Inhaltverzeichnis

1 Einleitung

Hohe Kundenanforderungen, die Verkürzung von Produktlebenszyklen stellen Unternehmen seit Beginn des neuen Jahrtausends immer wieder vor großen Herausforderungen. Damit ein Unternehmen in der heutigen Zeit international konkurrenzfähig sein kann, muss es sich laufend der Marktstruktur anpassen. Bei zunehmendem Kosten- und Wettbewerbsdruck ist es notwendig Qualitätsverbesserungen, eine höhere Leistungsfähigkeit des jeweiligen Systems sowie die Senkung von Betriebskosten zu erzielen. Oft sind diese Ziele voneinander abhängig und tragen damit zu Zielkonflikten bei, die durch entsprechende Maßnahmen beseitigt werden müssen.

Automatisierungstechnik ist der innovative Motor der industriellen Produktion. Roboter steuern Prozessleitsysteme und überwachen Produktionsanlagen in nahezu allen Bereichen der Industrie. Auch die elektrische Energieerzeugung, die Steuerung von Verkehrsleitsystemen sowie die dezentrale Lagerverwaltung und die Logistik wären in der heutigen Zeit ohne den Einsatz von Steuerungs- und Regelungstechnik undenkbar. Durch den Übergang zu der Automatisierungstechnik werden Funktionen, die bisher durch Mechanik oder Elektromechanik realisiert wurden, zunehmend durch mechatronische Softwaresysteme ersetzt. Dadurch werden die zu steuernden Anlagen immer größer und komplexer.

Die Überprüfung der korrekten Funktionalität einer Anlage, die Fehlersuche und das frühzeitige Erkennen von drohenden Komponentenausfällen sind von entscheidender Bedeutung für das wirtschaftliche Betreiben komplexer Automatisierungsprozesse. Eine große Aufgabe der Industrie ist es, im Falle eines Fehlers den Schaden für die Menschen sowie für die Firma möglichst klein zu halten. Tritt ein Prozesszustand auf, der Probleme verursachen könnte und ein Eingreifen des Bedieners erfordert, so wird in gut programmierten Anlagen sofort eine Alarmmeldung optisch und/oder akustisch ausgegeben.
Die Rückverfolgung von Störauslösern und die genaue Ermittlung der Fehlerquellen erfordert Angaben über einen längeren Zeitraum. Dies ist die Motivation für eine komplette Aufzeichnung aller Betriebsmeldungen und Alarmereignisse.

2 Background

2.1 Beschreibung der Firma

Die GEA Tuchenhagen Dairy Systems GmbH ist ein international tätiges Unternehmen für technisch und wirtschaftlich optimierte Prozesstechnologie zur Verarbeitung von Milch, Nahrungsmitteln und Saft. Die Gesellschaft hat circa 300 Mitarbeiter. Sie entstand aus einem Zusammenschluss folgender Gesellschaften:

- Tuchenhagen, Bereich Nahrungsmittel und Milch (gegründet 1931, Erfinder des Doppelsitzventils, Mitglied der GEA Gruppe seit 1995, Sitz Büchen, Vertretungen weltweit)
- GEA Ahlborn (gegründet 1856, Pionier in der Milchindustrie und Erfinder des Plattenwärmetauschers, Mitglied der GEA Gruppe seit 1979, Sitz Sarstedt)
- GEA Finnah (gegründet 1975, führend in der aseptischen Prozesstechnologie, Röhrenwärmetauschern und UHT-Anlagen, Mitglied der GEA Gruppe seit 1989, Sitz Ahaus)

Inzwischen wurden diese drei erfolgreichen Gesellschaften unter dem Namen Tuchenhagen Dairy Systems zusammengeführt und sind heute wie folgt organisiert:

Sarstedt
Milch
Fruchtsaft
Montage

Büchen
Nahrungsmittel
Export
Automation

Ahaus
Aseptik

Abbildung 2.1 Tuchenhagen Dairy Systems GmbH

Diese Informationen stammen aus der Internetwebseite der Firma („www.gea-tds.de“).

2.2 Aufgabenstellung

Ziel dieser Arbeit ist die Entwicklung einer Anwendung in Microsoft Visual Basic .NET zur kompletten Aufzeichnung der Alarme und Ereignisse aus den WinCC- und Intouch-Applikationen mit Statistikfunktionen zur Ermittlung von Hitlisten (Häufigkeit der Meldungen).

Für die Software gibt es folgende Anforderungen:

- Die Oberfläche und die Handhabung der Software sollten praktisch sein. Ziel soll sein, dass die Software selbsterklärend und einfach zu bedienen ist.
- Die Konfiguration der Software soll nur mit den entsprechenden Benutzerrechten editiert werden können.
- Die Alarme und Ereignisse sollen aus der Alarmdatenbank von WinCC bzw. Intouch ausgelesen und in eine einheitliche Datenstruktur gebracht werden.
- Die Aktualisierung der Meldungen soll sowohl manuell als auch zeitgesteuert erfolgen können.
- Aufgetretene Fehler sollen dem Bediener zur Anzeige gebracht werden. Hierfür soll ein Reset- oder Quittierknopf geben, um die Fehler aufzuheben.
- Die Alarmmeldungen und Ereignisse sollen in einem vom Benutzer ausgewählten Zeitabschnitt tabellarisch angezeigt werden
- Um eine detaillierte und aussagekräftige Auswertung der Meldungen zu ermöglichen, soll das Programm über konfigurierbaren Filterfunktionen verfügen.
- Das Programm soll außerdem über zwei Ansichten verfügen:
  - Chronologische Ansicht: die Alarme und Ereignismeldungen werden nach dem Zeitpunkt des Auftretens sortiert.
  - Statistik Ansicht: die Alarme und Ereignismeldungen werden nach der Häufigkeit ihres Auftretens sortiert.
- Die Meldungen von unterschiedlichen Filtern sollen unterschiedlich eingefärbt werden. Die Filterfarben sollen wiederrum einstellbar sein.

3 Grundlagen

Dieses Kapitel dient zur Darstellung und Erklärung theoretischer Grundlagen, die für die Durchführung dieser Arbeit erarbeitet werden mussten und von grundlegender Bedeutung sind. Dazu gehört ein Hinblick auf die Meldeprojektierung bzw. –archivierung in WinCC und in Intouch. Alle Informationen stammen entweder aus der WinCC- oder aus der Intouch-Dokumentation. Die verwendeten Dateien sind auf der beigefügten CD-ROM zu finden.

3.1 WinCC Meldesystem

Das WinCC Meldesystem besteht aus Konfiguration- und Runtime-Komponenten:

- Die Konfiguration-Komponente des Meldesystems ist das AlarmLogging. Hier wird konfiguriert, wann welche Meldungen mit welchen Inhalten erscheinen. Eine Meldung kann zum Beispiel ausgelöst werden, wenn ein bestimmtes Bit in einem Automatisierungssystem gesetzt wird, oder wenn ein Prozesswert einen festgelegten Grenzwert überschreitet. Das folgende Bild zeigt den Aufbau des AlarmLogging.

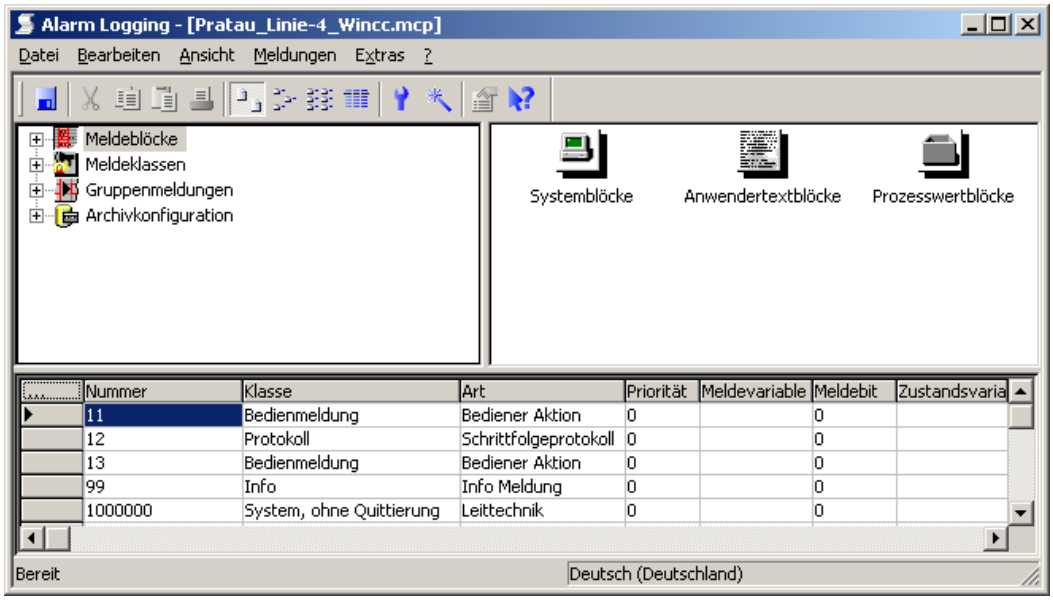

Abbildung 3.1 Editor des WinCC AlarmLoggings

- Die Runtime-Komponente des Meldesystems ist die AlarmLogging Runtime. Sie ist dafür zuständig, Prozessmeldungen und lokale Ereignisse in die Archivdatenbank zu speichern, die definierten Überwachungen durchzuführen und die Meldungs-quittierungen zu verwalten.

3.1.1 Meldeprojektierung

WinCC erfasst Prozessmeldungen und lokale Ereignisse und trägt sie in chronologischer Reihenfolge in die Archivdatenbank ein. Die Meldestruktur ist frei definierbar und kann dadurch auf die speziellen Erfordernisse der Anlage zugeschnitten werden. Im Folgenden werden die Typen sowie die Grundzustände von Meldungen erläutert. Außerdem erfolgt ein Hinblick auf die Mechanismen einer Meldeprojektierung unter WinCC.

Typen von Meldungen

Es wird in WinCC zwischen folgende Meldungstypen unterschieden:

- Prozessmeldungen melden Ereignisse des automatisierten Prozesses.
- Leittechnikmeldungen sind Fehlermeldungen, die von SIMATIC PCS7-Komponenten verursacht oder erkannt werden.
- Bedienmeldungen werden bei einer Bedienung einer Prozessgröße ausgelöst.
- Systemmeldungen werden vom WinCC generiert.

Grundzustände einer Meldung

In WinCC werden drei Grundzustände einer Meldung unterschieden:

- Eine Meldung ist "gekommen", solange das auslösende Ereignis noch vorliegt, die Ursache für die Meldung also noch besteht.
- Eine Meldung ist "gegangen", sobald die Ursache nicht mehr besteht.
- Eine Meldung ist "quittiert", sobald der Bediener sie quittiert hat. (Eine Meldung kann auch so projektiert sein, dass der Bediener sie nicht quittieren muss)

Mechanismen einer Meldungsprojektierung

Bei der Projektierung einer Meldung unterstützt das WinCC-Meldesystem folgende Mechanismen:

- Beim Bitmeldeverfahren löst ein Ereignis eine projektierte Einzelmeldung aus, wenn sich in der zugeordneten Prozessvariablen (Meldevariable) ein Meldebit ändert, wobei der Zeitstempel der Meldung vom Meldesystem automatisch generiert wird.
- Beim zeitfolgerichtigen Melden versendet die Steuerung beim Eintreten eines Ereignisses ein Telegramm mit Meldedaten und Zeitstempel.

Der erste Punkt soll hier etwas genauer erklärt werden, während der zweite für diese Bachelorarbeit unwichtig ist und deswegen nicht weiter erläutert werden wird.

Um das Meldesystem im Bitmeldeverfahren zu projektieren, muss wie folgt vorgegangen werden:

1- Eine Meldung aus fertigen Informationsbausteinen (Meldeblöcken) frei definieren. Dazu sind folgende Meldeblöcke zu unterscheiden:

- Systemblöcke

  Sie beinhalten Systemdaten, die vom Alarm Logging vergeben werden. Dazu gehören zum Beispiel Datum, Uhrzeit, Zustand usw.

Abbildung 3.2 WinCC-Editor zum Bearbeiten der Systemblöcke

- Prozesswertblöcke

  Sie beinhalten Werte, die aus dem Prozess geliefert werden, z.B. aktuelle Füllstände oder Temperaturen.

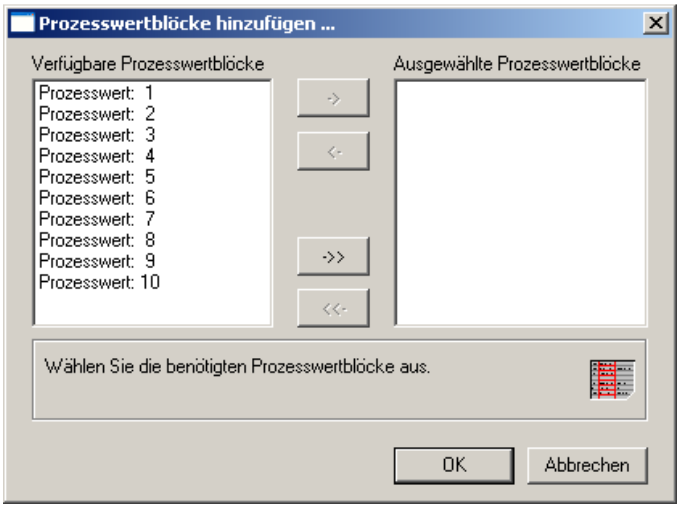

Abbildung 3.3 WinCC-Editor zum Bearbeiten der Prozesswertblöcke

- Anwendertextblöcke

  Sie beinhalten Texte, die zur allgemeinen Information und Verständlichkeit beitragen, z.B. Erläuterungen zu Meldungen, Störort, Meldeursache usw.

