buf gespeicherte Zeile, wird mit den Daten der                        ///
                // aktuellen Recordposition verglichen. Bei                                      ///
                // Übereinstimmungen wird ein entsprechendes Flag                                ///
                // gesetzt. Am Ende des Verlgeichs einer Recordposition                          ///
                // mit der Zeile in buf, entscheidet die Verknüpfungslogik                       ///
                // darüber, ob ein Kommentar ersetzt werden muss                                 ///
                ////////////////////////////////////////////////////////////////////////////////////////////
                do
                {       //Record aus DB laden
                        pNormiert = (CNormiert*)pDoc->m_oaNormiert.GetAt(dbPosition);
                        ////////////////////////////////////////////////////////////////////////
                        //________Schlagwortgruppe 1_________________
                        ////////////////////////////////////////////////////////////////////////
                        strcpy(m_strBed, pNormiert->GetHit1());
                        strlwr(m_strBed);
                        if ( m_strBed[0] != 0)
                        {
                                if(strstr (buf, m_strBed) != NULL)
```

```
		{
			hit1flag = true;
		}
}

strcpy(m_strBed, pNormiert->GetHit2());
strlwr(m_strBed);
if ( m_strBed[0] != 0)
{
		if(strstr (buf, m_strBed) != NULL)
		{
			hit2flag = true;
		}
}
else
		hit2flag = true;

strcpy(m_strBed, pNormiert->GetHit3());
strlwr(m_strBed);
if ( m_strBed[0] != 0)
{
		if(strstr (buf, m_strBed) != NULL)
		{
			hit3flag = true;
		}
}
else
		hit3flag = true;
////////////////////////////////////////////////////////////////////////////
//_______Schlagwortgruppe 2_________________
////////////////////////////////////////////////////////////////////////////
strcpy(m_strBed, pNormiert->GetHit4());
strlwr(m_strBed);
if (( m_strBed[0] != 0) && hit1flag == FALSE)
{
		if(strstr (buf, m_strBed) != NULL)
		{
			hit4flag = true;
		}
}

strcpy(m_strBed, pNormiert->GetHit5());
strlwr(m_strBed);
if (( m_strBed[0] != 0) && hit1flag == FALSE)
{
		if(strstr (buf, m_strBed) != NULL)
		{
			hit5flag = true;
		}
}
else
		hit5flag = true;

strcpy(m_strBed, pNormiert->GetHit6());
strlwr(m_strBed);
if (( m_strBed[0] != 0) && hit1flag == FALSE)
{
		if(strstr (buf, m_strBed) != NULL)
		{
			hit6flag = true;
		}
}
else
		hit6flag = true;
////////////////////////////////////////////////////////////////////////////
//_______Schlagwortgruppe 3_________________
////////////////////////////////////////////////////////////////////////////
strcpy(m_strBed, pNormiert->GetHit7());
```

```
strlwr(m_strBed);
if (( m_strBed[0] != 0) && hit1flag == FALSE && hit4flag == FALSE)
{
	if (strstr (buf, m_strBed) != NULL)
	{
		hit7flag = true;
	}
}
strcpy(m_strBed, pNormiert->GetHit8());
strlwr(m_strBed);
if (( m_strBed[0] != 0) && hit1flag == FALSE && hit4flag == FALSE)
{
	if(strstr (buf, m_strBed) != NULL)
	{
		hit8flag = true;
	}
}
else
	hit8flag = true;
strcpy(m_strBed, pNormiert->GetHit9());
strlwr(m_strBed);
if (( m_strBed[0] != 0) && hit1flag == FALSE && hit4flag == FALSE)
{
	if(strstr (buf, m_strBed) != NULL)
	{
		hit9flag = true;
	}
}
else
	hit9flag = true;
////////////////////////////////////////////////////////////////////////////
//__Strings die nicht gefunden werden dürfen__
////////////////////////////////////////////////////////////////////////////
strcpy(m_strBed, pNormiert->GetNoHit1());
strlwr(m_strBed);
if ( m_strBed[0] != 0)
{
	if(strstr (buf, m_strBed) != NULL)
	{
		nohit1flag = false;
	}
}

strcpy(m_strBed, pNormiert->GetNoHit2());
strlwr(m_strBed);
if ( m_strBed[0] != 0)
{
	if(strstr (buf, m_strBed) != NULL)
	{
		nohit2flag = false;
	}
}
strcpy(m_strBed, pNormiert->GetNoHit3());
strlwr(m_strBed);
if ( m_strBed[0] != 0)
{
	if(strstr (buf, m_strBed) != NULL)
	{
		nohit3flag = false;
	}
}
////////////////////////////////////////////////////////////////
// VERKNUEPFUNGSLOGIK für die Flags
////////////////////////////////////////////////////////////////
if (((hit1flag && hit2flag && hit3flag) ||
    (hit4flag && hit5flag && hit6flag) ||
    (hit7flag && hit8flag && hit9flag))&&
    (nohit1flag && nohit2flag && nohit3flag)))
```

```
                    {
                        ////////////////////////////////////////////////////////////
                        //Wenn Treffer, ersetze Kommentar durch ein Normiertes und
                        //kopiere Zeile in neue Symboltabelle
                        ////////////////////////////////////////////////////////////
                        i=0;
                        t_zaehler++;
                        for(i= strlen(buf); i>=0; i--)
                        {
                            if(buf[i] == '\t')
                            break;
                        }
                        memset(pdest, '\0', 30);            //|
                        strncpy(pdest, buf, i+1);           //|Adressen und Kürzel kopieren
                        neueST += pdest;                    //|und in neue SymbolT einfügen
                        strcpy(m_strBed, pNormiert->        //|
                        GetKommentar());                    //|
                        neueST += m_strBed;                 //|Neues Kom. in neue SymbolT kopieren
                        neueST += "\r\n";                   //|CR, NL
                        changedLines += buf;                //|Welche Zeilen wurden geändert?
                        changedLines += "\n";               //|NL
                        modified = true;                    //|wurde eine Kommentar geändert - Flag
                    }
                    else
                    {
                        ////////////////////////////////////////////////////////////
                        //Wenn kein Treffer, kopiere Originalzeile in neue ST
                        ////////////////////////////////////////////////////////////
                        if(dbPosition+1 >= (pDoc->m_oaNormiert.GetSize()) )
                        {
                            neueST += buf;                  //|Original in neue SymbolT kopieren
                            neueST += "\n";                 //|NL
                            changedLines += "\r\n";
                        }
                    }
                ////////////////////////////////Flags zurücksetzen//////////////////////////////////////////////
                hit1flag=FALSE, hit2flag=FALSE, hit3flag=FALSE, hit4flag=FALSE, hit5flag=FALSE;
                hit6flag=FALSE;  hit7flag=FALSE, hit8flag=FALSE, hit9flag=FALSE, nohit1flag=TRUE;
                nohit2flag=TRUE, nohit3flag=TRUE;