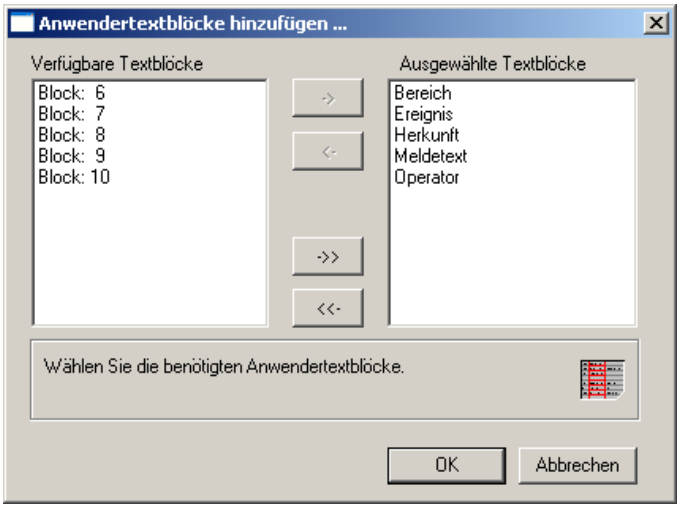

Abbildung 3.4 WinCC-Editor zum Anwendertextblöcke

2- Die Eigenschaften gleichartiger Meldungen in einer frei definierbaren Meldeklasse festlegen.

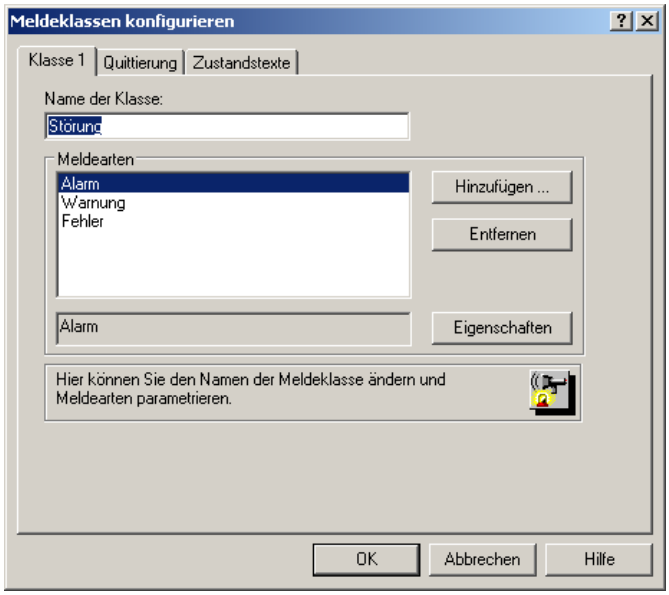

Abbildung 3.5 WinCC-Editor zur Konfiguration der Meldeklassen

3- Meldeklassen in Meldearten unterteilen, z.B. „Alarm“, „Warnung“. Jede Meldeart erhält ein eigenes Farbschema bestehend aus Schrift- und Hintergrundfarbe für die Meldungszustände „gekommen“, „gegangen“ und „quittiert“.

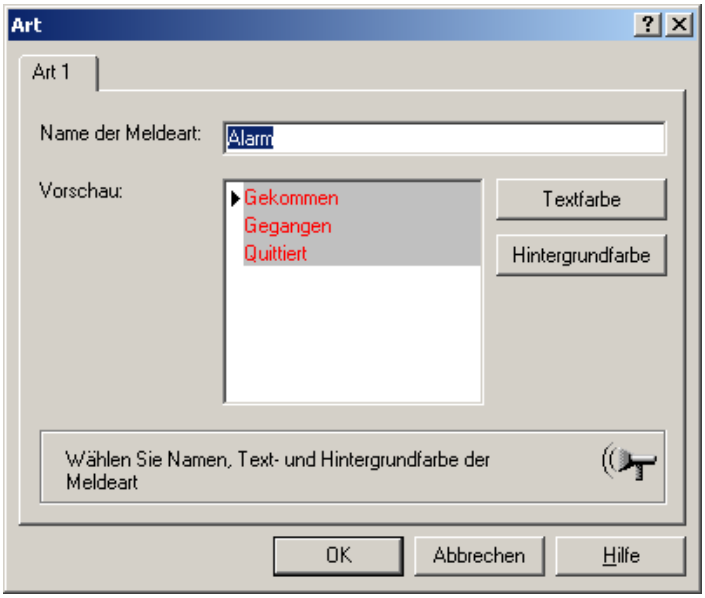

Abbildung 3.6 WinCC-Editor zur Konfiguration der Meldearten

Hinweis:

- Während die Systemblöcke fest an die Meldungen gebunden sind, können Prozesswertblöcke und Anwendertextblöcke vom Benutzer selbst projektiert werden.
- Die WinCC-Meldeblöcke, Meldeklassen und die Meldearten werden nicht durch ihren Namen, sondern durch ihre eindeutige Nummer identifiziert.

Leistungsdaten

Bei der Projektierung sind folgende Leistungsdaten aus der WinCC-Hilfe zu beachten.

	Maximal
Prozessvariablen pro Meldezeile	10
Anwendertextblöcke pro Meldezeile	10
Meldeklassen	10
Meldearten	10
Meldeprioritäten	17 (0…16)

Tabelle 1 Leistungsdaten bei der Meldeprojektierung unter WinCC

3.1.2 Meldearchivierung

Mit dem Archivmanagement in WinCC besteht die Möglichkeit, Prozesswerte und Meldungen für die gezielte Dokumentation von Betriebs- und Störzustände zu archivieren. Zur Archivierung wird der SQL-Server eingesetzt.
Das WinCC Meldearchiv besteht aus mehreren Einzelsegmenten. Man kann für das Archiv folgendes einstellen:

- den Zeitraum sowie die maximale Größe über alle Segmente. Mit diesen Angaben wird die Größe der Archivdatenbank festgelegt. Wenn eines dieser Kriterien überschritten wird, wird ein neues Segment begonnen und das älteste gelöscht.
- den Zeitraum, den ein Einzelsegment umfasst, sowie dessen maximale Größe. Wenn eines dieser Kriterien überschritten wird, wird ein neues Segment begonnen.
- Den Zeitpunkt des ersten Segmentwechsels.

Die folgende Abbildung zeigt das Konfigurationsfenster für das Meldearchiv.

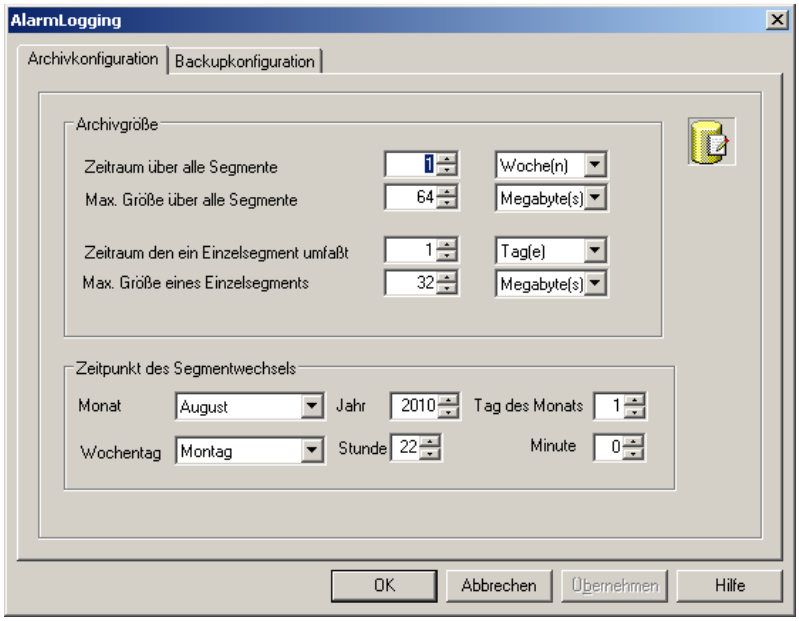

Abbildung 3.7 Konfigurationsfenster des WinCC-Meldearchives

Hinweis:

Ein neues Segment wird auch dann angelegt, wenn Meldedaten online projektiert werden.

In den Meldearchiven werden alle zu einer Meldung gehörigen Daten inklusive der Konfigurationsdaten gespeichert. So können aus den Archiven alle Eigenschaften einer Meldung abgelesen werden, z.B. Meldeklasse, Meldeart, Zeitstempel und Meldetext. In der folgenden Tabelle sind alle Felder eines Meldearchivdatensatzes aufgelistet.

Feldname	Typ	Bedeutung
MsgNr	Integer	Meldenummer
State	Small Integer	Status der Meldung
DateTime	DateTime	Zeitstempel der Meldung (Datum und Uhrzeit ohne Millisekunden)
Ms	Small Integer	Zeitstempel der Meldung (Millisekunden)
Instance	VarChar(255)	Instanzname der Meldung
Fleags1	Integer	(Nur interne Verwendung)
PValueUsed	Integer	Verwendete Prozesswerte
PValue1 bis PValue10	Real	Numerischer Prozesswert 1 bis 10
PText1 bis PText10	VarChar(255)	Prozesswerttext 1 bis 10
Computername	VarChar(255)	Rechnername
Application	VarChar(255)	Applikationsname
Comment	VarChar(255)	Kommentar
Username	VarChar(255)	Benutzername
Counter	Integer	Fortlaufender Meldezähler
TimeDiff	Integer	Zeitdifferenz zum Zustand „Gekommen“
Classname	VarChar(255)	Name der Meldeklasse
Typename	VarChar(255)	Name der Meldeart
Class	Small Integer	ID der Meldeklasse
Type	Small Integer	ID der Meldeart
Text1 bis Text10	VarChar(255)	Meldetext 1 bis 10
AG_NR	Small Integer	Nummer des AG
CPU_NR	Small Integer	Nummer der CPU
CrComeFore	Integer	Vordergrundfarbe für Status „gekommen“
CrComeBack	Integer	Hintergrundfarbe für Status „gekommen“
CrGoFore	Integer	Vordergrundfarbe für Status „gegangen“
CrGoBack	Integer	Hintergrundfarbe für Status „gegangen“
CrAckFore	Integer	Vordergrundfarbe für Status „quittiert“
CrAckBack	Integer	Hintergrundfarbe für Status „quittiert“
LocaleID	Integer	Ort des Alarms
Priority	Integer	Priorität
AP_type	Integer	Loop in Alarm
AP_name	VarChar(255)	Loop in Alarm Funktionsname
AP_PAR	VarChar(255)	Loop in Alarm Bild
InfoText	VarChar(255)	Infotext
TxtCame	VarChar(255)	Text gekommen
TxtWent	VarChar(255)	Text gegangen
TxtCameNWent	VarChar(255)	Text gekommen und gegangen
TxtAck	VarChar(255)	Text quittiert
AlarmTag	Integer	Meldevariable
AckType	Small Integer	Quittiertyp
Params	Integer	Parameter
Servername	VarChar(255)	Servername

Tabelle 2 Felder eines WinCC-Meldearchivdatensatzes

Hinweis:

Der Uhrzeitstempel der archivierten Meldungen ist immer im Standard-Format UTC (Coordinated Universal Time). Daher muss bei Angabe der Start- und Endzeitpunkte in absoluter Form die regionale Zeitzone und ggf. Sommer-/Winterzeit berücksichtigt werden. Ebenso die Archivsegmente des AlarmLoggings verwenden im Dateinamen die UTC-Zeit.

Da Meldungen sprachabhängig projektiert werden, existiert in den Archiven für jede projektierte Sprache eine Tabelle mit den Konfigurationsdaten. Informationen über die möglichen Abfrage-Sprachen bzw. den entsprechenden „ViewName“ sind im SQL-Server in den verbundenen Meldearchiven unter „Views“ zu finden. Darin werden alle Sprachen mit ihren Kennungen z.B. „ALGVIEWDEU“ angezeigt.

3.2 Intouch Alarmsystem

Das Alarmsystem von Intouch besteht aus folgenden Komponenten:

- Alarm-Manager, der die aktiven sowie archivierten Alarme und Ereignisse verwaltet.
- Alarm DB Logger, ein Dienstprogramm, das Archivalarme und Ereignisse in die SQL Server-Alarmdatenbank schreibt.
- Alarmdrucker, der zum Ausdrucken von Archivalarmen dient.
- ActiveX-Steuerelementen, mit denen zur Laufzeit Alarme und Ereignisse aus dem internen Alarmspeicher oder der Alarmdatenbank angezeigt werden können.

Die folgende Abbildung zeigt das System im Überblick.