                dbPosition++;                                   //nächster Eintrag in der DB

                }while(dbPosition != (pDoc->m_oaNormiert.GetSize() ) && modified == false );
                p += index + 1;                                 //von wo wird buf neu eingelesen
                dbPosition = 0;                                 //DB wieder auf ersten Record setzen
                modified = false;                               //Flag zurücksetzen
        }
}
```

6. Erweiterung des Personenrufdienstes - SIMAplus

Im Rohbau werden Meldungen an den Anlagen während der Produktion erzeugt. Diese Meldungen werden von SICALIS PMC empfangen und ausgewertet um sie zur Visualisierung zum Einen an digitale Anzeigetafeln weiterzuleiten, andererseits aber auch im Störfall an eine bestimmte Personengruppe mit Pagergeräten zu senden. Ein weiterer PC, der so genannte Pager-Server AMIS, erhält hierzu vom SICALIS Server über ein spezielles Modul des Pagerherstellers, alle zu versendenden Nachrichten. Das Empfangsmodul des Pager-Servers leitet die Nachricht an das Hauptmodul weiter, welches wiederum die Nachricht an die Sendeeinheit des Systems weiterleitet. Sollte nun im Fehlerfall eine weitere Person hinzugezogen werden müssen, so bietet dieses Pager System keine entsprechende Rufmöglichkeit.

Deshalb soll das vorhandene Pager-System nun so erweitert werden, dass individuelle Nachrichten von Personen an Personen mit Pager gesendet werden können.

6.2 Beschreibung des Personenrufsystems bei VW/Audi

Das bei VW/Audi do Brasil eingesetzte Personenrufsystem – auch Pager-System genannt - stammt von der Firma Ascom Interaction. Dieses besteht aus einem Pager-Server, einem Interfacebauteil und einer modularen Appliktationseinheit. Der Server befindet sich innerhalb des normalen firmeninternen Netzwerkes und ist daher von jedem Arbeitsplatzrechner aus via TCP/IP anzusprechen. Die Sendeantennen befinden sich ebenfalls innerhalb des Firmengeländes und sind somit auch für eventuelle Modifikationen erreichbar. Der Serverrechner ist ein Siemens PC mit Microsoft NT 4.0 als Betriebssystem und einer 3COM 10/100 Ethernet Karte als Schnittstelle zum Netzwerk. Außer der Pagerapplikation übt der Server keine weiteren Tätigkeiten aus.

Diese Pagersoftware besteht aus 3 Elementen:

Der "AMIS Alarmbearbeiter" ist das zentrale Element im AMIS System. Er übernimmt eingehende Alarme und Meldungen (aus dem AMIS Eingabe-Modul), gibt diese an das entsprechende Ausgabe-Modul weiter und überwacht den Alarmierungsablauf. Des Weiteren können über dessen Oberfläche Detailinformationen zu aufgetretenen Alarmen angezeigt werden. Ein Modul zum „manuellen Rufauslösen“ ist ebenfalls enthalten.

Das AscomPSA Modul ist das Ausgabemodul im Alarm Management und Informations System AMIS. Es dient zur Alarmierung von Personen über Personensuchanlagen. Die Ansteuerung der Hardware (Ascom ESPA4.4.4 Interfacebaugruppe) erfolgt über eine serielle RS232 Schnittstelle und dem Protokoll ESPA4.4.4.

Das AMIS-KCOM-Modul ist das Eingabemodul von AMIS. Es dient zur Kopplung an das System SICALIS PMC der Firma Siemens. Über das KCOM Modul werden Störddaten von SICALIS PMC übernommen und dem AMIS System zur weiteren Verarbeitung zur Verfügung gestellt. Die Kommunikation zwischen SICALIS PMC und dem KCOM-Modul erfolgt über eine Netzwerkverbindung und dem TCP/IP Protokoll.

6.3 Alternative Lösungsansätze

Um es zu ermöglichen, von einem Arbeitsplatzrechner aus Nachrichten an einen Pager zu versenden, wird zum einen auf jeden Fall eine Netzwerkverbindung zum Pager-Server benötigt. Für die Art und Weise wie die Kommunikation stattfinden sollte, gab es nach ersten Analysen vier verschiedene Lösungsansätze:

Lösung 1: Ein weiteres Eingabemodul, welches parallel zu KCOM läuft und Meldungen von PC's empfangen kann, wird der AMIS Software hinzugefügt.

Lösung 2: Der Sender einer Nachricht wird als SICALIS PMC Sender "getarnt", sodass AMIS die Meldung über das KCOM Modul empfangen kann.

Lösung 3: Eine parallele Anwendung wird auf den Pager-Server installiert, die mit AMIS um die RS232 Schnittestelle konkurriert, wobei beide Programme abwechselnd auf die Schnittstelle zugreifen können.

Lösung 4: Eine Software wird auf dem Pager-Server implementiert, welche die "manuelle Rufauslösung"-Funktion von AMIS nutzen wird.

Lösung 1:
Es gab keinerlei Informationen, auf welche Art und Weise ein Eingabemodul seine Daten an das Hauptmodul übermittelt. Um diese Lösung umsetzten zu können, hätte zuerst mit Hilfe von speziellen Tools die Kommunikation zwischen AMIS und dem Eingabemodul analysiert werden müssen. Ob dann noch weitere Einstellungen am Hauptmodul gemacht werden hätten müssen, damit ein weiteres Eingabemodul akzeptiert wird, wäre dadurch noch nicht geklärt worden. Da es auch zum Einbinden weiterer Module in der Dokumentation zum AMIS System keinerlei Informationen gab, schied diese Lösung also aus.

Lösung2:
Um die Nachrichten eines PC an den Pager-Server so aussehen zu lassen, als würden sie von SICALIS stammen, hätten zunächst mit einem Portsniffer die Meldungen von SICALIS an AMIS geloggt und analysiert werden müssen. Nach Auswertung dieser Daten hätte man die Nachrichten, die man selbst verschicken möchte, in ein identisches Netzwerkprotokoll einbinden und verschicken müssen. Leider konnte ich anhand der Dokumentation von AMIS nicht herausfinden, ob es

hierbei zu eventuellen Kollisionen und somit zu einer Systeminstabilität kommen würde, wenn weitere "SICALIS Meldungen" von anderen Rechnern beim Pager-Server eingehen. Somit schied auch diese Lösung aus.

Lösung 3:
Diese war nach ersten Analysen recht viel versprechend, musste dann jedoch ebenfalls ausscheiden, da ich im Internet keine Spezifikation zum ESPA 4.4.4 Protokoll finden konnte. Also die Spezifikation, auf welche Art und Weise die Kommunikation zwischen dem Ausgabemodul und der Interfacehardware stattfindet. Die einzige Information, welche ich von einer Firma aus der französischen Schweiz nach E-Mail Kontakt bekam, war dass dieser Standard, welcher von AMIS benutzt wird, veraltet sei und ich lediglich in Büchern noch eventuell eine Spezifikation von ESPA 4.4.4 finden könnte. Außerdem hätte das Sharing der RS232-Schnittstelle zwischen AMIS und meiner Software dazu führen können, dass einerseits Meldungen verloren gehen oder andererseits AMIS instabil würde.