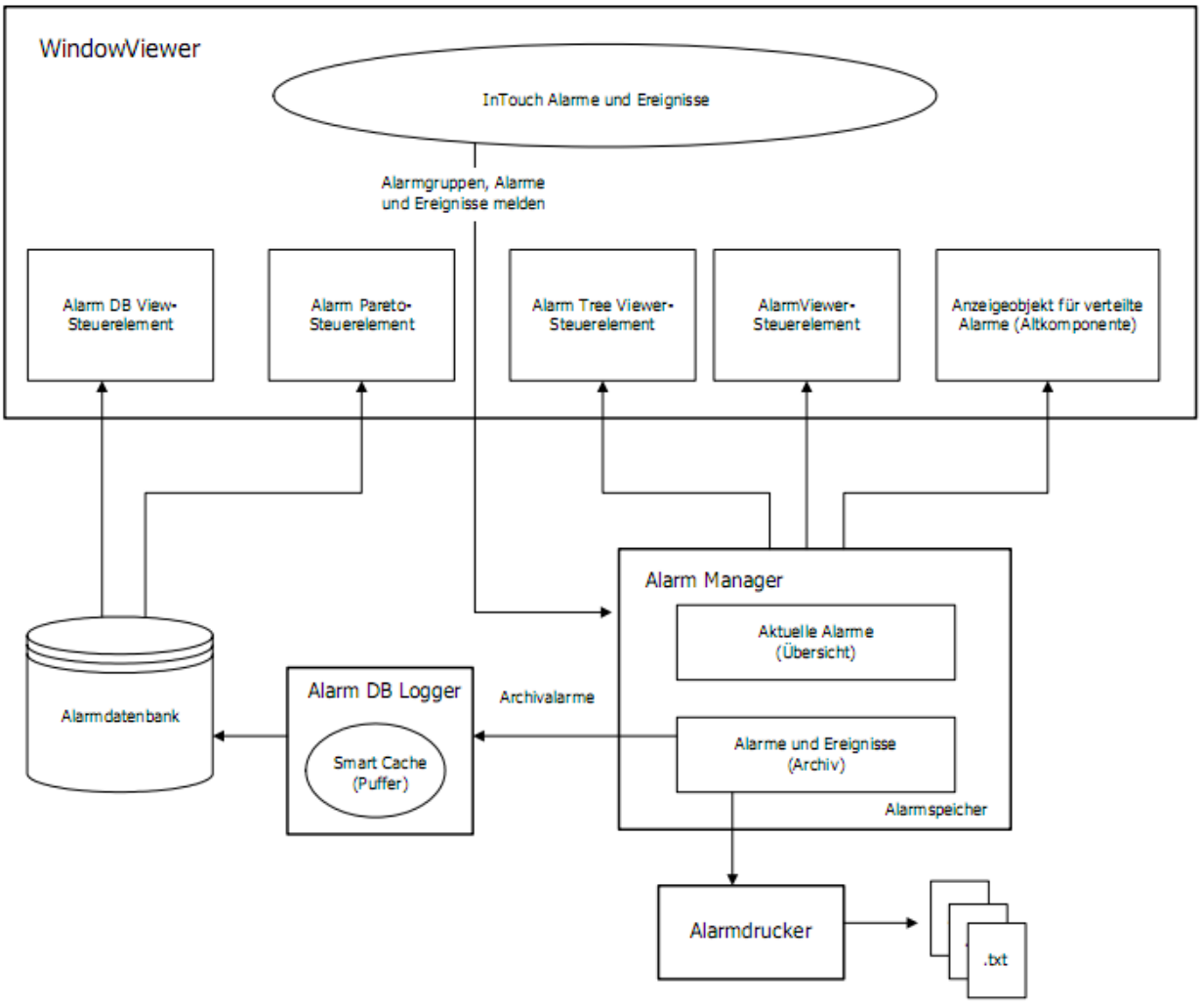

Abbildung 3.8 Das verteilte Alarmsystem von Intouch

3.2.1 Alarmprojektierung

Um ein Alarm für eine Variable zu konfigurieren, muss in der Variablendefinition den gewünschten Alarmtyp aktiviert und den dazugehörigen Grenzwert angegeben werden. Der Alarm wird ausgelöst, sobald dieser Grenzwert über- bzw. unterschritten wird. Hierzu unterscheidet man zwischen folgenden Alarmtypen:

- Binär Alarm gehört zu einer binären Variablen. Hier kann festgelegt werden, bei welchem der beiden möglichen Variablenwerte („0“ oder „1“) der Alarm ausgelöst werden soll.

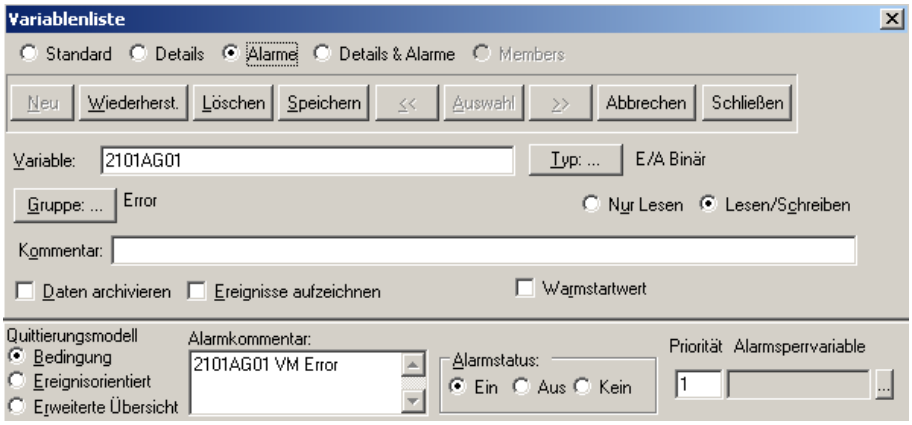

Abbildung 3.9 Konfiguration von binären Alarmen in Intouch

- Wertalarm ist mit einer Integer- oder Real-Variablen verknüpft. Dabei werden verschiedene Grenzwerte (Low oder High) oder „extreme“ Grenzwerte (LoLo oder HiHi) konfiguriert, bei deren Über- bzw. Unterschreiten jeweils ein Alarm ausgelöst wird.

Abbildung 3.10 Konfiguration von Wertalarmen in Intouch

- Abweichungsalarm ist mit einer Integer- oder Real-Variablen verknüpft. Der aktuelle Wert wird dabei zunächst mit einem Zielwert verglichen. Die Differenz der beiden Werte wird dann mit einem oder mehreren Grenzwerten verglichen, die als Prozentsatz des Wertbereichs der Variablen angegeben sind.

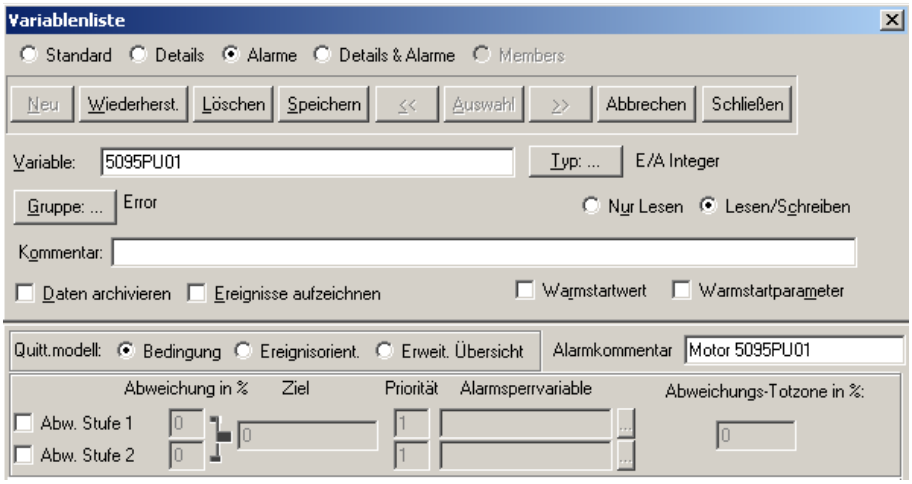

Abbildung 3.11 Konfiguration von Abweichungsalarmen in Intouch

- Änderungsalarm wird ausgelöst, wenn sich der Wert einer Variablen in einem bestimmten Zeitraum um mehr als einem bestimmten Betrag ändert. Bei jeder Wertänderung wird die Variable auf ihre Änderungsrate überprüft.

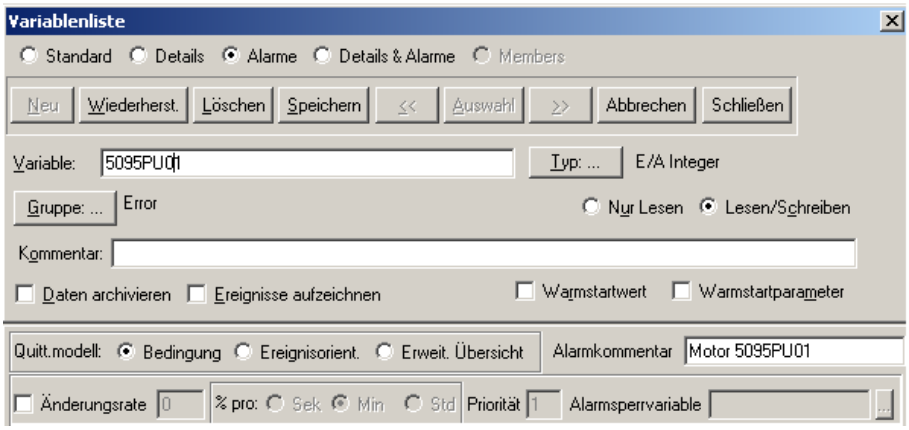

Abbildung 3.12 Konfiguration von Änderungsalarmen in Intouch

Hinweis

Ein wichtiges Mittel zur Strukturierung von Alarmmeldungen ist die Zuordnung zu Alarmgruppen und die Einstufung in Prioritäten. Alle Variablen gehören zur Systeminternen Alarmgruppe $System, die in Unter-Alarmgruppen gegliedert werden kann.

3.2.2 Alarmarchivierung

Anders als bei WinCC werden Alarme und Ereignisse direkt aus der Alarmconsumer-Schnittstelle aus den jeweiligen Intouch Applikation in die Datenbank geschrieben. Dazu dient das Dienstprogramm „Alarm DB Logger“.

Bevor die Alarmdatensätze in die Datenbank geschrieben werden können, muss zuerst die Aufzeichnung konfiguriert werden.

- Datenbankverbindung konfigurieren

  Auf der ersten Seite des Alarm DB Logger Manager muss die Verbindung zur Alarmdatenbank konfiguriert werden. Hier kann ausschließlich die SQL Server-Authentifizierung verwendet werden. Diese muss daher im SQL Server auf den gemischten Modus eingestellt sein.

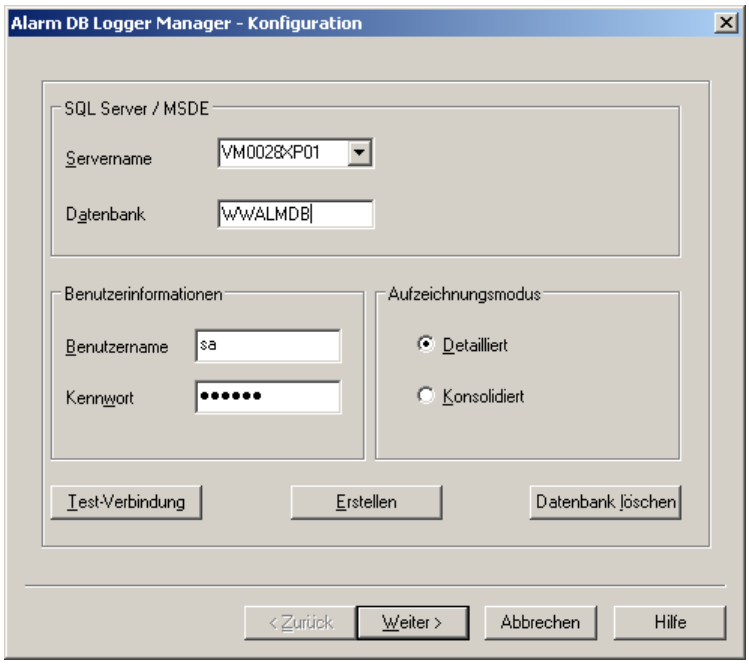

Abbildung 3.13 Konfiguration der Datenbankverbindung in Intouch

- Abfrageauswahl konfigurieren

  Auf der zweiten Seite des Alarm DB Logger Manager wird mit einer Alarmabfrage festgelegt, welche Alarme aufgezeichnet werden sollen. Außerdem wird ein Prioritätsbereich für die Abfrage festgelegt.

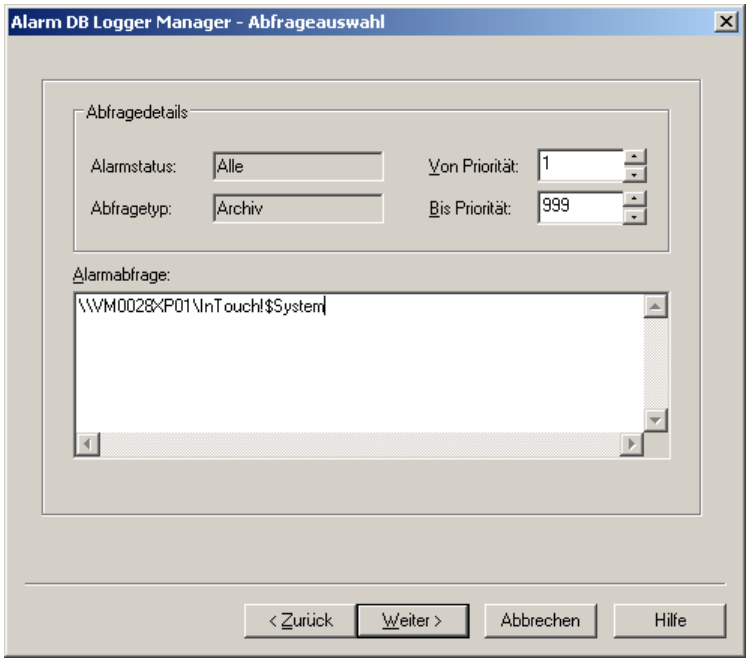

Abbildung 3.14 Konfiguration der Abfrageauswahl in Intouch

- Auf der dritten Seite des werden zusätzliche Einstellungen für die Aufzeichnung konfiguriert. Hier wird festgelegt, ob Ereignisse ebenfalls in der Alarmdatenbank aufgezeichnet werden sollen und ob der Logger als Windows-Dienst oder als normale Anwendung ausgeführt werden soll. Außerdem kann hier die Alarmaufzeichnungsrate konfiguriert werden.

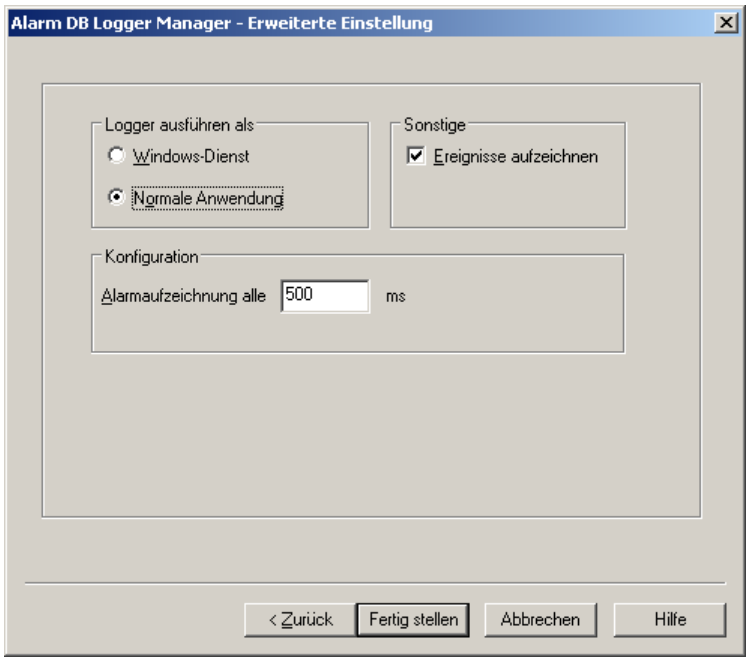

Abbildung 3.15 Konfiguration der Alarmaufzeichnungsrate in Intouch

Die Alarme werden mittels einer Abfrage von Intouch-Alarmquellen abgerufen und dann in einem Zwischenspeicher (dem "Smart Cache") abgelegt. Der Inhalt dieses Zwischenspeichers wird dann in regelmäßigen Abständen in die Datenbank geschrieben.

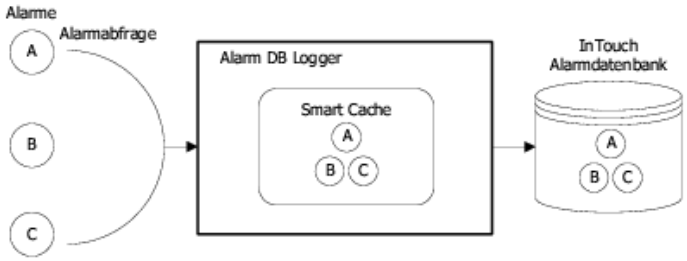

Abbildung 3.16 Alarmaufzeichnung mittels dem Alarm DB Logger in Intouch

Während der Alarmaufzeichnung wird im Alarm DB Logger Manager angezeigt, wie weit der Smart Cache mit Alarmdatensätzen gefüllt ist. Der Cache füllt sich in der Regel dann, wenn die Verbindung zum SQL Server unterbrochen ist oder wenn die Alarme schneller auftreten, als sie in die Datenbank geschrieben werden können.

In der folgenden Tabelle sind alle Felder eines Alarmdatensatzes aufgeführt.

Feldname	Typ	Bedeutung
EventStamp	DateTime	Datum und Uhrzeit des Alarmereignisses (Lokale Zeit)
AlarmState	Nvarchar	Alarmstatus
TagName	Nvarchar	Name des Objekts, das den Alarm erzeugt hat
Description	Small Integer	Beschreibender text zum Alarm
Area	VarChar(255)	Name des Bereichs oder der Alarmgruppe
Type	Integer	Alarmtyp
Value	Integer	Wert der Alarmvariablen zum Zeitpunkt des Alarms
CheckValue	Real	Alarmgrenzwert zum Zeitpunkt des Alarms
Priority	VarChar(255)	Alarmpriorität
Category	VarChar(255)	Alarmklasse oder Alarmkategorie
Provider	VarChar(255)	Alarmquelle: Intouch-Knoten oder Galaxy-Name
Operator	VarChar(255)	Name des Bedieners
DomainName	VarChar(255)	Name der Domäne
UserFullname	Integer	Vollständige Name des aktiven Bedieners
UnAckDuration	Integer	Die Zeitdauer zwischen dem letzten Alarmübergang und der Quittierung, falls eine solche erfolgt ist
User1 bis User3	VarChar(255)	Benutzerdefinierte Felder
EventStampUTC	VarChar(255)	UTC-Datum/Uhrzeit des Alarmereignisses
MilliSec	Small Integer	Millisekunden
OperatorNode	Small Integer	Name des Rechner, von dem aus der Bediener den Alarm quittiert hat

Tabelle 3 Felder eines Intouch-Alarmdatensatzes

4 Programmiersprache

Zu Beginn jedes Programmierprojektes stellt sich die Frage nach der verwendeten Programmiersprache.
Bei diesem Projekt war die Programmiersprache bereits in der Aufgabenstellung vorgegeben. Das zu entwickelnde Programm soll in Microsoft Visual Basic .NET geschrieben werden. VB .NET kann als Nachfolger von Visual Studio 6 betrachtet werden. Mit der Umstellung auf .NET hat die Visual-Basic-Sprache tiefgreifende Veränderungen erfahren, so dass man von einer neuen Programmiersprache sprechen kann.

Warum das Programm in VB .NET geschrieben werden soll, hat mehrere Gründe:
Zum einen werden alle Anwenderprogramme bei Tuchenhagen Dairy Systems GmbH mit Visual Basic .NET geschrieben. Damit wird auch eine spätere Weiterentwicklung durch andere Mitarbeiter erleichtert.
Zum anderen bietet Visual Basic .NET leistungsstarke Funktionen an, die den Datenbankzugriff deutlich erleichtern. Das liegt daran, dass Microsoft bei Visual Basic den Datenbankzugriff von Version zu Version verbessert hat.

Dass Visual Basic keine hardwarenahe Programmiersprache ist, wie zum Beispiel C, und somit die Programme nicht so schnell und effizient sind, wie die von C, ist für viele Firmen aufgrund der kürzeren Entwicklungszeit und der einfachen Einarbeitung zu ertragen. Außerdem spielt die Performance bei den modernen, leistungsstarken Clientsystemen keine Rolle mehr. Der Hauptfaktor bei der Performance sind vielmehr die SQL Datenbankabfragen und die eventuelle Netzwerkverbindung zum Datenbankserver.

5 Umsetzung

Ein großes Problem der Softwareentwicklung ist die Änderung, Anpassung und Erweiterung vorhandener Systeme in kurzer Zeit. „Allein mit traditionellen funktionalen und prozeduralen Programmiermethoden führt dies zu schwer wartbaren Programmen, die fehleranfällig sind. Ein Großteil des Gesamtaufwands der Softwareentwicklung verlagert sich auf die Wartung und Korrektur von Programmen“ **(Grosche 2002)**. „Zwar hat die objektorientierte Programmierung die Programmierung verändert, dennoch wurde die Softwareentwicklung nur zum Teil erleichtert“ **(Jähnichen 2009)**. Abhilfe soll hier die Idee der komponentenbasierten Softwareentwicklung bringen. Die Komponentenarchitektur soll eine zusätzliche Qualitätssteigerung der Software ermöglichen, indem der Entwicklungsprozess in einfache überschaubare Bausteine unterteilt wird. Dabei soll jeder Baustein eine Teilfunktion des gesamten Prozesses eigenständig realisieren. Um aus den einzelnen Bausteinen eine vollständige Anwendung zusammenbauen zu können, müssen diese mit Hilfe von definierten Schnittstellen miteinander kommunizieren können.

Das zu entwickelnde Softwaresystem wird in drei Softwarekomponenten zerlegt: Konfiguration, Datenerfassung und Anzeige. Das folgende Diagramm zeigt einen vereinfachten Softwareentwurf des vorliegenden Systems. Es stellt die Projektstruktur dar und spezifiziert die einzelnen Komponenten sowie ihre erforderlichen Beziehungen miteinander.

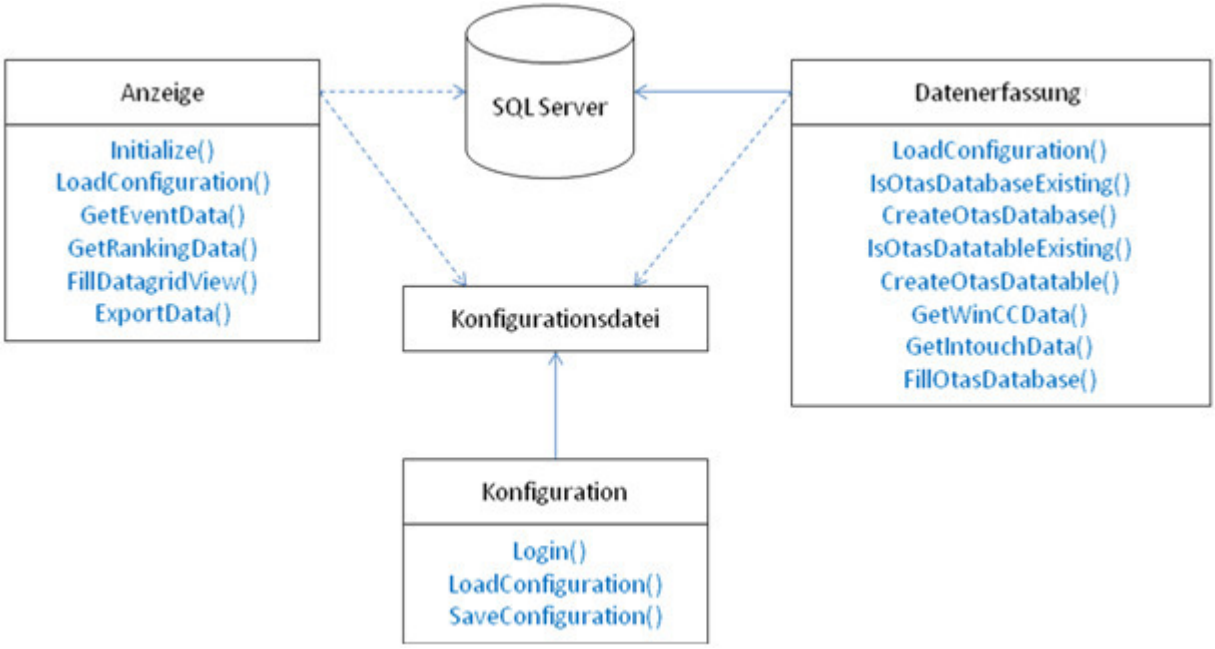

Abbildung 4.1 vereinfachter Softwareentwurf des vorliegenden Systems

Im Folgenden werden die Anforderungen der einzelnen Softwarekomponenten aufgestellt, basierend auf der Aufgabenstellung im Kapitel 2.

Konfiguration

Dieses Programm wird den Namen OTAS.Event Configuration bekommen und folgende Anforderungen erfüllen:

- Es soll die Möglichkeit geben, alle notwendigen Einstellungen für sämtliche Module vorzunehmen. Dazu gehören zum Beispiel der Servername, die Art der Meldungsaktualisierung usw.
- Die Konfiguration muss nur mit den entsprechenden Benutzerrechten editiert werden können.
- Die Einstellungen müssen in einer xml-Datei bzw. Dateien gespeichert werden

Außerdem ist folgende graphische Anforderung zu erfüllen:

Abbildung 5.1 Vorgabe für die Benutzeroberfläche der Konfiguration-Komponente

Datenerfassung

Dieses Programm soll den Namen OTAS.Event Databridge bekommen und folgende Anforderungen erfüllen:

- Es soll die Alarme und Ereignisse aus der Alarmdatenbank von WinCC bzw. Intouch auslesen und in einer neu zu entwerfenden Datenbank speichern.
- Die Meldungsaktualisierung muss sowohl manuell als auch zeitgesteuert erfolgen
- Aufgetretene Fehler müssen angezeigt werden.
- Bei Konfigurationsänderung müssen die Einstellungen aktualisiert werden können.