Lösung 4:
Somit blieb also nur noch die Möglichkeit, die "manuelle Rufauslösung"-Funktion von AMIS zu nutzen. Das Problem dieser Lösung bestand darin, dass zuerst die Struktur von AMIS analysiert werden musste. Für eine solche Analyse wurde ein Tool wie Microsoft Spy++ benötigt, um in Erfahrung bringen zu können, welche Eigenschaften die Oberfläche hat und welche Bezeichnungen bzw. ID die Steuerelemente der AMIS Anwendung in der internen Kommunikation haben.

6.4 Realisierung

Auf den Pager-Server wird ein weiteres Programm installiert. Dieses ist der Server Part, der Client-Server-Lösung. Eine Client-Server Software sind Programme oder Module, welche über eine Verbindung miteinander kommunizieren können. Diese Verbindung kann über die RS232 Schnittstelle, über das Internet oder auch über ein Ethernet Netzwerk hergestellt werden.

Die Client-Server-Lösung funktioniert weiters so:
Als erstes muss der Server-Dienst gestartet sein. Dann kann über ein gestartetes Client-Modul eine Nachricht an den Serverteil via TCP/IP gesendet werden. Das Server-Modul empfängt die Nachricht und bindet diese in die Auftragsliste des bereits vorhandenen Pager-Systems (AMIS) ein. Dies geschieht durch Manipulation der zentralen AMIS Einheit. Das Server Modul sucht auf dem Server-Rechner nach dem Toplevel Fenster des AMIS Pager-Systems und generiert einen Handle darauf. Mit diesem Handle werden Nachrichten in die Auftragliste der zu versendenden Nachrichten mit eingehängt. Die Abarbeitung der Auftragsliste ist Aufgabe des bereits vorhandenen Systems und muss von der Client-Server Anwendung daher nicht mehr weiters beachtet werden.

Auf der Client Seite hat der Benutzer die Möglichkeit in einem Eingabefenster, die gewünschte Nachricht einzugeben und über eine Adressliste dem Empfänger zuzuweisen.

6.5 Nutzen der Funktionen von AMIS

Wie bereits erwähnt, musste zunächst einmal das Hauptmodul analysiert werden, damit entsprechende Handle auf das Hauptfenster und die Funktionen des Programms erzeugt werden konnten. Für diese Analyse kam das Freeware Tool "Winspector Spy" der Firma Gipsysoft zum Einsatz (Abbildung 5.1). Dieses Tool bietet per Drag und Drop eines Markierungselements die Möglichkeit, Eigenschaften eines Fensters oder eines Fensterelements anzuzeigen. Die Informationen, welche das Tool zurückliefert sind:

- der Klassennamen, auf welchem die Anwendung basiert,
- die Eigenschaften des Windows,
- die Parameter mit denen ein Window initialisiert wurde und
- die ID, welche zur Identifikation eines Fensters oder eines Fensterelements programmintern verwendet wird.

So konnte mit Hilfe von Winspector Spy nach und nach das Gerüst des Hauptfensters von AMIS Alarmbearbeiter, dem Hauptmodul von AMIS, aufgezeichnet werden. Nachdem die notwendigen Funktionen und Klassen des Hauptfensters erfasst waren, musste das Dialog-Fenster "manuelle Rufauslösung" ebenfalls analysiert und ein Gerüst hierfür erstellt werden.
Mit den nun vorhandenen Werten, wo vor allem die ID's der Funktionen wichtig sind, kann mit Hilfe von Windowsfunktionen das Hauptfenster "AMIS Alarmbearbeiter" ausfindig gemacht werden. Wird auf das gefundene Fenster ein Handle erzeugt, so können über diesen Handle Nachrichten an das Fenster gesendet werden. Für das Programm ist es dann so, als würde ein Benutzer auf der Windowsoberfläche mit Maus und Tastatur die Funktionen des Programms aktivieren.

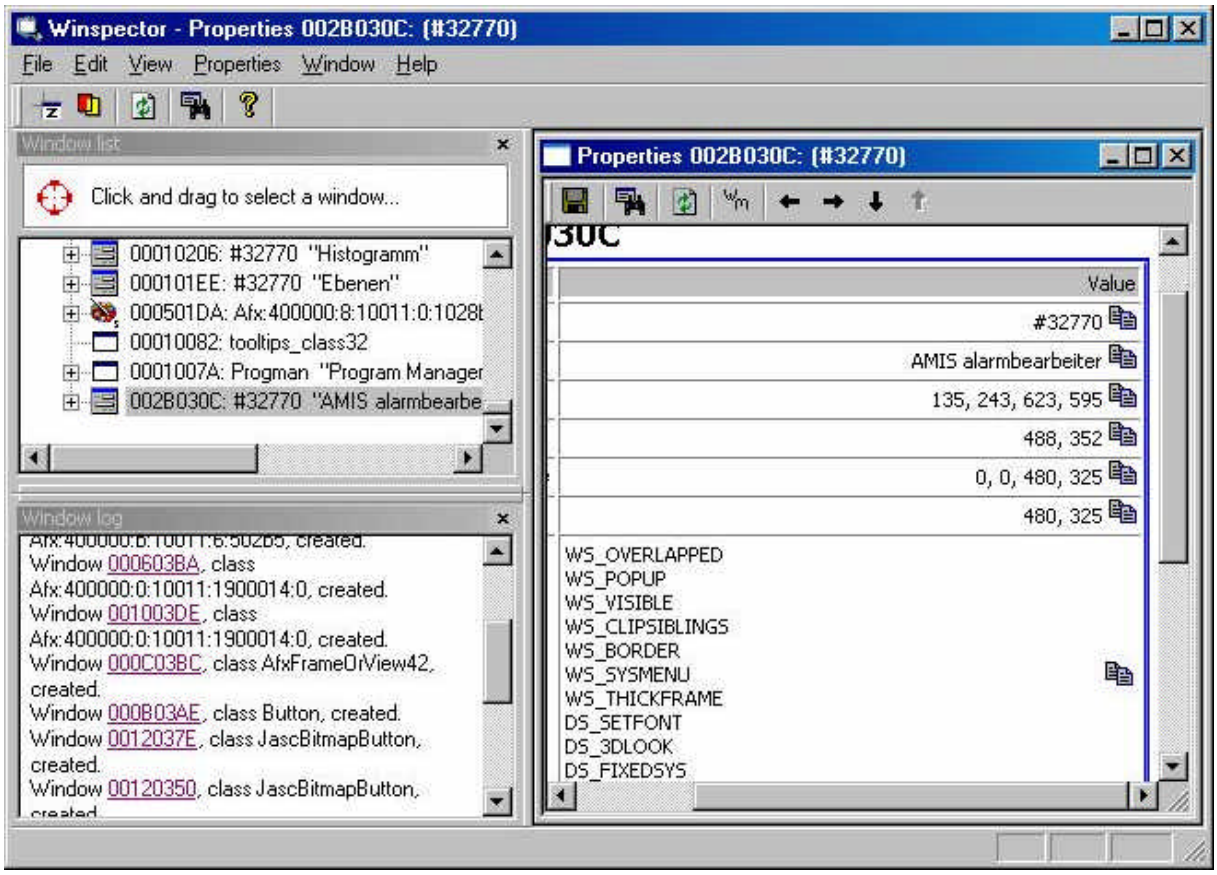

Abbildung 6.1: Winspector

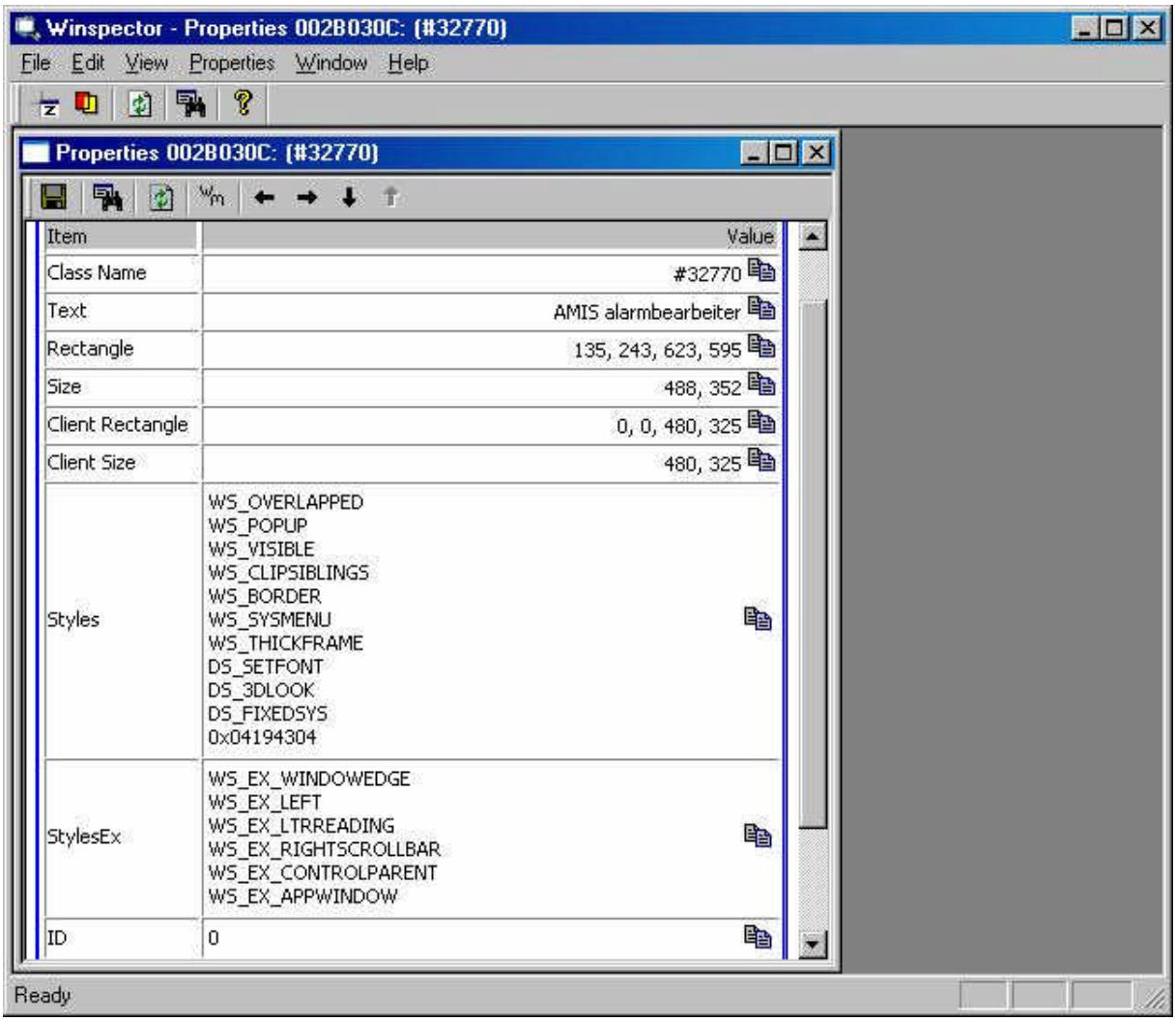

Abbildung 6.2: Winspector Eigenschaftenfenster

Abbildung 6.2 zeigt das Eigenschaftenfenster, welches Winspector ausgibt, nachdem man ein Programm oder ein Programmelement auf der Windowsoberfläche markiert hat. In diesem Fall handelt es sich um die Eigenschaften des AMIS Alarmbearbeiters. Zu sehen ist hier die Klasse "#32770" auf welcher die Anwendung basiert. Dieser Klassenname ist spezifisch für CDialog basierende Anwendungen. So verwenden die beiden SIMAplus Programme ebenfalls diesen Klassennamen.

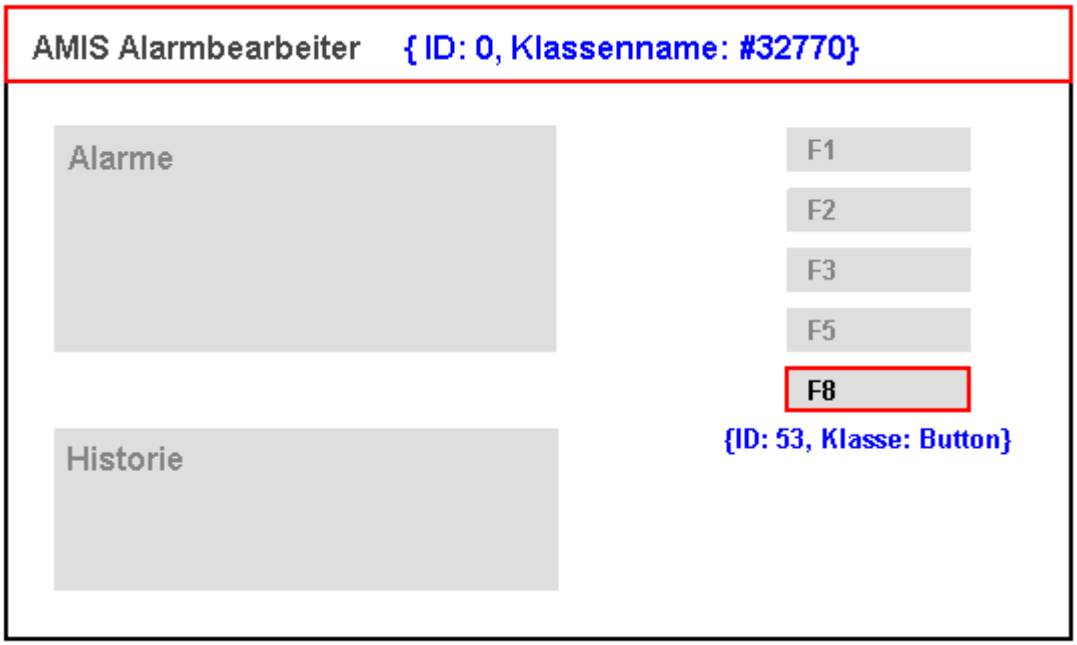

Abbildung 6.3: AMIS Alarmbearbeiter

Die Abbildungen 6.3 und 6.4 zeigen nun die Gerüste, welche sich aus den Informationen von Winspector, haben erstellen lassen.

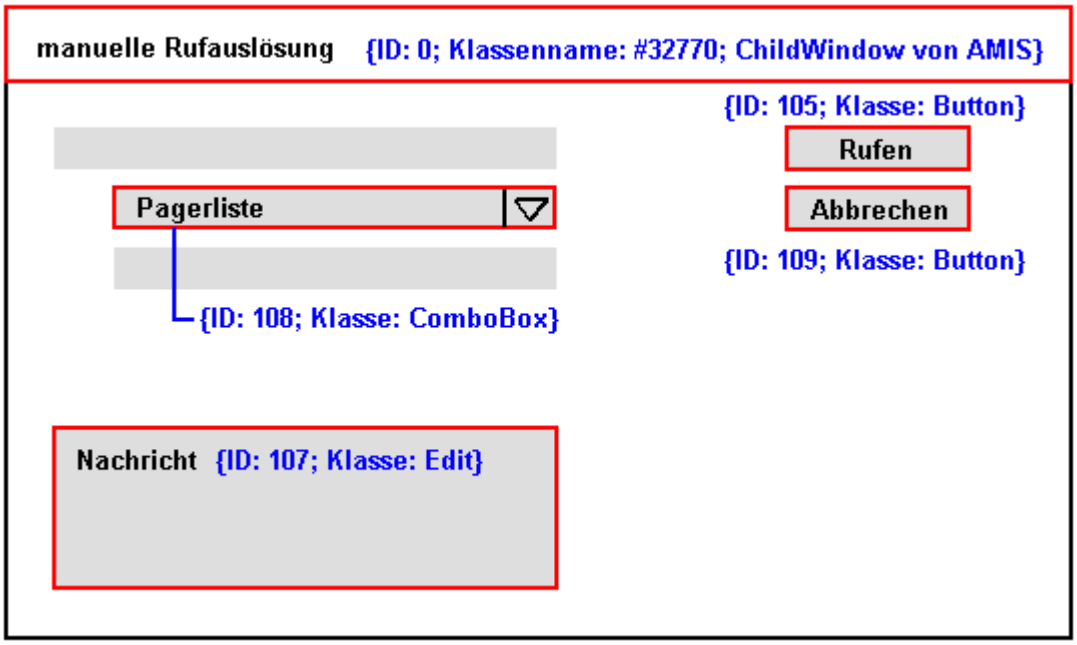

Abbildung 6.4: AMIS – manuelle Rufauslösung

Damit während der Programmentwicklung das Testen der Client-Server Applikation nicht zum Absturz von AMIS oder gar des gesamten Pager-Servers führen kann, habe ich anhand der, durch Winspector Spy, erfassten Eigenschaften von AMIS eine Dummyoberfläche erstellt. Dieser Dummy besitzt dieselben Grundfunktionen und

ID's, wie das AMIS Alarmbearbeiter-Programm (Abbildungen 6.5 und 6.6). Da diese Dummyoberfläche auf demselben Rechner, auf dem die Client-Server Applikation programmiert wurde, gestartet werden konnte, bestand somit auch die Möglichkeit auf Seiten des Clients, des Servers und der Dummyoberfläche während der Programmentwicklung zu debuggen.

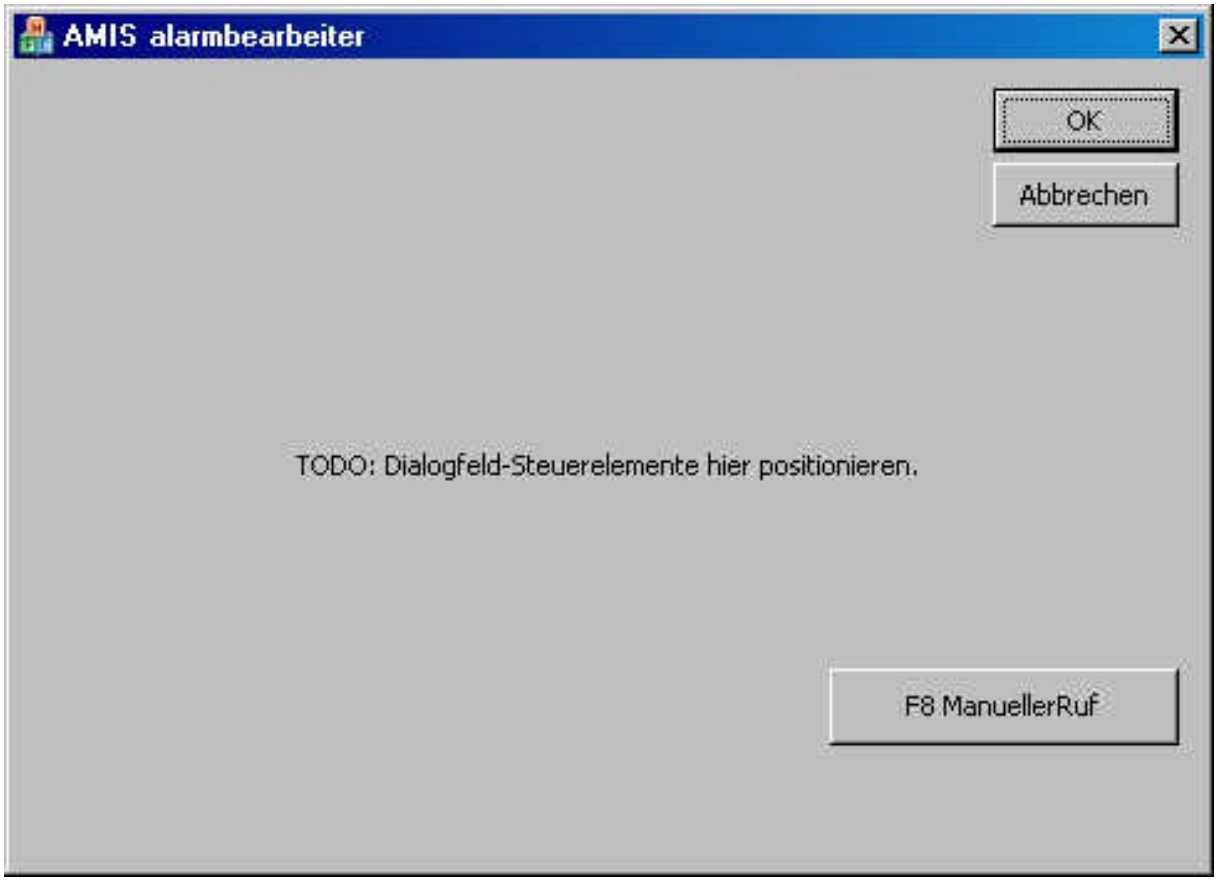

Abbildung 6.5: Dummyoberläche

Abbildung 6.6: Dummyoberfläche manuelle Rufauslösung - Dialog

6.7 Client-Server Lösung

Die Client-Server Software muss folgende Funktionen beinhalten:

- TCP/IP Kommunikation zwischen Client und Server
- Management von Clients die sich mit dem Server verbinden
- Server-Part: muss Handle auf AMIS erzeugen
- Aufrufen von AMIS Funktionen über Handle

Um eine bessere Performance der Applikation zu gewährleisten, kamen noch weitere Funktionen hinzu.

1) Nach dem Starten des Server-Parts auf dem Pager-Server zunächst einmal das Fenster "manuelle Rufauslösung" aufgerufen, um die darüber zugänglichen Pagernutzer auszulesen.