Außerdem ist folgende graphische Anforderung zu erfüllen:

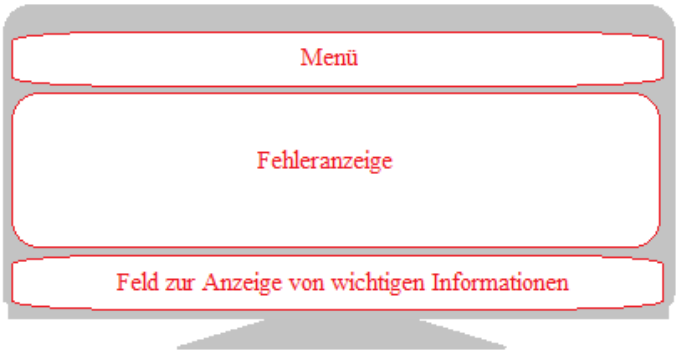

Abbildung 5.2 Vorgabe für die Benutzeroberfläche der Datenerfassung-Komponente

Anzeige

Dieses Programm soll den Namen OTAS.Event Viewer bekommen und folgende Anforderungen erfüllen:

- Alarmmeldungen und Ereignisse in einem vorher ausgewählten Zeitabschnitt müssen tabellarisch angezeigt.
- Das Programm muss über konfigurierbaren Filterfunktionen verfügen.
- Das Programm muss über eine chronologische sowie statistische Ansicht verfügen.
- Meldungen von unterschiedlichen Filtern müssen unterschiedlich eingefärbt werden. Die Filterfarben müssen wiederrum einstellbar sein.

Außerdem ist folgende graphische Anforderung zu erfüllen:

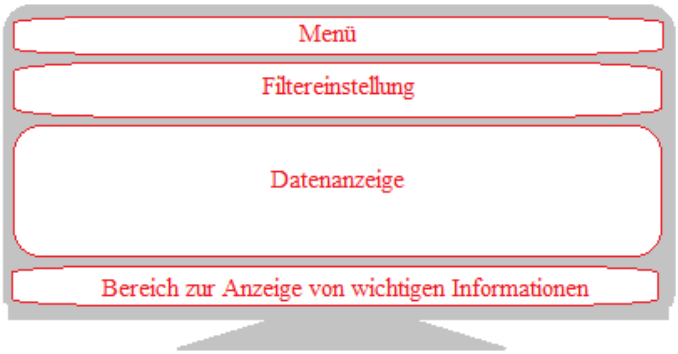

Abbildung 5.3 Vorgabe für die Benutzeroberfläche der Anzeige-Komponente

6 Realisierung

6.1 Anwendungseinstellungen

Windows Forms-Anwendungen nutzen Daten, die zum Ausführen der Anwendung erforderlich sind. In den meisten Fällen ist es nicht sinnvoll, diese Daten direkt in den Code aufzunehmen, da andernfalls die Anwendung jedes Mal neu kompiliert werden müsste, wenn eine Änderung an den Daten vorgenommen wird. Wie unter Kapitel 5.3.1.2 beschrieben wurde, könnte man diese Daten selbst in einer XML-Datei ablegen. Dafür muss man zuerst eine XML-Baumstruktur entwerfen. Das war deswegen erforderlich, weil diese Daten von verschiedenen Anwendungen verwendet werden. Allerdings wenn es darum geht, die Einstellungen einer Softwarekomponente zu speichern, die nur für diese Komponente erforderlich sind, bietet sich als neues Konzept die Anwendungseinstellung an, welches mit dem .NET Framework eingeführt wurde. Dies bringt einige Vorteile in sich:

- Einstellungen sind typsicher.
- Kein Codierungsaufwand ist erforderlich.

Die Funktionsweise von Anwendungseinstellung beruht darauf, dass Daten jeweils im XML-Format in einer CONFIG-Konfigurationsdatei gespeichert werden. Dazu muss man nicht einmal die Datei öffnen, um die gewünschte neue Einstellung eingeben zu können. Stattdessen muss man lediglich den Einstellungs-Designer auf dem Formular für die Projekteigenschaften öffnen und die neuen Einstellungen hinzufügen. Dort kann man den Namen, den Datentyp, den Standardwert und den Bereich (Anwendung oder Benutzer) für jede Einstellung eingeben.

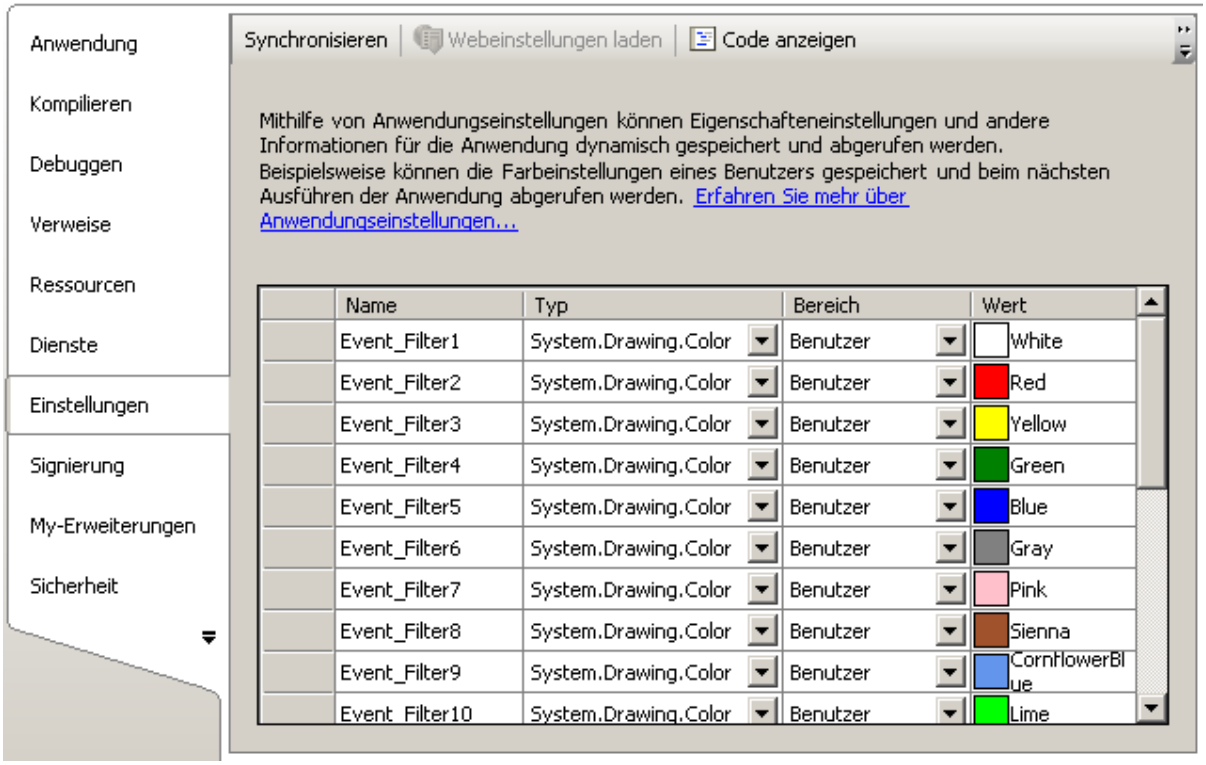

Abbildung 6.1 Anwendungseinstellungen

Beim Hinzufügen einer neuen Einstellung im Einstellungs-Designer werden verschiedene Vorgänge im Hintergrund ausgeführt. Zunächst wird in der Datei <Name der Applikation>**.config** ein Eintrag hinzugefügt, der die Abschnitte zur Definition der Einstellungen definiert.

```
<configSections>
        <sectionGroup name="userSettings"
                      type="System.Configuration.UserSettingsGroup, System,
                            Version=2.0.0.0, Culture=neutral,
                            PublicKeyToken=b77a5c561934e089" >
            <section name="OTAS.Event_Viewer.My.MySettings"
                      type="System.Configuration.ClientSettingsSection,
                            System, Version=2.0.0.0, Culture=neutral,
                            PublicKeyToken=b77a5c561934e089"
                      allowExeDefinition="MachineToLocalUser"
                      requirePermission="false" />
        </sectionGroup>
    </configSections>
```

Im nächsten Schritt wird die Einstellungsdefinition in die Datei eingefügt (siehe unten). In Abhängigkeit vom Typ der definierten Einstellung werden die Abschnitte **applicationSettings** und/oder **userSettings** angezeigt, je nachdem, welche Einstellungen definiert worden sind.

```
<userSettings>
     <OTAS.Event_Viewer.My.MySettings>
         <setting name="Event_Filter1" serializeAs="String">
             <value>White</value>
         </setting>
         <setting name="Event_Filter2" serializeAs="String">
             <value>Green</value>
         </setting>
           .
           .
           .
     </OTAS.Event_Viewer.My.MySettings>
 </userSettings>
 <applicationSettings>
     <OTAS.Event_Viewer.My.MySettings>
         <setting name="ComputerName" serializeAs="String">
             <value />
         </setting>
     </OTAS.Event_Viewer.My.MySettings>
 </applicationSettings>
```

Nachdem eine Einstellung hinzugefügt wurde, kann man sie abrufen und aktualisieren. Der Vorgang zum Abrufen einer Einstellung ist sehr einfach und kann über den Syntax `My.Settings.<Name der Einstellung>` ausgeführt werden.

6.2 Konfiguration

Ziel dieser Softwarekomponente ist, alle notwendigen Einstellungen für das gesamte Softwaresystem vorzunehmen. Die Parameterwerte werden in einer Konfigurationsdatei abgelegt, um sie dann anderen Komponenten zur Verfügung zu stellen. Die Verwendung solcher Anwendung zur Konfiguration des Gesamtsystems bietet im Allgemeinen folgende Vorteile: Es ist keine Modifikation der Software notwendig. Anpassungen in der Konfiguration können daher mit vergleichsweise geringem Aufwand vorgenommen werden und erfordern keine Programmierkenntnisse. Vorteilhaft ist außerdem, dass dieses Konfigurationstool beim Upgrade der Software erhalten bleiben kann.

6.2.1 Aufbau der Applikationsfenster

Beim Ausführen des Programms gelangt man zuerst zum Login Fenster. Um Änderungen in der Konfiguration vorzunehmen und einen sicheren Zugang zu gewährleisten, muss man sich zuerst anmelden.

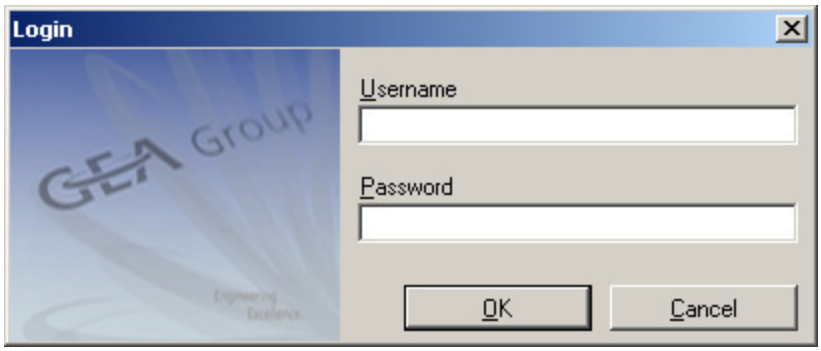

Abbildung 6.2 Anmeldefenster für die Konfiguration

Nachdem die Anmeldung erfolgreich durchgeführt ist, öffnet sich das Konfigurationsfenster. Die Benutzeroberfläche teilt sich in drei Bereichen.

- Allgemeine Einstellungen
- Systemspezifische Einstellungen
- Filtereinstellungen

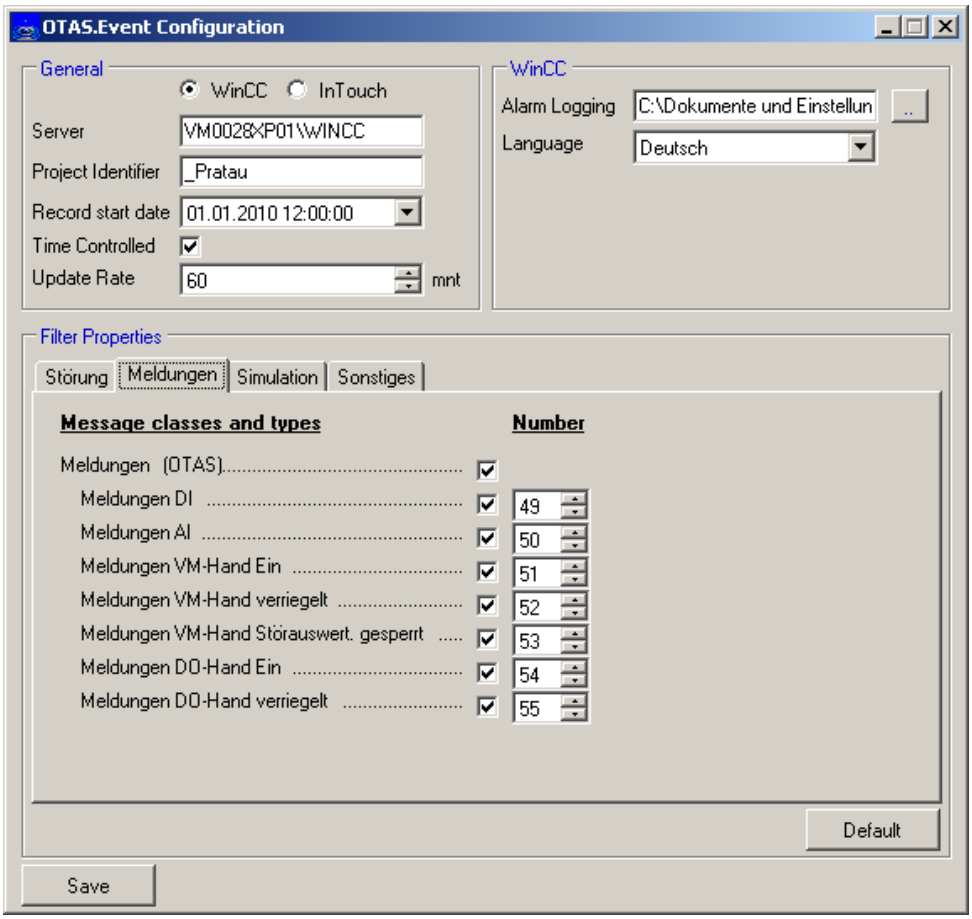

Abbildung 6.3 Konfigurationsfenster

Die einzelnen Parameterwerte werden im folgenden Kapitel erläutert. Es wird an dieser Stelle nur noch darauf eingegangen, wie projektierte Sprachen in WinCC in die Konfiguration hinzugefügt werden können.

In der vorliegenden Applikation ist die Klappliste „*Language*“ vorerst nur mit „*Deutsch*“ vorbelegt. Um die Liste mit weiteren Sprachen zu erweitern, wählt man unter *Language* den Eintrag *New Language* aus.

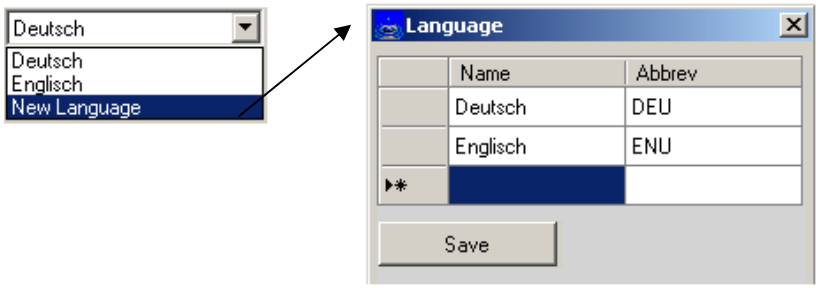

Abbildung 6.4 Hinzufügen bzw. Bearbeiten von Meldesprachen

Dann öffnet sich ein neues Fenster, wo man neue Sprachen und ihre Kennungen hinzufügen bzw. vorhandene Sprachen bearbeiten oder löschen kann.

6.2.2 Konfigurationsdatei

Dieses Kapitel beschränkt sich auf das Abrufen und Aktualisieren von Konfigurationsdaten, die von verschiedenen Programmen verwendet werden. Parameterwerte, die nur für eine Anwendung relevant sind, werden mittels Anwendungseinstellungen gespeichert bzw. ausgelesen, siehe Kapitel 5.1.

Es gibt unter .NET verschiedene Varianten wie man mit Konfigurationsdaten umgehen kann. Zwar kann man diese Daten an vielen verschiedenen Speicherorten ablegen, beispielsweise in INI- oder CSV-Dateien. Allerdings hat jedes der genannten Verfahren den Nachteil, dass die Dateien extrem unflexibel sind und nur einfache bzw. zweidimensionale Datenstrukturen ermöglichen.
Bei diesem Projekt werden die Konfigurationsdaten in XML-Dateien gespeichert. So kann man komplexe Datenstrukturen realisieren und trotzdem einfach auf die Attribute und Knotenelemente zugreifen.

„Ein XML-gerechtes Dokument besteht aus Elementen, Attributen, ihren Wertzuweisungen, und dem Inhalt der Elemente, der aus Text oder aus untergeordneten Elementen bestehen kann, die ihrerseits wieder Attribute mit Wertzuweisungen und Inhalt haben können. Es gibt Elemente mit und ohne Attribute, Elemente, innerhalb deren viele andere Elemente vorkommen können, und solche, innerhalb deren nur Text vorkommen kann, und sogar leere Elemente, die keinen Inhalt haben können.“ **(selfhtml)**. Die Struktur, die aus diesen Bestandteilen und ihren Grundregeln entsteht, lässt sich als Baumstruktur begreifen. In der folgenden Abbildung ist ein Abschnitt der Baumstruktur der Konfigurationsdatei, die im Rahmen dieser Bachelorarbeit entworfen worden ist.

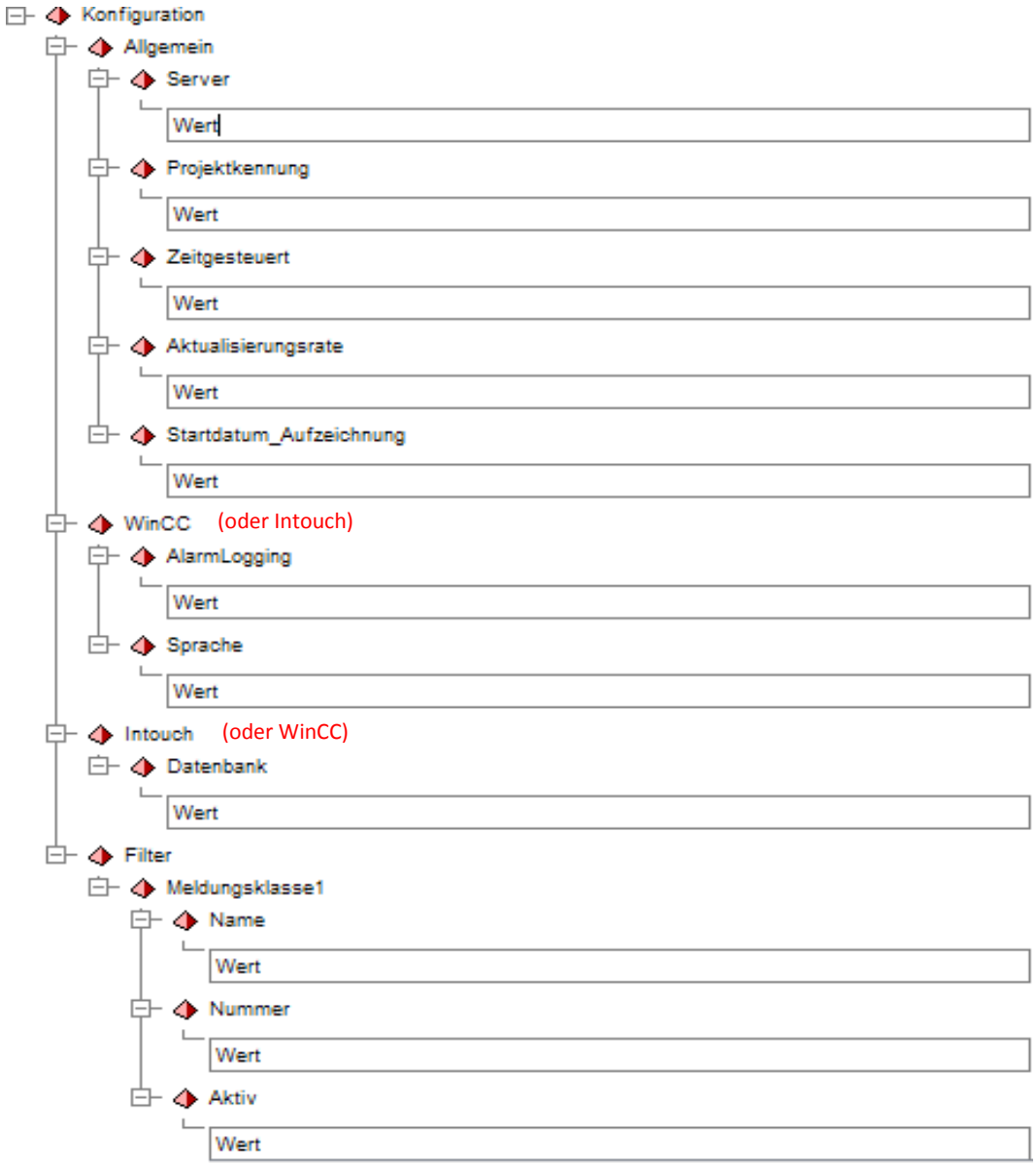

Abbildung 6.5 Baumstruktur der Konfigurationsdatei

In der Abbildung wurde ausgehend von dem Dokument-Element (das äußerten Element) *Konfiguration* drei Kinknoten:

- *Allgemein* hat fünf untergeordnete Knoten, nämlich die Attribute:
  - *Server* : Name des SQL-Servers
  - *Projektkennung* : Projektnamen bzw. die Projektabkürzung
  - *Zeitgesteuert* : ‚True' für zeitgesteuerte und ‚False' für manuelle Aktualisierung der Meldungen.
  - *Aktualisierungsrate* : Zeitintervall für eine neue Meldungsaktualisierung (aktiv nur bei zeitgesteuerten Aktualisierung).
  - *Startdatum_Aufzeichnung* : Anfangsdatum, ab dem die Aufzeichnung startet.

- *WinCC* (Knote existiert nur, wenn WinCC ausgewählt ist) hat zwei untergeordnete Knoten, nämlich die Attribute:
  - *AlarmLogging* : komplettes Pfad des WinCC-AlarmLoggings
  - *Sprache* : projektierte Sprache, um auf die richtige Datenbanktabelle zuzugreifen

- *Intouch* (Knote existiert nur, wenn Intouch ausgewählt ist) hat einen untergeordneten Knoten, nämlich das Attribut:
  - *Datenbank* : Name der im Alarm DB Logger Manager konfigurierten Datenbank

- *Filter* hat 28 unterordnete Knoten (Jeder Knote stellt eine Meldeklasse oder eine Meldeart dar). Da sie dieselben Eigenschaften besitzen, wird nur ein Knote weiter untersucht.
  - *Meldeklasse 1* hat drei untergeordnete Knoten, nämlich die Attribute:
    - *Name* : Name der Meldeklasse bzw. Meldeart
    - *Nummer* : - Falls WinCC, Nummer der Meldeart bzw. Nummernbereich der Meldeklasse (setzt sich aus den Nummern ihrer zugeordneten Meldearten zusammen)
      - Falls Intouch, Prioritätsbereich der Meldeklasse (setzt sich aus den Prioritäten ihrer zugeordneten Meldearten zusammen) bzw. Priorität der Meldeart
    - *Aktiv* : ‚True' wenn die Meldeklasse bzw. Meldeart projektiert ist

Anstelle von "Baumstruktur" zu reden, könnte man auch den Vergleich zu einer Verzeichnis- und Dateistruktur verwenden. Die Datenstruktur einer XML-Datei lässt sich nicht nur genauso abbilden, sondern sie wird von der verarbeitenden Software auch ganz ähnlich gehandhabt. Jedes Element, jedes Attribut, jede Wertzuweisung an ein Attribut, jeder Zeicheninhalt eines Elements wird dabei zu einem eigenen Bestandteil des Baums. Jeder dieser Bestandteile ist einzeln adressierbar.

6.2.3 Schreiben der Konfiguration in die XML-Datei

Basierend auf den Erkenntnissen des vorhergehenden Kapitels wird nun anhand eines Programmabschnittes gezeigt, wie die Daten in die Konfigurationsdatei unter VB.NET geschrieben werden (Es bleibt zunächst auf den Knoten *Allgemein* beschränkt, weitere Knoten werden genauso behandelt und an das XML-Dokument angefügt).