2) Die Liste mit den Pagernutzern wird über eine weitere Funktion des Server-Programms an jeden Client gesendet, der sich mit dem Server verbindet.

Durch diese beiden Funktionen ist es dann nicht notwendig, eine Empfängerliste bei der Clientinstallation anzulegen und diese zu aktualisieren, da die Liste programmintern nach jedem Connect auf der Client-Seite erzeugt wird.

6.7.1 Netzwerkkommunikation

Für die Netzwerkkommunikation wird eine TCP/IP Verbindung hergestellt. Für einen Verbindungsaufbau, muss der Server bereits gestartet sein. Andernfalls bekommt der Client, bei dem Versuch eine Verbindung zum Server herzustellen, eine Fehlermeldung. Die IP Adresse des Servers und der Port für die Verbindung sind programmintern eingestellt. Die Socketverbindung, über welche Client und Server kommunizieren basiert auf Funktionen eines Netzwerktools, welches auf www.codeproject.com gehostet wird. In der Client-Server Applikation wird für das Verbinden und Senden von Nachrichten über ein Netzwerk auf die von Microsoft bereitgestellte MFC-Klasse CSocket zurückgegriffen. CSocket ist eine abgeleitete Klasse von CAsyncSocket und erbt über diese die Funktionalität der Windows Socket API. Über die Windowsklassen CSocketFile und CArchive bietet CSocket die Möglichkeit, Daten zu senden und zu empfangen. Zum Senden und Empfangen von Socketnachrichten, kann auf die Funktionen Receive, Send, SendTo, Connect und Accept zugegriffen werden.

Der Verbindungsaufbau in der Anwendung geschieht somit nach folgendem Schema. Der Client macht eine Connect-Anfrage beim Server. Ist der Server bereit die Verbindung anzunehmen, so führt dieser die Accept Funktion aus. Nun wird ein CSocketFile erzeugt und mit einem CSocket-Objekt im CSocketFile-Konstruktor

assoziiert. Außerdem wird ein CArchive-Objekt angelegt, welches die Nachricht enthält. Wie bei CSocket, so muss auch für CArchive das Objekt im CArchive.-Konstruktor assoziiert werden. Nach dem Versenden einer Nachricht, müssen alle Objekte wieder gelöscht werden und Client und Server sind somit bereit, neue Socketmeldungen zu senden/empfangen.
Da der Server in der Lage ist, mehrere Client-Verbindungen auf einmal zu akzeptieren, wird noch ein Punkt in der Kommunikation zwischen Client und Server eingefügt. So muss beim Verbindungsaufbau, nachdem eine Connect-Anfrage vom Client beim Server eingegangen ist, der Client sich zunächst mit einer ID beim Server registrieren. Über diese ID wird für den Client ein Thread in der Threadverwaltung erzeugt. Jeder, für einen Client erzeugter, Thread hat Zugriff auf die Funktionen des Servers. Erst nachdem für den Client ein Thread erzeugt ist, kann nun auch eine Meldung vom Client gesendet werden. Der Server empfängt diese Meldung und wertet diese aus. Diese Meldung ist ein dynamisches Array, welches alle gängigen C++ Formate beinhalten kann. Bisher verwende ich bei SIMAplus jedoch lediglich ein CString. In diesem CString sind nun für den Server verschiedene Informationen enthalten. So enthält die Meldung als erstes die Kennung des Pagers, an welchen die Nachricht gesendet werden soll und als zweites die Nachricht selbst. Das Array, welches auch den CString enthält, trägt in sich eine Kennung, welche eine Nachricht bei der Client-Server Kommunikation identifiziert.

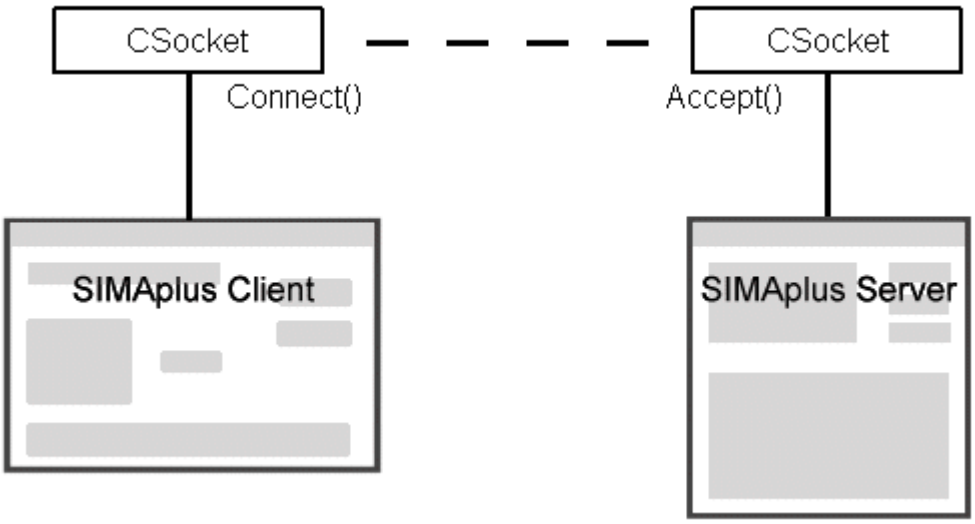

Abbildung 6.7: Netzwerkkommunikation mit CSocket

Abbildung 6.7 zeigt einen Client, der eine Connect-Anfrage beim Server über das Netzwerk macht. Ist der Server bereit, die Verbindung anzunehmen, so beantwortet er die Anfrage mit einem Accept.

Nun wird im Socket User Management für die ID, mit welcher sich der Client verbunden hat, ein Thread angelegt und verwaltet. (Abbildung 6.8)

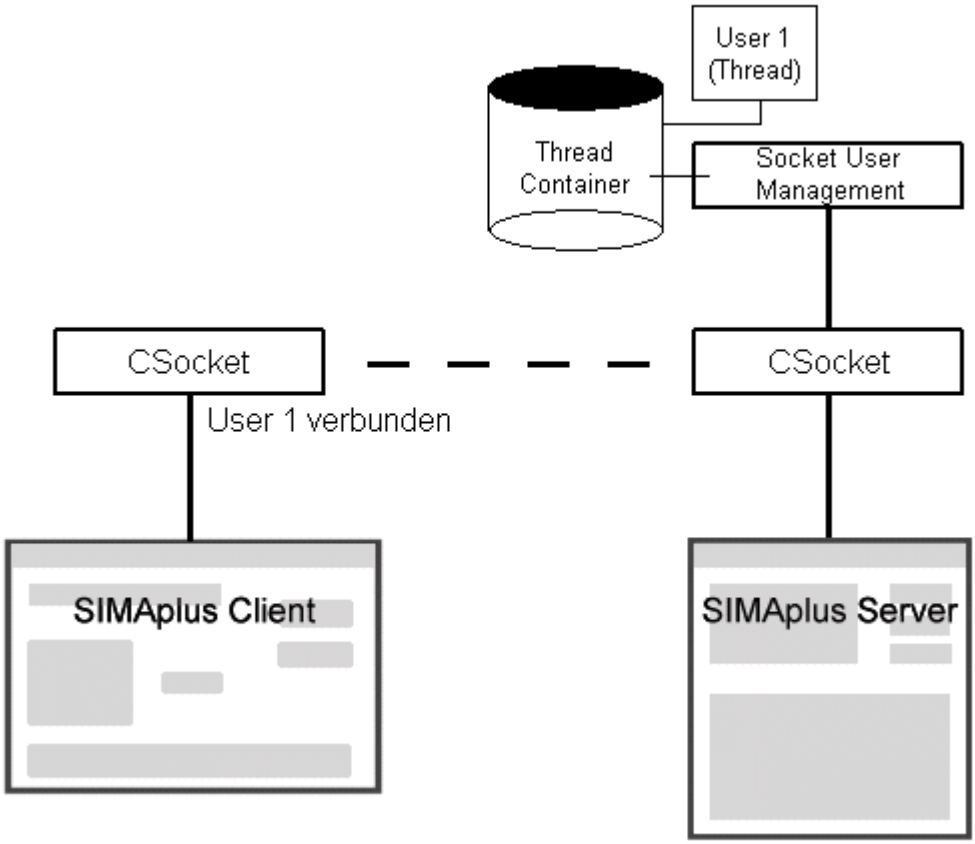

Abbildung 6.8: Verwalten von Benutzern

Abbildung 6.9 zeigt nun, einen weiteren Client, der eine Verbindungsanfrage zum Server macht. Der Server akzeptiert die Anfrage und wird über das Socket User Management einen weiteren Thread anlegen. Der Server hat also eine Multiuser Schnittstelle. Er ist in der Lage, gleichzeitig mehrere Verbindungen zu mehreren Clients zu verwalten.

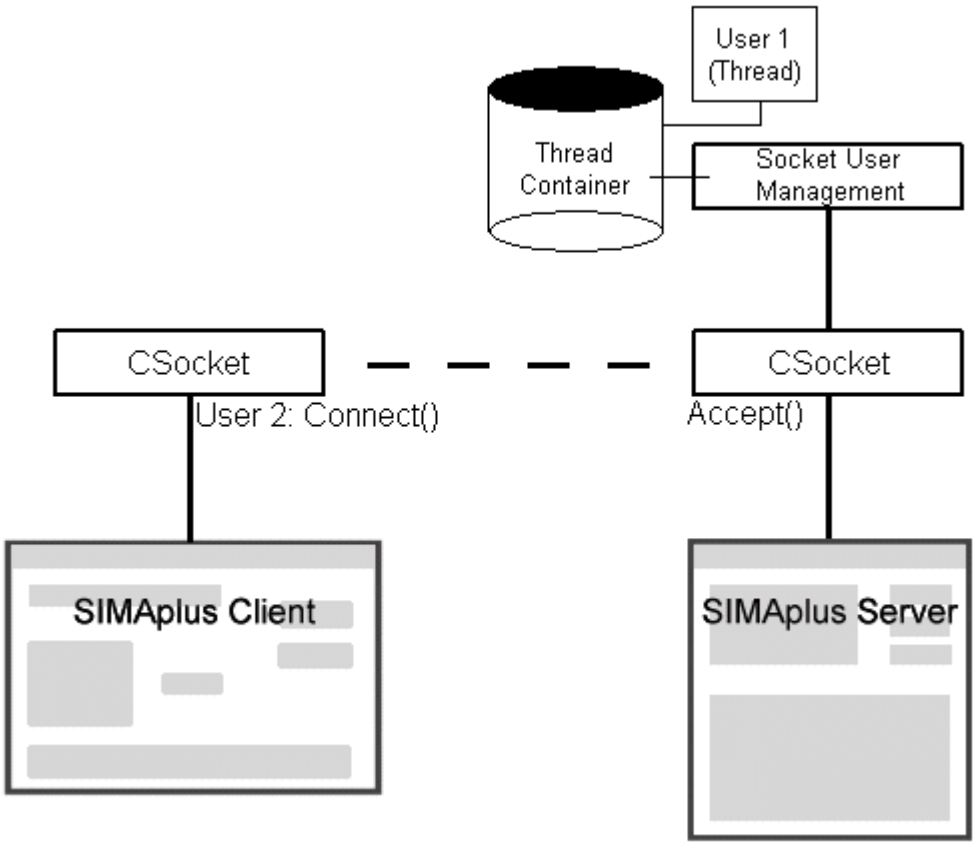

Abbildung 6.9: Durchführen einer Verbindungsanfrage

In Abbildung 6.10 ist nun zu sehen, dass für den zweiten Client ein weiterer Thread im Threadcontainer angelegt wurde. Sobald sich ein Client wieder vom Server trennt, wird der entsprechende Thread, im Usermanagement der Serverapplikation, wieder gelöscht.

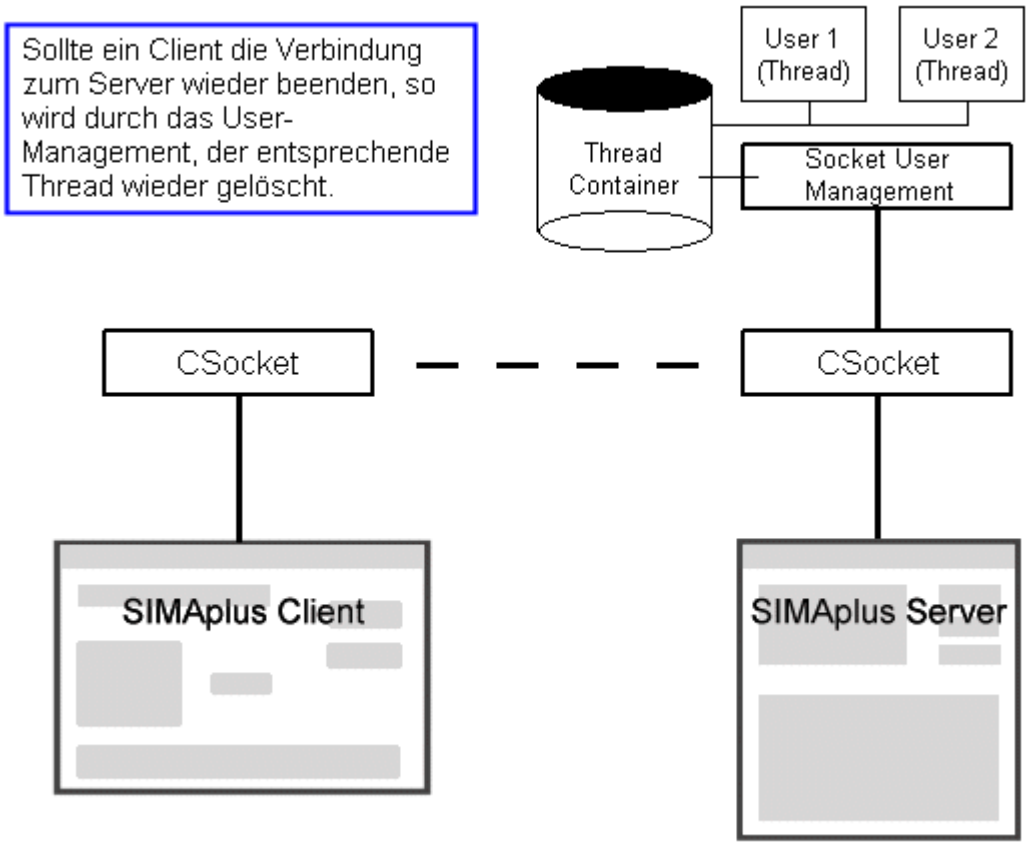

Abbildung 6.10: Anlegen eines Threads

6.7.2 Ablauf

In der gesamten Struktur der Client-Server-Applikation ist also zum Versenden einer Pagernachricht ein fester Ablauf vorgegeben:

1. SIMAplus Server wird gestartet.
2. SIMAplus Server überprüft ob AMIS läuft; wenn nicht, stoppt SIMAplus Server automatisch wieder.
3. Kann AMIS gefunden werden, so erzeugt SIMAplus Server einen Handle auf das Hauptfenster von AMIS, über welchen nun auf Funktionen von AMIS zugegriffen werden kann.
4. Nachdem ein Handle auf das AMIS Hauptfenster existiert, ruft SIMAplus Server die "manuelle Rufauslösung"-Funktion von AMIS auf, um dort die Liste der Pagernutzer auszulesen.
5. SIMAplus Server speichert die Pagernutzerliste intern, als CString, ab.
6. Macht ein SIMAplus Client einen Verbindungsaufbau zum SIMAplus Server, so sendet der Server den CString mit den Pagernutzern zurück an den Client.
7. Der Client wandelt den empfangenen CString um und initialisiert mit den enthaltenen Daten die eigene ComboBox-Liste.
8. Über diese ComboBox-Liste kann nun auf der Client-Seite ein Empfänger für eine Pagernachricht ausgewählt werden. Und im Textfeld kann eine Nachricht für den jeweiligen, ausgewählten Empfänger eingegeben werden.
9. Wird die Nachricht vom Client abgesendet so wertet der SIMAplus Server diese nach dem Empfang aus, aktiviert bei AMIS die manuelle Rufauslösung und initialisiert mit den Daten aus dem CString die verschiedenen Felder der manuellen Ruffunktion.