Zunächst wird ein XML-Dokument erzeugt. Mit der Methode LoadXml wird automatisch das Wurzelelement generiert.

```
Dim xmlConfig As New XmlDocument()
xmlConfig.LoadXml("<Konfiguration/>")
```

Der Knoten *Allgemein* und zwei seiner Unterelemente werden deklariert und initialisiert.

```
Dim xmlGeneral As Xml.XmlElement = xmlConfig.CreateElement("Allgemein")
Dim xmlServer As XmlElement = xmlConfig.CreateElement("Server")
Dim xmlProjectName As XmlElement= xmlConfig.CreateElement("Projektkennung")
```

Den beiden Attributen *Server* und *Projektkennung* wird ein Text zugewiesen.

```
xmlServer.AppendChild(xmlConfig.CreateTextNode(tbxServer.Text))
xmlProjectName.AppendChild(xmlConfig.CreateTextNode(tbxProjectName.Text))
```

Dann werden sie an das Ende der Knotenliste des Elternknoten *Allgemein* angefügt.

```
xmlGeneral.AppendChild(xmlServer)
xmlGeneral.AppendChild(xmlProjectName)
```

Das Elternknoten Allgemein wird an das Dokument angehängt.

```
xmlConfig.DocumentElement.AppendChild(xmlGeneral)
```

Die Konfigurationsdaten werden in einer XML-Datei gespeichert.

```
xmlConfig.Save("Konfiguration.xml")
```

Ein wenig anderes wenn es darum geht, die Filtereinstellungen zu speichern. Dafür könnte man jede Meldeklasse bzw. Meldeart separat behandeln. Allerdings entsteht dabei einen sehr langen unübersichtlichen Quellcode. Deshalb werden zu dieser Aufgabe die Einstellungen der Meldeklassen bzw. Meldearten mittels einer Schleife in die Konfigurationsdatei geschrieben. Dabei müsste auf die einzelnen **Controls** mit der Syntax *Me.Controls* zugegriffen werden. Dennoch wenn die Controls einem TabControl-Objekt untergeordnet sind, können sie mit *Me.Controls* nicht angesprochen werden, da sie sich nicht direkt in der Controls-Auflistung des Formulars befinden. Sie befinden sich stattdessen in der Controls-Auflistung für das TabControl-Objekt. Deshalb wird in diesem Fall folgende Syntax verwendet:
Me.TabControl1.TabPages(Registernummer).Controls

Da die Controls nicht direkt mit ihrem Namen angesprochen werden, sondern über die Controls-Auflistung, müssen sie in den entsprechenden Typ (Checkbox, NumericUpDown, usw.) mittels der Casting-Methode **CType** gezwungen werden, um auf die Objekt-Eigenschaften zugreifen zu können.

```
CType(Me.TabControl1.TabPages(iTabCtrlRegisterNr).Controls _
      ("lblMsgType" & i.ToString & "_" & (j + 1).ToString),Label).Text
```

6.2.4 Lesen der Konfiguration aus der XML-Datei

Nachdem im letzten Kapitel gezeigt wurde, wie die Konfigurationsdaten gespeichert werden, wird nun in diesem Abschnitt erklärt, wie diese Daten gelesen werden. Eine Abhilfe soll die Abbildung 6.5 dienen.
Zuerst muss das XML-Dokument aus einer Datei geladen werden.

```
Dim xml As New XmlDocument
xml.Load("Konfiguration.xml")
```

Dann wird ein XML-Element erzeugen. Es stellt das äußerte Element des Dokuments dar.

```
Dim xmlConfig As XmlElement= xml.DocumentElement
```

Nun kann der Inhalt der einzelnen Attribute abgerufen werden, indem man den Pfad in der Baumstruktur folgt. Zum Beispiel:

- *Server* liegt im Pfad „Konfiguration->Allgemein->Server->Wert“

```
tbxServer.Text = xmlConfig.ChildNodes(0).ChildNodes(0).ChildNodes(0).Value
```

- *Projektkennung* liegt im Pfad „Konfiguration->Allgemein->Projektkennung->Wert“

```
tbxProjectName.Text = xmlConfig.ChildNodes(0).ChildNodes(1).ChildNodes(0)._
                      Value
```

Wie unter Kapitel 5.3.1.3 beschrieben wurde, erfolgt auch das Laden der Filtereinstellungen in einer Schleife. Der Name und der Status (Kontrollkästchen aktiviert oder nicht) der Meldungsklasse bzw. Meldeart können ganz einfach aus der Konfigurationsdatei ausgelesen werden.

```
strMsgTypeName = xml.DocumentElement.ChildNodes(2).ChildNodes(i). _
                 ChildNodes(0).ChildNodes(0).Value
bMsgTypeActiv = xml.DocumentElement.ChildNodes(2).ChildNodes(i). _
              ChildNodes(2). ChildNodes(0).Value
```

Handelt es sich um eine Meldeart, wird außerdem die Nummer bzw. Priorität ermittelt.

```
iMsgTypeNumber = xml.DocumentElement.ChildNodes(2).ChildNodes(i). _
                 ChildNodes(1).ChildNodes(0).Value
```

Nun kann das Formular mit den ausgelesenen Parametern aktualisiert werden. Da die Controls über die Controls-Auflistung eines TabControl-Objektes angesprochen werden, muss zuerst die Registernummer ermittelt werden, wo sich die aktuelle Meldeklasse bzw. Meldeart befindet.

```
Select Case i
      Case 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
            iTabCtrlRegisterNr = 0
      Case 10, 11, 12, 13, 14, 15, 16, 17
            iTabCtrlRegisterNr = 1
      Case 18, 19, 20, 21
            iTabCtrlRegisterNr = 2
      Case Else
            iTabCtrlRegisterNr = 3
End Select
```

Nachdem die Registernummer der Meldeklasse bzw. Meldeart ermittelt wurde, kann auf die zugehörigen Controls zugegriffen werden. Dafür müssen sie in den entsprechenden Typ (Checkbox, NumericUpDown, usw.) mittels der Casting-Methode **CType** gezwungen werden.

```
CType(Me.TabControl1.TabPages(iTabCtrlRegisterNr).Controls _
      ("cbxMsgTypeActiv" & i.ToString), CheckBox).Checked = bMsgTypeActiv
CType(Me.TabControl1.TabPages(iTabCtrlRegisterNr).Controls _
      ("nupMsgTypeNr" & i.ToString), NumericUpDown).Value = iMsgTypeNumber
```

6.2.4 Laden der Default-Einstellungen

Falls WinCC ausgewählt ist, steht die Schaltfläche „Default“ zum Laden der Default-Einstellungen zur Verfügung. Dabei handelt es sich um eine Anzahl von Meldeklassen bzw.

Meldearten, deren Parameter vordefinierte Werte haben. Im Gegensatz dazu ist diese Schaltfläche für Intouch nicht bedienbar. Das liegt daran, dass Intouch bei Tuchenhagen Dairy Systems GmbH noch nicht standardisiert ist.

Wie man die Standardwerte in das Formular einbindet, dafür gibt es mehrere Möglichkeiten. Die erste wäre, jeden Parameter einzeln einzugeben. Das wäre sehr mühsam und würde dazu führen, dass der Quellcode schnell sehr lang und unübersichtlich wird. Man kann aber einige Steuerelemente vom gleichen Typ als *Collection* ansprechen und in einer Schleife initialisieren. Das wäre die zweite Möglichkeit. So elegant geht nämlich nur für Checkbox-Objekte, die nur die Werte „True“ oder „False“ haben können. Die NumericUpDown-Steuerelemente haben als Parameter Dezimalwerte. Diese müssen einzeln eingegeben werden.

```
For i As Integer = 0 To Me.TabCtrl.TabPages.Count - 1

    For Each c As Control In Me.TabCtrl.TabPages(i).Controls

        If TypeOf c Is CheckBox Then
            CType(c, CheckBox).Checked = True

        ElseIf TypeOf c Is NumericUpDown Then
            Select Case i
                Case 0
                    Select Case c.Name
                        Case "nupMsgType1_2"
                            CType(c, NumericUpDown).Value = 1
                        Case "nupMsgType1_3"
                            CType(c, NumericUpDown).Value = 2
                        Case "nupMsgType1_4"
                            CType(c, NumericUpDown).Value = 3
                         ..............................
                    End Select
                ..............................
            End Select
        End If
    Next
Next
```

6.3 Datenerfassung

Ziel dieser Softwarekomponente ist, die Meldungen aus der Alarmdatenbank von WinCC bzw. Intouch zu erfasse und in einer neu zu entwerfenden Datenbank zu archivieren. Vorteil dabei ist, dass die Daten zur Verfügung stehen, auch wenn zum Beispiel bei WinCC einige Archivsegmente ausgelagert bzw. gelöscht werden. Dieses Programm kann als Windows-Dienst oder als normale Anwendung ausgeführt werden.

6.3.1 Aufbau der Applikationsfenster

Beim Ausführen des Programms gelangt man direkt zum Hauptfenster.

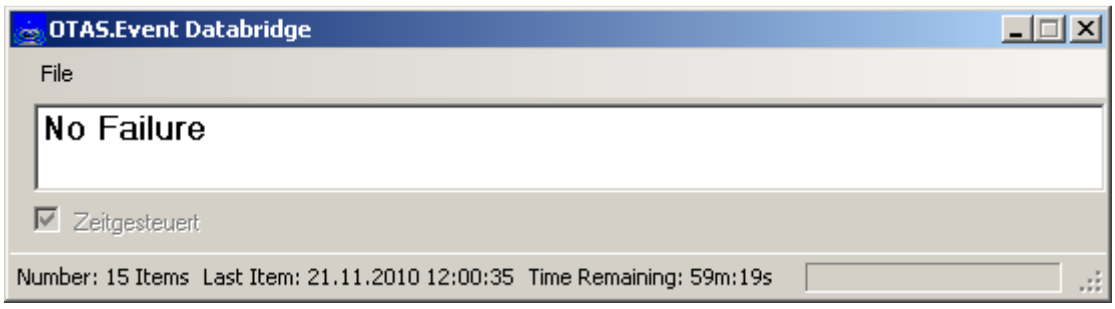

Abbildung 6.6 Datenerfassungsfenster

Die Benutzeroberfläche teilt sich in drei Bereichen.

- Menüleiste

  Sie enthält die notwendigen Befehle, um die Meldungen manuell zu aktualisieren, die Konfigurationsdatei neu auszulesen oder ggf. aufgetretene Fehler aufzuheben.

- Fehleranzeige

  Hier werden alle Datenzugriffsfehler angezeigt. Andere Fehlerquellen sind nicht vorhanden.

- Statusleiste

  Mit Hilfe eines StatusBar-Steuerelements werden sämtliche Informationen über den letzten Kopiervorgang angezeigt, wie zum Beispiel die Anzahl der kopierten Elemente oder der Zeitstempel des letzten Dateneintrags. Die einzelnen Elemente der Statusleiste werden im Kapitel 5.3.8 näher erläutert.

6.3.2 Datenzugriff mittels ADO.NET

„ADO.NET ist die zentrale Datenzugriffstechnologie für das .NET-Framework und gilt als Nachfolger der ActiveX Data Objects (ADO), hat aber nichts mit der ActiveX Technologie zu tun“ **(Wikipedia)**. „In der Tat ist es um zahlreiche Funktionen erweitert worden, so dass man von einer Neuentwicklung sprechen kann“ **(nach Monadjemi, Seite 84)**.
ADO.NET setzt sich aus einer ziemlich komplexen Hierarchie vieler Klassen zusammen. Die daraus erzeugten Objekte lassen sich zunächst in zwei Gruppen aufteilen:

- Provider
- DataSet

Während der Datenprovider die Daten zur Verfügung stellt, ist das DataSet der Teil der Applikation, welche die Dienste eines Datenproviders nutzt, um auf beliebige Daten zuzugreifen, sie zu lesen, zu speichern und zu ändern.

6.3.2.1 ADO.NET-Provider

„Der Datenprovider im .NET Framework kapselt die Datenbank und ermöglicht den Zugriff über einheitliche Schnittstelle, er fungiert als Brücke zwischen einer Anwendung und einer Datenbank und wird zum Abrufen von Daten aus einer Datenbank und zum Abgleichen von Änderungen an diesen Daten verwendet“ **(Doberenz/Gewinnus, Seite 183)**. Dazu muss das Programm eine Verbindung zur Datenquelle herstellen. ADO.NET stellt passende Klassen zur Verfügung, mit denen eine Verbindung aufgebaut und gesteuert werden kann:

- SqlClient-Provider
- OleDb-Provider
- Odbc-Provider
- Oracle-Provider

Bei diesem Projekt wird ausschließlich der SqlClient-Datenprovider benutzt, weil die Datenquelle bei WinCC und InTouch ein Microsoft SQL Server ist. So wird eine hohe Performance gewährleistet, weil dieser Provider einen schnellen Direktzugriff über die API (Application Programming Interface) des SQL Servers ermöglicht.

In der folgenden Tabelle sind die wichtigsten Klassen des SqlClient-Datenproviders und die dazugehörigen Methoden (nicht die Eigenschaften) aufgelistet, die im Rahmen dieser Bachelorarbeit verwendet worden sind.

Klasse	Bedeutung/Konstruktor/Methoden
SqlConnection	Das Objekt stellt die Verbindung zur Datenquelle her. Es benötigt einen *ConnectionString*, der alle erforderlichen Verbindungsparameter kapselt.
	Dim conn As New *SqlConnection* (ConnectionString As *String*)
	Open(), ChangeDatabase(), Close()
SqlCommand	Das Objekt führt eine SQL-Abfrage aus. Es benötigt eine SQL-Abfrage und ein SqlConnection-Objekt.
	Dim cmd As New SqlCommand (sql As String, conn As SqlConnection)
	ExecuteReader(), ExecuteScalar(), ExecuteNonQuery()
SqlDataReader	Das Objekt ermöglicht einen sequenziellen Lesezugriff auf die Datenquelle.
	Kein Konstruktor wird benötigt Dim dr As SqlDataReader = cmd.ExecuteReader()
	Read(), GetValue(), Close()
SqlDataAdapter	Das Objekt ermöglicht das Füllen eines DataSets mit den Ergebnissen einer SQL-Abfrage. Es benötigt eine SQL-Abfrage und ein SqlConnection-Objekt.
	Dim da As New SqlDataAdapter (sql As String, conn as SqlConnection)
	Fill(), Update()

Format des *ConnectionString*

Im ConnectionString werden folgende Parameter benötigt:

Parameter	Bedeutung
Data Source	Der Name des SQL Servers.
AttachDbFilename	Der vollständige Name der gewünschten Datenbank auf dem SQL-Server. Die Datenbank wird im SQL-Server angehängt, falls sie dort nicht angemeldet ist.
Database	Der Name der gewünschten Datenbank auf dem SQL-Server.
Integrated Security	Gibt an, ob es sich um eine SQL Server Authentifizierung (False) oder um eine Integrierte Windows Authentifizierung (True oder SSPI) handelt.