6.7.3 Erläuterung zur verwendeten Programmiertechnik

Zur Realisation der Funktionen und um es zu ermöglichen Oberflächen anderer Programme zu manipulieren, kommen nachfolgend aufgeführte Funktionen und Operationen zum Einsatz.

- HWND
- CWnd
- PostMessage
- FindWindow
- FindWindowEx
- Attach/Detach
- GetDlgItem
- ReplaceSel
- GetCount

HWND → Ein Windows Fenster wird im der Betriebssystemverwaltung über einen so genannten Windowhandle – HWND – identifiziert. Dieser wird erzeugt, sobald ein CWnd Objekt durch den Aufruf der Create-Funktion, der CWnd Klasse, generiert wurde.

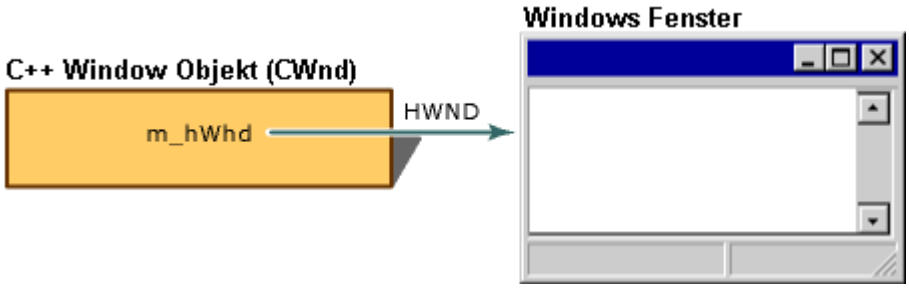

Abbildung 6.11: HWND und CWnd

Die CWnd Klasse beinhaltet die Grundfunktionalität aller MFC Klassen. CWnd und seine abgeleiteten Windowsklassen stellen Konstruktoren, Destruktoren, und Memberfunktionen zur Verfügung, um Objekte zu initialisieren und um Zugang zu einem eingekapselten HWND Objekt zu erhalten. Ebenfalls sind in CWnd Memberfunktionen enthalten, um Nachrichten an Fenster zu senden, welche den Fensterstatus und Fenstereigenschaften verändern können. Mit CWnd können Child-Fenster erzeugt werden. Für jedes Child-Fenster existiert ein so genanntes CWnd-Objekt. Dieses CWnd-Objekt ist zwar unabhängig von einem Windows Fenster, dennoch sind Fenster und Objekt miteinander verbunden. Erzeugt wird das CWnd-Objekt durch den Konstruktor und Destruktor von CWnd. Das Windows Fenster hingegen, wird durch die Create Memberfunktion von der CWnd Klasse erzeugt. Um ein Fenster zu löschen, wird ein virtueller Destruktor von CWnd verwendet. Mit der Funktion DestroyWindow wird das Windows Fenster gelöscht, ohne dass das CWnd Objekt hierfür auch gelöscht wird.
Jedes CWnd Objekt enthält eine Message-Map, in der Windows Nachrichten zwischengespeichert werden. Die Nachrichten in der Message-Map werden automatisch direkt an entsprechende CWnd Memberfunktionen zur Abarbeitung weitergeleitet.

Um den Handle von AMIS zu finden, kommt die FindWindow-Funktion zum Einsatz. Diese Funktion kann entweder nach der Klasse suchen, auf welcher die Anwendung basiert, oder auch nach dem Namen des Fensters. Sollte kein Fenster gefunden werden, welches den Sucheigenschaften entspricht, so liefert die Funktion den Wert NULL zurück. Andernfalls erhält man den HWND.
Die Funktion durchsucht nur die Toplevel Fenster in Windows. Child Fenster werden nicht mit berücksichtigt.

Um ein Child Fenster auszumachen, muss von der Funktion FindWindowEx Gebrauch gemacht werden. Diese Funktion entspricht der FindWindow Funktion, sucht aber im Gegensatz zu dieser nur abgeleitete Fenster. Beide Funktionen sind nicht Case-Sensitiv, d. h., es wird nicht auf die Groß- und Kleinschreibung bei der Suche nach den Fensternamen geachtet.

Wurde das passende Fenster ausgemacht, so ist es zwingend dieses Fenster über die Attach Funktion zugängig zu machen. Die Attach Funktion hängt ein Windows Fenster an ein CWnd Objekt. Mit dessen Hilfe kann nun auf Elemente dieses Fenster zugegriffen werden.
Wird das angehängte Fenster nicht mehr benötigt, so muss der Handle auf dieses Objekt mit der Detach-Funktion wieder getrennt werden.

So lange nun aber ein Fenster angehängt ist, können mit weiteren Funktionen alle Elemente eines Fensters manipuliert werden. So selektiert man zunächst mit der Funktion GetDlgItem das gewünschte Element und ändert mit weiteren Funktionen dessen Eigenschaften. So kann ein Button beispielsweise mit PostMessage-Funktionen aktiviert werden. Der Gebrauch von PostMessage ist hier der SendMessage Funktion vorzuziehen, da SendMessage auf einen Rückgabewert des manipulierten Elements wartet. Da aber SIMSplus Server ein externes Programm manipuliert und nicht ein eigenes Child-Fenster, gibt es daher auch keine Rückgabewerte des Elements. PostMessage sendet die Nachricht ohne auf einen Rückgabewert zu warten.
Bei der Manipulation von Elementen ist jedoch darauf zu achten, dass man zunächst einen passenden Pointer erzeugt, welcher es ermöglicht Eigenschaften eines Elements zugängig zu machen. Soll also beispielsweise eine Combobox manipuliert werden, so ist der Pointertyp, welcher hierfür anzulegen ist, ein Pointer der Klasse CComboBox. (Beispiel: `CComboBox* pWnd1 = (CComboBox*)m_hdlWindow.GetDlgItem(108);`)
Um auf ein Edit-Element zuzugreifen, wird ein Pointer der Klasse CEdit benötigt. (Beispiel: `CEdit* pWnd1 = (CEdit*)m_hdlWindow.GetDlgItem(107);`)

Abhängig davon, ob nun eine Combobox oder ein Editbereich manipuliert werden soll, bietet der einem Element zugewiesene Zeiger die Möglichkeit, elementabhängige Funktionen aufzurufen. Also können so beispielsweise für ein Combobox Element die Funktionen ShowDropDown oder SetCurSel, oder für ein Edit Element die Funktion ReplaceSel aufgerufen werden.

6.8 Oberfläche und Funktionen des Serverprogramms

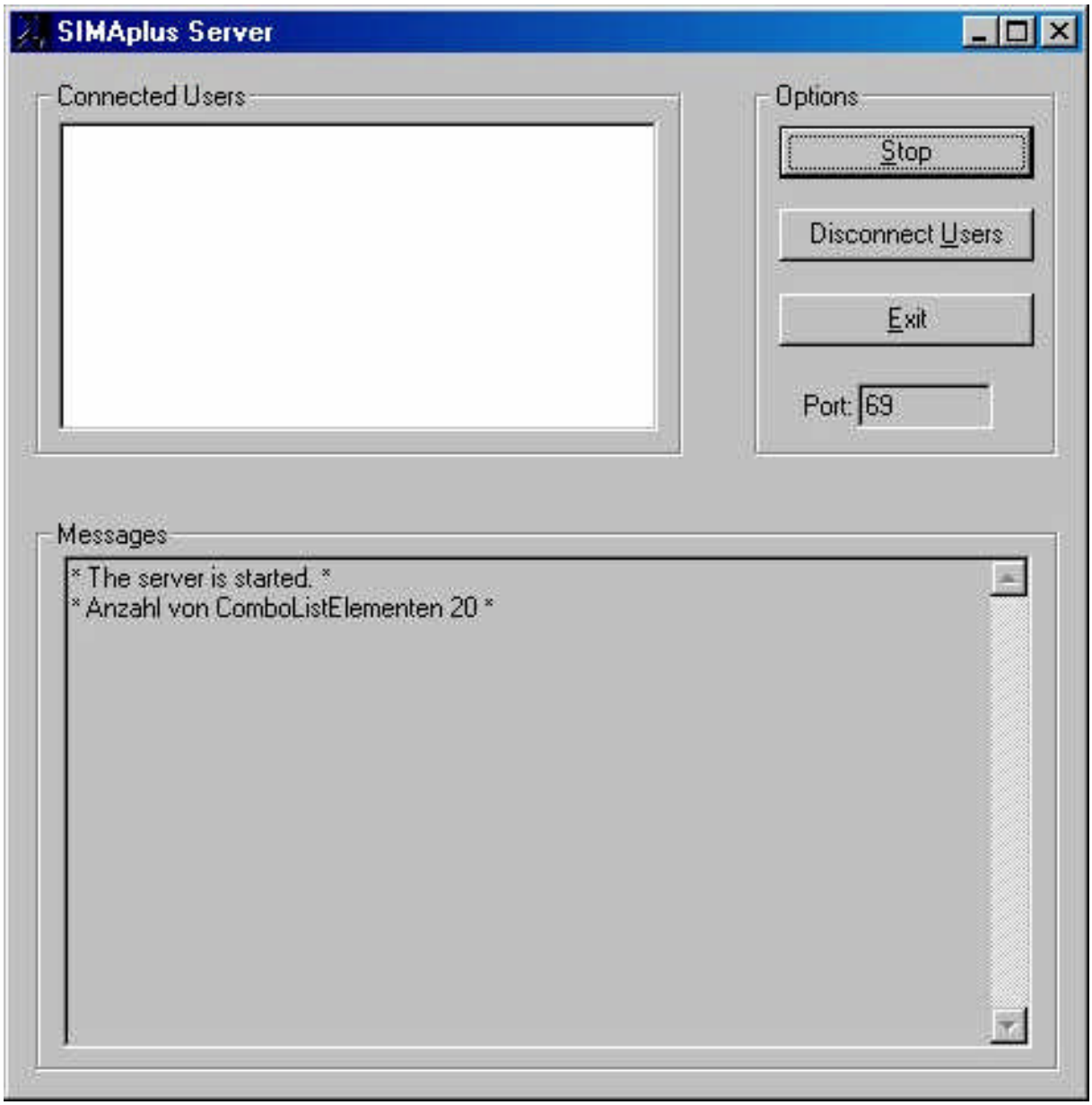

Abbildung 6.12: SIMAplus Server

Um die Oberfläche von SIMAplus Server zu generieren, basiert die Anwendung auf der MFC-Klasse "CDialog". CDialog ist die Basisklasse, welche genutzt wird um Dialoge anzuzeigen. Solche Dialoge können auf zwei unterschiedliche Weisen aufgebaut werden. Entweder modal, also so dass der User mit dem Dialog interagieren kann. Oder modeless wenn nur Informationen angezeigt werden sollen, ohne dass der User mit agieren muss. Wie jedes andere Fenster, so kommuniziert auch ein CDialog Fenster über Messages mit Windows. Diese Eigenschaft ist sehr wichtig, da gerade auch AMIS auf dieser Klasse basiert. Dies garantiert, dass eine Kommunikation mit Meldungen zwischen SIMAplus und AMIS möglich ist.

In der OnButtonStart Funktion des SIMAplus Servers, ruft das Programm das AMIS-Child-Window "manuelle Rufauslösung" auf, um die ComboBox auszulesen, welche die Liste der Pagernutzer enthält (siehe Listing 6.1). Durch diese Funktionalität bleibt die Pagernutzerliste, mit welcher SIMAplus arbeitet, immer aktuell.

Listing 6.1: OnButtonStart Funktion von SIMAServerDlg

```
void CSIMAServerDlg::OnButtonStart()
{
	hdlMainWin = *FindWindow(classname, 0);
	CWnd* pWnd = GetWindow(GW_HWNDLAST);
	pWnd0 = GetWindow(GW_HWNDFIRST);

	do	//|Suche AMIS Mainwindow
	{
		pWnd0 = pWnd0->GetNextWindow(GW_HWNDNEXT);
	} while(pWnd0 != pWnd && pWnd0->GetDlgItem(53) == 0);

	if (UpdateData(TRUE))
	{
		if (IsStarted())		//Statusüberprüfung
		{
			Stop();		//Stoppen der Serveranwendung
			UpdateUI();
			AddSystemText((LPCSTR)IDS_STOPPED);
		}
		else
		{
			//Falls AMIS nicht läuft
			if(!pWnd0->GetDlgItem(53) && StartListening(m_lPort))
			{
				//Start der Serverandwendung verhindern
			AddSystemText("Please, start AMIS first!");
				Stop();
				UpdateUI();
				AddSystemText((LPCSTR)IDS_STOPPED);
			}




			//ansonsten kann die Serveranwendung gestartet werden
			else if(StartListening(m_lPort))
			{
				UpdateUI();
				AddSystemText((LPCSTR)IDS_STARTED);
				//Ist bereits das "manuelle Ruf..."-Fenster offen?
				hdlWindow = *FindWindowEx(0, 0, 0, winname);

				while(!hdlWindow)		//Wenn nicht, dann eins öffnen
				{
					pWnd0->GetDlgItem(53)->
					PostMessage(WM_LBUTTONDOWN , 0, 0);
					pWnd0->GetDlgItem(53)->
					PostMessage(WM_LBUTTONUP , 0, 0);
					Sleep(100);
					//und Handle auf "manuelle Rufauslösung" erzeugen
					hdlWindow = *FindWindowEx(0, 0, 0, winname);
			}
				//Handle ans Fenster anhängen
				m_hdlWindow.Attach(hdlWindow);

				//Pointer auf ComboBox, welche die Nummern der Pager enthält
				CComboBox* pWnd1 = (CComboBox*)
							m_hdlWindow.GetDlgItem(108);
```

```
                        //Anzahl der ListBox Einträge zählen
                        int count = pWnd1->GetCount();
                CString listCount;
                        //Int nach CString Wandlung
                        listCount.Format("%d", count);
                        AddSystemText("Number of pagerusers: " + listCount);
                        int j = 0;

                        //|Erzeugen eines CStrings, welcher sämtliche Pagerbenutzer enthält.
                        while(j < count)
                        {
                                pWnd1->SetCurSel(j);
                                pWnd1->GetLBText(j, comboBoxList);
                                comboListComplete += comboBoxList;
                                //Trennzeichen zwischen den einzelnen Pagerbenutzern
                                comboListComplete += '§';
                        j++;
                }
                        //"manuelle Rufauslösung"-Fenster kann nun wieder geschl. werden
                        m_hdlWindow.GetDlgItem(109)->PostMessage
                                (WM_LBUTTONDOWN , 0, 0);
                        m_hdlWindow.GetDlgItem(109)->PostMessage
                                (WM_LBUTTONUP , 0, 0);
                        //Handle lösen
                        m_hdlWindow.Detach();
                }
        }
    }
}
```

Aus den ComboBox-List-Elementen wurde in der Funktion "OnButtonStart" ein CString erzeugt, welcher nun bei einem Connect eines Clients an diesen gesendet wird. Der Client, welcher diesen String erhält, wertet die Informationen aus und initialisiert mit diesen Daten seine eigene ComboBox-Liste. Dies geschieht in der Case-Abfrage "SIMAUserJoin", der OnMessage Funktion (Listing 6.2).

Listing 6.2: OnMessage Funktion von SIMAServerDlg

```
void CSIMAServerDlg::OnMessage(long lUserId, CSocketMessage& message)
{
        switch (message.GetId())
        {
        case SIMAUserJoin:
                {
                        CString strNickname;
                        message.GetAt(0, strNickname);
                        //Item in Benutzerverwaltung anlegen
                        AddUser(lUserId, strNickname);
                        CString strUserJoin;
                        strUserJoin.Format(IDS_USER_JOIN, strNickname);
                        //User anzeigen
                        AddSystemText(strUserJoin);
                        //Für den User die Liste der Pagernutzer
                        message.SetAt(0, comboListComplete);
                        //zum Senden vorbereiten
                        message.GetAt(0, comboListComplete);
                        //SENDEN der Liste
                        SendMessageToUser(lUserId, message);
                }
                break;
```

Wird eine Textnachricht, also ein zu versendender Pagerruf, vom Server empfangen, so muss der vom Client erhaltene CString ausgewertet und entsprechend verarbeitet werden. Dies geschieht ebenfalls in der Funktion OnMessage, jedoch bei der Case-Abfrage "SIMAText" (Listing 6.3).

Listing 6.3: Fortsetzung OnMessage → Case-Abfrage SIMAText

```
case SIMAText:
        {
                int i = 0;
                int lengthComplete;
                //Komplette, empfangene Nachricht
                CString messageComplete;
                //Die Nachricht höchstpersönlich
                CString messageBody = "";
                //Vorgesehener Empfänger
                CString messageTarget = "";
                //Vom Client empfangene Message
                message.GetAt(0, messageComplete);
                //Länge der Message ermitteln
                lengthComplete = messageComplete.GetLength();

                //|Aus dem String den Empfänger ermitteln
                while(messageComplete[i] != '€')
                {
                        messageTarget += messageComplete[i];
                        i++;
                }
                //Restliche Zeichen sind die Message, welche als "messageBody"-CString
                //abgelegt werden
                while(i < lengthComplete)
                {
                        i++;
                        messageBody += messageComplete[i];
                }
                CString strNickname;
                m_mapIdsNicknames.Lookup(lUserId, strNickname);
                ////////////////////////////////////////////////////
                //manipuliere AMIS
                hdlMainWin = *FindWindow(0, 0);
                if(hdlMainWin)
                {
                        //Pointer auf ein MainWindow
                        pWnd0 = GetWindow(GW_HWNDFIRST);

                        //So lange nächstes MainWindow, bis AMIS gefunden ist
                        while(pWnd0->GetDlgItem(53) == 0)
                              pWnd0 = pWnd0->GetNextWindow(GW_HWNDNEXT);

                        //Ist bereits Child "man. Ruf..." geöffnet?
                        hdlWindow = *FindWindowEx(0, 0, 0, winname);

                        //warten bis dieser Dialog beendet ist
                        while(hdlWindow)
                        {
                                Sleep(100);       //|Wartezeit bis zur nächsten Abfrage
                                hdlWindow = *FindWindowEx(0, 0, 0, winname);
                        }
```

```
		do
		{
			//|Kein Dialog mehr offen? Dann kann für "unsere Nachricht"
			//|ein Fenster geöffnet werden um dann einen
			//|Handle darauf zu erzeugen
			pWnd0->GetDlgItem(53)->
			PostMessage(WM_LBUTTONDOWN , 0, 0);

			pWnd0->GetDlgItem(53)->
			PostMessage(WM_LBUTTONUP , 0, 0);
			Sleep(100);
			hdlDlg = *FindWindowEx(0, 0, 0, winname);
		}while(!hdlDlg);
}

if(hdlDlg)
{
		int targetPointer;
		//Handle auf "manuelle Rufauslösung" anhängen
		m_hdlWindow.Attach(hdlDlg);


		//Pointer auf ComboBox von "manuelle..."
		CComboBox* pCombo = (CComboBox*)
		m_hdlWindow.GetDlgItem(108);

		//String nach Int Wandlung
		targetPointer = atoi(messageTarget);
		pCombo->ShowDropDown(TRUE);
		pCombo->SetCurSel(targetPointer);
		//Empfänger aus der ListBox auswählen
		pCombo->PostMessage(WM_LBUTTONDOWN , 0, 0);
		pCombo->SetCurSel(targetPointer);
		pCombo->PostMessage(WM_LBUTTONUP , 0, 0);
		//Auswahl bestätigen
		pCombo->SendMessage(CBN_SELENDOK, 0, 0);
		//Oberfläche updaten
		m_hdlWindow.UpdateWindow();
		//Pointer auf EditControl (Nachrichtenfenster)
		CEdit* pWnd1 = (CEdit*)m_hdlWindow.GetDlgItem(107);
		//Message einfügen
		pWnd1->ReplaceSel(messageBody, FALSE);
		//Oberfläche updaten
		m_hdlWindow.UpdateWindow();
		Sleep(50);
		//Button Rufen down
		m_hdlWindow.GetDlgItem(105)->
		PostMessage(WM_LBUTTONDOWN , 0, 0);
		//Button Rufen up
		m_hdlWindow.GetDlgItem(105)->
		PostMessage(WM_LBUTTONUP , 0, 0);
		Sleep(50);
		//Button Abbrechen down
		m_hdlWindow.GetDlgItem(109)->
		PostMessage(WM_LBUTTONDOWN , 0, 0);
		//Button Abbrechen up
		m_hdlWindow.GetDlgItem(109)->
		PostMessage(WM_LBUTTONUP , 0, 0);
		//Handle lösen
```

```
                        m_hdlWindow.Detach();

                    }
                    //Im Nachrichtenfenster des Servermoduls Nachrichtenverfasser und
                    //Nachricht anzeigen
                    AddText(strNickname + _T(": ") + messageComplete);
                    message.SetAt(0, strNickname);
                    message.SetAt(1, messageComplete);

                }
                break;
        }
}
```

6.9 Oberfläche und Funktionen des Clientprogramms

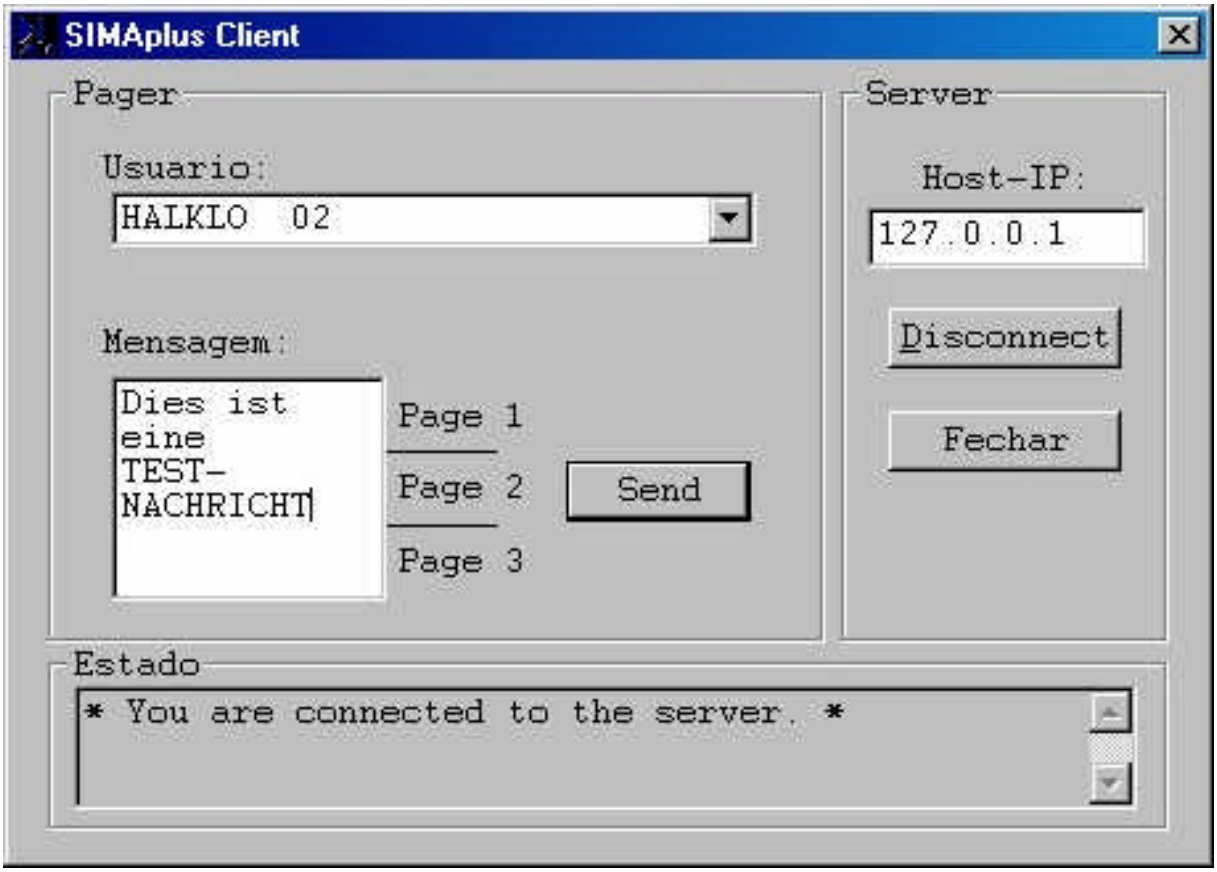

Abbildung 6.13: SIMAplus Client

Während bei SIMAplus Server die Anwendung eine weitere Anwendung manipuliert, so werden in der Client Anwendung lediglich Nachrichten empfangen, ausgewertet und der Nachrichteninhalt entsprechend umgesetzt. Außerdem werden Nachrichten an den Server gesendet. Somit wird in dieser Applikation nur ein Pointer der Klasse CComboBox benötigt, um die Combobox, welche die Pagernutzerliste beinhalten soll, zu initialisieren.
Der CString, welcher nach dem Verbindungsaufbau mit dem Server, empfangen wurde, wird im Client bearbeitet. So wird der String eingelesen und in einer Schleife so lange aufgetrennt, bis alle im String enthalten Pagernutzer als einzelne Strings dazu genutzt werden konnten, die Comboboxliste zu initialisieren.
Der Vorteil, die Pagerliste nicht über eine Datei oder gar statisch im Client einzubinden, sondern diese vom Server zu erhalten, liegt darin, dass diese Liste somit immer aktuell ist. Es muss bei einer installierten SIMAplus Client Anwendung also nicht darauf geachtet werden, dass die Pagernutzerliste die richtigen Pagernutzer enthält. Die Initialisierung der ComboBox Liste wird in der Funktion OnMessage unter der der Case-Abfrage "SIMAUserJoin" durchgeführt. Siehe Listing 6.4.

Listing 6.4: OnMessage Funktion in SIMAClientDlg.cpp

```
void CSIMAClientDlg::OnMessage(CSocketMessage& message)
{
	switch (message.GetId())
	{
	case SIMAUserJoin:
		{
			int lengthComplete;
			CString comboListComplete = "";
			CString comboListPart = "";
			message.GetAt(0, comboListComplete);
			// Länge des Strings, welcher die Liste der Pagernutzer enthält
			lengthComplete = comboListComplete.GetLength();
			if(initComboBox.GetCount() <= 2)
			{	//Abfrage, welche verhindert, dass die Liste der
				//ComboBox mehrere male aufgefüllt wird

				for(int i=0; i<lengthComplete; i++)
				{	//|Solange Ende des String noch nicht erreicht
					if(comboListComplete[i] != '§')
					{	//und aktuelles Zeichen ungleich "§"
						//Pagernutzer-String erzeugen
						comboListPart += comboListComplete[i];
					}
					else
					{	//bei "§", wird String wieder geleert.
						initComboBox.AddString(comboListPart);
						comboListPart = "";
					}
				}
				UpdateWindow();
			}
		}
		break;
```

Soll eine Pagernachricht gesendet werden, so werden mehrere Vorbereitungen zum Versenden der Nachricht getroffen. In einen CString wird der selektierte Pagernutzer aus der Combobox eingetragen. Weiter wird ein Trennzeichen eingefügt zur Kennzeichnung der Empfängerkennung und der Textnachricht an diesen Empfänger ist. Nun wird eine neue Message angelegt, welcher die Nachricht zugewiesen werden kann. Bei der Zuweisung können Empfänger und Textnachricht im CString zusammengefügt werden.
Nachdem die Nachricht an den Server nun fertig initialisiert ist und die notwendigen Informationen, Pagernutzer und Textnachricht enthält, kann die Message über das Netzwerk an den Server gesendet werden. Dieser Ablauf ist in der Funktion OnButtonSend realisiert (Listing 6.5).

Listing 6.5: OnButtonSend Funktion von SIMAClientDlg

```
void CSIMAClientDlg::OnButtonSend()
{
	if (UpdateData(TRUE))
	{
		int ready = initComboBox.GetCount();
		int target = initComboBox.GetCurSel();  //ausgewählter Pagernutzer (=Empfänger)
		CString targetString;
		if(ready < 1)
		{
			CloseConnection();
			AddSystemText("Servidor está ocupado,
					por favor  tente novomente mais tarde.");
		}
		else if(target == CB_ERR)
			AfxMessageBox("Primeiro selecionar um receptor.", MB_OK);
		else
		{
			//Int nach CString Wandlung
			targetString.Format("%d", target);

			//Trennzeichen "€" an String anhängen
			targetString += '€';

			//Neue Message ID erzeugen
			CSocketMessage message(SIMAText);

			//Empfänger + Message in einen String
			message.Add(targetString + m_strSIMAInput);

			//diesen String an Server senden
			SendMessageToServer(message);

			m_buttonSend.EnableWindow(FALSE);

			//Systemnachricht
			AddText(m_strSIMAInput);

			//Pause, um AMIS Abarbeitung zu ermöglichen
			Sleep(1500);

			//String leeren
			m_strSIMAInput.Empty();

			//Oberfläche updaten
			UpdateData(FALSE);

			CloseConnection();
			Sleep(1500);
			DestroyWindow();
		}
	}
}
```

Anhang

A.1 Bedienungsanleitungen für den Editor SIC-AU

A.1.1 Englisch

1. Transfer the symbol tables (*Z0.seq) from the S5 Stations to the PC on which SicAU is installed. Best way is to create a folder, where you will store all the tables (e.g. c:\tmp\s5)
2. Open a symbol table within the program
3. Press the GO-button
4. The program will search for the comments that fit to the new standard
5. When the search is completed, press the OK-button from the status-bar-box
6. Now you will be asked, where to save the new symbol table. For example create a new folder c:\tmp\s5\new and save the file into this folder.
7. After saving, you are able to control and change the new symbol table manually. To give you a better control of the changed comments, you see at the right split screen the original line. In the left split screen, there is the whole new symbol table. You may edit here, if necessary. After editing manual, you have to save the file once again to store the changes.
8. Now, you regenerate the program for the S5 stations with the SIMATIC software, using the new symbol-table.
9. DONE
10. With the Database-Editor you will be able to change the settings for the search of a normalziable comment. You may change the database, but the file must remain in the original path (C://SIC/Database.sdb)
11. If the database-file might be lost, restore copy the file from C://SIC/Default/Database.sdb to C://SIC/Database.sdb

A.1.2 Deutsch

1. Zunächst sind die Symboltabellen der S5 Anlagen auf den PC, auf dem SicAU installiert ist, zu kopieren. Am Besten wird hierfür ein eigener Ordner angelegt (z.B. c:\tmp\s5)
2. Öffnen einer Symboltabelle im Programm.
3. GO-Button betätigen
4. Das Programm sucht nach Kommentaren, welche auf den neuen Standart normiert werden können
5. Wenn die Suche beendet ist, ist das Status Fenster mit dem OK Button zu bestätigen

6. Nun werden Sie gefragt, wo die neue Symboltabelle gespeichert werden soll. Legen Sie beispielsweise einen neuen Unterordner c:\tmp\s5\new an und speichern Sie dort die Datei.
7. Nach dem Speichern können Sie die neue Symboltabelle noch von Hand nachbearbeiten. Um Ihnen eine bessere Kontrolle über die ersetzten Kommentare zu geben, sehen Sie im rechten Splitt Screen die Originalzeile. Im linken Splitt Screen steht die komplette, neue Symboltabelle. Falls notwendig, kann die Datei hier bearbeitet werden. Nach einer manuellen Nachbearbeitung, muss die Datei nochmals gespeichert werden, um die Änderungen zu sichern.
8. Nun kann unter Verwendung der neuen Symboltabelle das Programm für die S5 Anlage neu mit der SIMATIC Software erstellt werden.
9. FERTIG
10. Mit dem Datenbank-Editor ist die Möglichkeit gegeben, die Einstellungen für die Suche nach zu normierenden Kommentaren, zu ändern. Änderungen können gespeichert werden, es ist jedoch darauf zu achten, dass die Datei im original Pfad bleibt (C://SIC/Database.sdb)
11. Falls die Datei Database.sdb verloren gehen sollte, kann diese aus dem Ordner C://SIC/Default wieder ins Verzeichnis C://SIC/ kopiert werden

A.1.3 Portugiesisch

1. Primeiramente copie as tabelas de símbolos da instalação S5 para o Computador, no qual está instalado SicAU. O melhor é abrir um novo arquivo (p.ex. c:\tmp\s5).
2. Abrir uma tabela de símbolos no programa.
3. Clique no ícone GO
4. O programa procura comentários normalizados/que se encaixam no novo padrão
5. Quando a busca tiver terminado, clique no ícone OK na janela de status para confirmar
6. Agora o programa quer saber onde deve ser salva a nova tabela de símbolos. Crie por exemplo um novo índice c:\tmp\s5\new e salve a tabela neste índice.
7. Após salvar você estará apto a controlar e alterar a nova tabela de símbolos manualmente. Para possibilitar um melhor controle de comentários alterados, na tela da direita você pode visualizar a linha original. Na tela da esquerda você visualiza a nova tabela completa. Você pode editar aqui mesmo, se necessário. Após editar manualmente, salve o arquivo mais uma vez e armazene os dados.
8. Em seguida, com a nova tabela de símbolos, o programa para a instalação S5 pode ser re-elaborada com o software SIMATIC.
9. Pronto
10. Com o editor de banco de dados abre-se a possibilidade de alterar a configuração para a busca de comentários normalizáveis. Você pode salvar dados, mas deve-se observar que o arquivo fique no endereço original (C://SIC/Database.sdb)
11. Se o arquivo database.sdb for perdido, este pode ser copiado do índice C://SIC/Default

A.2 SIC-AU

A.2.1 SIC-AU - Dateiüberblick

MainFrm.cpp/MainFrm.h
- Generiert Fenster und handelt die Ansichten (Splittscreen – Datenbank)
- Anzeige des Hilfedialoges

NewTest.cpp/NewTest.h
- Parametrierung der Registry- und Umgebunsvariablen
- Einstellungen für Datei-Öffnen
- Algorithmus zum Finden und Ersetzen der Kommentare der Symboltabelle

NewTestDoc.cpp/NewTestDoc.h
- Einstellung der Datenbankdatei
- Aufräumen der Datenbankdatei
- Datenbanknavigation
- Serialisierung der Datenbankdatei

NewTestView.cpp/NewTestView.h
- Druckerparametrierung für Symboltabelle

Normiert.cpp/Normiert
- Initialisierung der Datenbankvariablen

Splash.cpp/Splash.h
- Begrüßungsbildschirm

StdAfx.cpp/StdAfx.h
- Windows Basisklassen einbinden

TeslaView.cpp/TeslaView.h
- Druckerparametrierung für Datenbank
- Ansichtmanagement der Datenbank

Resource.h
- Makros und Instanzen der Ressourcen

A.2.2 SIC-AU Kommentare und Suchkürzel

Agua de Resfriamento	Âgua Resfr
Agua de Resfriamento RIP **X** – Robo	Flux RIP; Fluxo de Agua, Fluxo Agua ohne Flanco
Akarid Dados com Erro	Akar Err; Err Akar
Akarid Falha Escrita	Akar Escr, Escr Akar
Akarid Falha Leitura	Leit Akar
Akarid Offline com SPS	Comu Akar
Akarid sem Conexao com FIS	Akar Cone; Cone Akar
Akarid Sem Dados	Akar Dado; Dado Akar
Akarid Sem TAG	TAG Sem
Akarid TAG nao Fixado Assoalho	TAG Fix
Akarid Ultimos Carros	Akar Carr; Carr Akar
Bateria CLP	Bater CLP
Bomba Agua sem Fluxo	Agua Estacao
Buffer Limite Inferior	Limi Infer (ohne Cola)
Buffer Limite Superior	Limi Super (ohne Cola)
By-Pass E2 Area Protecao – Laser	E2 Las
By-Pass E2 Area Protecao	E2 (ohne Las/Port/Robo)
By-Pass E2 Area Protecao - Portal **X**	E2 Port
By-Pass E2 Area Protecao - Robo **X**	E2 Robo
By-Pass E7 Sequencia Programa	Mod E7
Chave Geral Grupo Desligado	Chave Geral
Cilindro Basculante Avancar	Cil Ass Ava
Cilindro Basculante Recuar	Cil Ass Rec
Cilindro Bloqueio Avancar	Cil Para Ava
Cilindro Bloqueio Recuar	Cil Para Rec
Cilindro Centralizador Avancar	Cil Cen Ava
Cilindro Centralizador Recuar	Cil Cen Rec
Cilindro Elevatorio Avancar	Cil Ele Ava
Cilindro Elevatorio Recuar	Cil Ele Rec
Cilindro Fixacao Avancar	Cil Fix Ava; Cil Anc Ava; Grampo
Cilindro Fixacao Recuar	Cil Fix Rec; Cil Anc Rec; Grampo
Cilindro Posicionador Avancar	Cil Desl Ava (ohne Ass)
Cilindro Posicionador Recuar	Cil Desl Rec (ohne Ass)
Desligado – Instalacao	Instal Des
Disjuntor Motor - **Equipamento X**	Disj Mot Fres; Disj Mot Jan ; Trans
Disjuntor WinTouch **X**	Disj Win
EMERGENCIA – BTK	Emerg BTK
EMERGENCIA - GERAL MIS	Emerg Ger MIS
EMERGENCIA – GERAL	Emerg Ger (ohne MIS)
Falta Peca **X**	Falt Pro
Freio Falha Unidade - **RB00**	Falh Freio RB
Freio Falha Unidade - **VF00**	Falh Freio GF
Fresagem Capa Eletrodo Falha - Robo **X**	Falh Fres
Fresagem Robo **X** – Tempo Excedido - **OP00**	Fresa Temp Ex (welcher Robbi?????)
Interbus Falha – **OPXX**	Interb Err ohne Perif
Interbus Falha Periferia - **OP00**	Perif Interb
Interbus Master Falha – **OP00 –** Robo **X**	Master nao
Inversor Falha Alimentacao – **DT00**	Falh Aliment (ohne Tens)
Inversor Falha Geral - **DT00**	Falh Invers …….. (ohne freio)
Laser Falha Geral - **OP00**	Lase Falh Ger
Limite Seg. Inf. **Equipamento X**	Lim Inf

Limite Seg. Sup. **Equipamento X**	Lim Sup
Modo Automatico – Laser	Las Mod Aut
Modo Automatico - **PU00**	Mod Aut P (ohne Las & ohne Robo)
Modo Automatico – Robo **X**	Mod Aut Robo
Modo Automatico nao Selecionado – **OP00**	Nao Modo Autom
Modo Manual – Laser	Las Man
Oleo Filtro **Equipamento X**	Oleo Filt (ohne Pre)
Oleo Nivel **Equipamento X**	Oleo Niv (ohne Pre)
Oleo Pre-aviso Nivel Minimo - **OP00**	Oleo Pre Niv
Parada no Agregado **OP00** - Linha Cheia	Parada Lin Ch (ohne Cil)
Parada no Agregado **OP00** - Sem Peca	Parada Sem Pec (ohne Cil)
Perceptron Carroceria Regeitada - **OP00**	Perce Reg
Perceptron Falha – OP00	Perce Falh; Perce Err
Pneumatica Falha 12 Bar - **OP00**	12 Bar
Pneumatica Falha 6 Bar - **OP00**	6 Bar
Pneumatica Falha Pressao - Portal **X**	Falh Press Port
Robo **X** nao Pronto – **OP00**	Nao Pronto
Scanner Laser Defeituoso - **OP00**	Lase Err; Lase Falh
Scanner Laser Sujo – **OP00**	Lase Suj
Seguranca de Area - Cortina **X**	Segura Cort
Seguranca de Area - Porta **X**	Porta de Protecao; Porta de Segura
Seguranca de Area - Scanner **X**	Segura Scan
Skid Falha Sensor Dianteiro - **RB00**	SKID Diant
Skid Falha Sensor Traseiro - **RB00**	SKID Tras
Skid Falha Transferencia	SKID Transf
Skid Falha Velocidade Lenta - **OP00**	SKID Exced; SKID Tempo
Skid Fora Posicao - **HT00**	SKID Posic
Skid Posicao Invertida - **RB00**	SKID RB Inv
Skid Posicao Invertida - **VF00**	SKID VF Inv
Sobretemperatura - **Equipamento X**	Sobret (ohne Pai)
Sobretemperatura Painel **XXXX** – **PU**	Sobret Pai
Solda Ponto Erro - Robo **X** - Bosch **X**	Err Sold B
Solda Ponto Erro - Robo **X** - Fronius **X**	Err Sold If not B
Solda Sem Corrente - Robo **X** - Bosch **X**	Sem Corrent
Solda Sem Pino - Robo **X** - Tucker **X**	Tuck Pino; Tuck Sem Sold
Solda Trocar Capa - Robo **X**	Ponto Sold
Start Ligado - Instalacao **OP00**	Esta Lig
Tempo Ciclo Excedido - **OP00**	Tempo Op Exce
Tempo Excedido - **Equipamento X**	Tempo El Ex; Tempo Tr Ex
Tensao Falha 220 VAC	Tensao 220 (mit E?)
Tensao Falha 24VDC	Falha 24
Tensao Falha 380 VAC	Tensao 380 (mit E ?)
Tensao Falha SPUE	Tensao SPUE
Tomadas sem 220VAC	Falh Tomad
Transportador Correia Frouxa	Corr Trans Fro
Transportador Correia Quebrada	Corr Trans Que
Transportador Fora Posicao	Fora Posi

A.3 SIMAplus

A.3.1 SIMAplus Server Dateiüberblick

SIMAServer.cpp/ SIMAServer.h
: Generiert Fenster
Anzeige des Hilfedialoges

SIMAServerDlg.cpp/ SIMAServerDlg.h
: Diese Datei beinhaltet die Funktionalität des Server Moduls der SIMAplus Client-Server Lösung. Hier werden die Elemente der grafischen Oberfläche mit Funktionen belegt. Es wird ein Handle auf das "AMIS" Fenster und ein Handle auf die "manuelle Rufauslösung" generiert. Die Liste der Pagernutzer von AMIS wird ausgelesen und bei einem Connect eines Client, an diesen versendet. Client Nachrichten werden empfangen, ausgewertet und an die "manuelle Rufauslösung" von AMIS übergeben.

SocketDataBuffer.cpp/SocketDataBuffer.h
: Speichert jegliche C++ Dateiformate in einem Array, welches dann versendet werden kann.

SocketDefines.cpp/ SocketDefines.h
: Auflistung der verschiedenen Meldungen, die bei der Socketverbindung auftreten können.

SocketMessage.cpp/SocketMessage.h
: Hier wird das Objekt generiert, welches an den Server gesendet wird. Das darin verwendete Array ist dynamisch angelegt und vergrößert sich automatisch, wenn es nötig sein sollte.

SocketMessageData.cpp/SocketMessageData.h
: Enthält Konstruktoren für alle möglichen C++ Datentypen, welche als Nachricht verwendet werden könnten.

SocketServer.cpp/SocketServer.h
: Implementiert die Server-Seite der Client-Server Architektur.

SocketServerSocket.cpp/SocketServerSocket.h
: Implementiert die MFC-Klasse CSocket und ermöglicht den Zugriff auf Memberfunktionen dieser MFC-Klasse.

SocketUser.cpp/SocketUser.h
: Stellt der Threadverwaltung die ID und den Socket eines jeden verbundenen Clients zur Verfügung.

SocketUserManagement.cpp/SocketUserManagement.h
: Realisiert die Threadverwaltung der Serverseite. Clients werden hier angelegt bzw. gelöscht und Socketnachrichten werden hier empfangen und versendet.

A.3.2 SIMAplus Client Dateiüberblick

SIMAClient.cpp/SIMAClient.h
: Generiert Fenster und handelt die Ansichten (Splittscreen – Datenbank) Anzeige des Hilfedialoges

SIMAClientDlg.cpp/SIMAClientDlg.h
: Diese Datei beinhaltet die Funktionalität des Client Moduls der SIMAplus Client-Server Lösung. Hier werden die Elemente der grafischen Oberfläche mit Funktionen belegt, es werden Nachrichtenstrings zum Versenden aufbereitet und empfangene Strings in verwertbare Informationen gewandelt.

OleaccProxy.cpp/OleaccProxy.h
: Wird benötigt, um Programm auf Win9x verwenden zu können.

SocketClient.cpp/SocketClient.h
: Implementiert die Client-Seite der Client-Server Architektur.

SocketClientSocket.cpp/SocketClientSocket.h
: Implementiert die MFC-Klasse CSocket und ermöglicht den Zugriff auf Memberfunktionen dieser MFC-Klasse.

SocketDataBuffer.cpp/SocketDataBuffer.h
: Speichert jegliche C++ Dateiformate in einem Array, welches dann versendet werden kann.

SocketDefines.cpp/SocketDefines.h
: Auflistung der verschiedenen Meldungen, die bei der Socketverbindung auftreten können.

SocketMessage.cpp/SocketMessage.h
: Hier wird das Objekt generiert, welches an den Server gesendet wird. Das darin verwendete Array ist dynamisch angelegt und vergrößert sich automatisch, wenn es nötig sein sollte.

SocketMessageData.cpp/SocketMessageData.h
: Enthält Konstruktoren für alle möglichen C++ Datentypen, welche als Nachricht verwendet werden könnten.

B. Quellen:

http://www.codeproject.com

http://www.codeguru.com

http://www.engiby.ch/espa444/espapri.htm

http://www.smlan-berlin.de/html/seminare/mfc-grund.html

http://www.angelcode.com/articles/window2/window2.asp

http://www.fawcette.com/vsm/2002_11/magazine/columns/qa/

http://www.cs.odu.edu/~swaney/CS595/Project/CS595_Main.html

http://www.codeproject.com/internet/NDK/NDK_src.zip

http://www.thescarms.com/vbasic/PassString.asp

http://msdn.microsoft.com/library/default.asp

http://archive.devx.com

- Behme, Wolfgang: ZP-Stichwort: Entscheidungsunterstützungssysteme. In: Zeitschrift für Planung 2/1992, S. 179 - 184.
- Bullinger, Hans-Jörg/Fähnrich, Klaus-Peter: Informationsarchtekturen im Unternehmen als strategische Herausforderung für das Management. In: Office Management 11/1992, S. 62 - 64.
- Herget, Josef: Das betriebliche Informationsmanagement vor einer Neuorientierung - Perspektiven und Konsequenzen. In: NfD 46 (1995), S. 25 - 32.
- Kaske, Silvia: MIS, EIS, CIS - Babylon ist mitten unter uns. In: Office Management 11/1992, S. 46 - 48.
- Schmelz, Jürgen: BusinessObjects: Schaufenster zum Data Warehouse. In: IT Management 03/04 1995, S. 72 - 74.

genutztes Tool:

Winspector Spy
http://www.gipsysoft.com/articles/winspector

C. Danksagung

Ich möchte diese Diplomarbeit meinen Eltern Brigitte und Edgar Stumpp widmen, die, davon abgesehen, dass sie mir das Studium ermöglichten, auch immer großes Interesse für meine Arbeit zeigten, und mich soweit wie möglich unterstützten.

Danken möchte ich meinem Bruder Patrick Stumpp, welcher mich durch seine Auslandstätigkeiten dazu inspirierte, meine Diplomarbeit im Ausland zu durchzuführen.

Ebenfalls gilt mein Dank Dipl. Kfm. Thomas Schmall, Plant Mangager bei VW/Audi do Brasil, welchem ich meine Diplomandenstelle zu verdanken hatte.

Des Weiteren möchte ich ganz besonders Dr. Wolfgang Arndt von der Fachhochschule Konstanz und Dipl. Ing. Heiko Borho des VW/Audi Werkes in Curitiba für die Zusammenarbeit in jeder Phase meiner Diplomarbeit danken.

Schließlich gilt mein Dank noch meinem Mitbewohner und Mitstudenten Sergio Manuel Lampreia Mestre und meinem Arbeitskollegen Egon Hübner und allen Freunden und Bekannten für ihre moralische Unterstützung.