Format des *CommandText* (SQL-Abfrage)

Im CommandText müssen der Name der Datenbanktabelle und ein oder mehrere Abfragebedingungen angegeben werden. Zum Beispiel:

```
SELECT * FROM <ViewName> WHERE <Condition>...
```

Parameter	Beschreibung
ViewName	Name der Datenbanktabelle. - Für Intouch lautet der „ViewName“ **v_AlarmHistory.** - Für WinCC muss die Tabelle in der projektierten Sprache angegeben werden. Der „ViewName“ für die fünf europäischen Sprachen lautet beispielsweise: **ALGVIEWDEU**: deutsche Meldearchivdaten **ALGVIEWENU**: englische Meldearchivdaten **ALGVIEWESP**: spanische Meldearchivdaten **ALGVIEWFRA**: französische Meldearchivdaten **ALGVIEWITA**: italienische Meldearchivdaten
Condition	Filterkriterium z.B.: MsgNr < 1000000 OR MsgNr > 2000000

Tabelle 4 Format der SQL-Abfrage

6.3.2.2 ADO.NET-DataSet

Ein DataSet repräsentiert ein Speicherabbild der zugehörigen Datenbank in der eigentlichen Anwendung. Ein DataSet wird immer dann eingesetzt, wenn Daten mehrmals benötigt und von der Anwendung geändert werden. In diesem Fall werden die Daten über den *DataAdapter* im DataSet gespeichert, wo sie der Anwendung zur weiteren Verwendung zur Verfügung stehen.

In der folgenden Tabelle sind die wichtigsten Klassen im DataSet aufgelistet, die im Rahmen dieser Bachelorarbeit verwendet worden sind.

Klasse	Bedeutung
DataSet	Kernobjekt von ADO.NET, kann als Container für alle anderen untergeordneten Objekte dienen
DataTable	Datentabelle, bestehend Zeilen und Spalten
DataRow	Eine bestimmte Zeile einer DataTable
DataColumn	Eine bestimmte Spalte einer DataTable
DataView	Sicht auf eine DataTable, z.B. für Sortieren und Filtern

Tabelle 5 Wichtige Klassen im DataSet

6.3.2.3 Zusammenspiel der ADO.NET-Klassen

Der Zusammenhang zwischen den einzelnen ADO.NET-Klassen, die in den vorherigen Kapiteln genannt worden sind, wird in der nächsten Abbildung in vereinfachter Form verdeutlicht. Eine Abhilfe dafür soll die Abbildung in **(Doberenz/Gewinnus, Seite 186)** dienen.

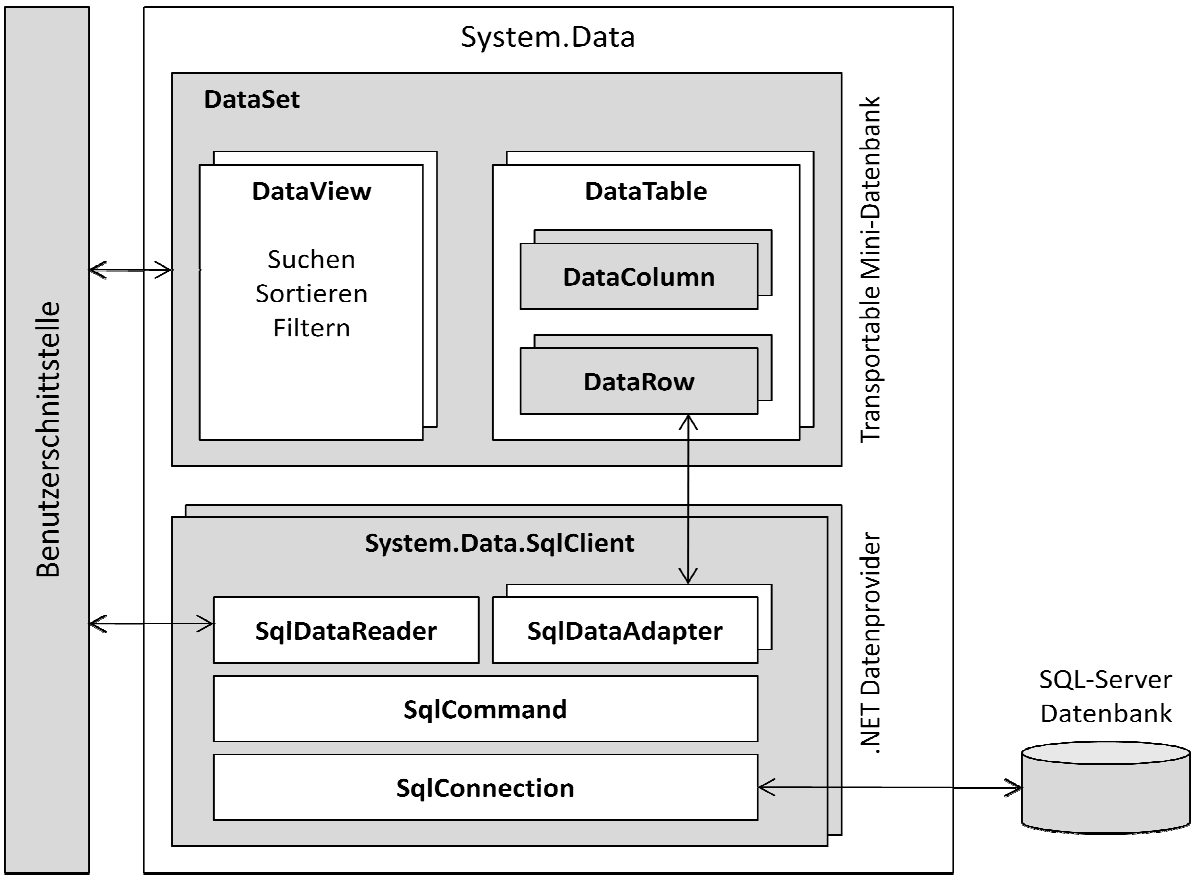

Abbildung 6.7 Zusammenspiel der wichtigen ADO.NET-Klassen

6.3.3 Datenstruktur der neuen Datenbank

In diesem Kapitel geht es darum, eine neue Datenbankstruktur zu entwerfen, die die beiden unterschiedlichen Formate der WinCC- bzw. Intouch-Alarmdatenbank vereinheitlicht. Der Vorteil dabei ist, dass andere Anwendungen, unabhängig vom verwendeten Visualisierungssystem, auf die Daten mit der gleichen Struktur zugreifen können.
Die Entwicklung einer Datenbank vollzieht sich in mehreren Schritten. Zunächst ist festzustellen, welche Informationen in der neuen Datenbank gespeichert werden sollen. Danach muss festgelegt werden, welche Datentypen für die einzelnen Tabellenspalten benötigt werden. Diesen Prozess bezeichnet man als Datenmodellierung. Erst wenn die Datenmodellierung abgeschlossen ist, kann die Tabelle angelegt werden.

Zunächst werden die Spalten der neuen Datenbanktabelle und ihre Datentypen festgelegt.

- *Nummer (int)* : Primärschlüssel der Tabelle (PRIMARY KEY). Er ermöglicht, jeden Datensatz eindeutig zu identifizieren
- *Datum (dateTime)* : Zeitstempel des Alarmmeldung
- *Bereich (varchar(256)) :* Ort der Alarmmeldung
- *Ereignis (varchar(256)) :* Name des Objekts, das den Alarm ausgelöst hat
- *Meldetext (varchar(256)) :* Beschreibender Text zum Alarm
- *Operator (varchar(256)) :* Name des Bedieners
- *Klasse (int) :* Nummer der Meldeklasse (Nur bei WinCC)
- *Typ (int) :* Nummer der Meldeart (Nur bei WinCC)
- *Prioritaet (int) :* Alarmpriorität

	Column Name	Data Type
🔑	Nummer	int
	Datum	datetime
	Bereich	varchar(256)
	Ereignis	varchar(256)
	Meldetext	varchar(256)
	Operator	varchar(256)
	Klasse	int
	Typ	int
	Prioritaet	int

Abbildung 6.8 Spaltendefinition der neuen Datenbanktabelle

Im Folgenden werden die SQL-Befehle vorgestellt, um eine neue Datenbank bzw. eine neue Datenbanktabelle zu erstellen.

Anlegen einer Datenbank

Mit dem SQL-Befehl

```
CREATE DATABASE <Name der Datenbank> ON PRIMARY (
     NAME = OtasEvent_Data,
     FILENAME = <Dateiname mit vollständigem Pfad>,
     SIZE = 4MB,
     FILEGROWTH = 10%)
```

kann die neue SQL-Datenbank erzeugt werden. Sie besitzt folgende Einstellungen: 4MB Anfangsgröße, 10% automatische Vergrößerung und unbeschränkte Dateigröße.

Erstellen einer Datenbanktabelle

Nach dem Erstellen der Datenbank wird die gewünschte Tabelle Durch den SQL-Befehl erstellt.

```
CREATE TABLE OtasEventDataTable(
     Nummer      int PRIMARY KEY,
     Datum       Datetime,
     Bereich     varchar(256),
     Ereignis    varchar(256),
     Meldetext   varchar(256),
     Operator    varchar(256),
     Klasse      int,
     Typ         int,
     Prioritaet  int)
```

6.3.4 Selektion der WinCC-Archivsegmenten

Ab WinCC V6.0 ist die WinCC-Runtime-Datenbank segmentiert. Das bedeutet, die Daten werden in mehreren Archivsegmenten (mehreren Datenbanken). In diesem Kapitel wird erklärt, wie die Archivsegmente in einem vorgegeben Zeitabschnitt selektiert werden. Hierzu sind zwei Fälle zu beachten:

1- Das Archivsegment ist abgeschlossen und sein Dateiname ist mit einem Anfangs- und Enddatum versehen oder es ist aktuell und sein Dateiname ist nur mit dem Anfangsdatum versehen. Hierzu wird das aktuelle Datum als Enddatum genommen.
   ⇨ In diesem Fall ist die Selektion eine reine Zeitselektion.

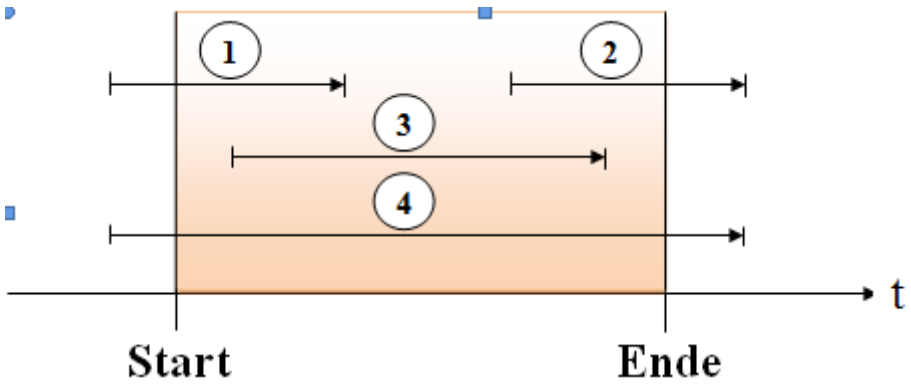

Abbildung 6.9 Zeitabschnitt mit WinCC-Archivsegmenten

- Im ersten Fall ist der Startzeitpunkt des Archivsegmentes vor dem Zeitabschnitt und der Endzeitpunkt im Zeitabschnitt.
- Im zweiten Fall ist es genau umgekehrt. Da ist der Startzeitpunkt des Archivsegmentes im Zeitabschnitt und der Endzeitpunkt außerhalb.
- Der dritte Fall zeigt ein Archivsegment, das vollkommen im Zeitabschnitt liegt.
- Der vierte Fall zeigt ein Archivsegment, das über den Zeitabschnitt hinweg läuft.

In der Software werden die Fälle 1 und 3 bzw. 2 und 3 jeweils zu einem zusammengefasst. Nämlich wird nur das Anfangsdatum bzw. das Enddatum betrachtet, ob es im Zeitabschnitt liegt.

Die Fälle 1 und 3:

```
DateTime.Compare(listOfWinccDBs(i).endDate, startDate) >= 0)  And _
DateTime.Compare(listOfWinccDBs(i). endDate, endDate")) <= 0
```

Die Fälle 2 und 3

```
DateTime.Compare(listOfWinccDBs(i).startDate, startDate) >= 0)  And _
DateTime.Compare(listOfWinccDBs(i).startDate, endDate")) <= 0
```

Der Fall 4

```
DateTime.Compare(listOfWinccDBs(i).startDate, startDate) <= 0  And _
DateTime.Compare(listOfWinccDBs(i).endDate, endDate") >= 0
```

2- Das Archivsegment ist aktuell und wird von der WinCC-Applikation noch beschrieben. Sein Dateiname ist aber mit dem Anfangs- und Enddatum versehen. Dies geschieht, wenn zum Beispiel die Runtime beendet wurde. Eine Zeitselektion würde dazu führen, dass neu aufgetretene Meldungen in diesem Archivsegment nicht mehr berücksichtigt werden könnten. Daher wird außerdem eine Namensselektion durchgeführt. Dies führt dazu, dass der Dateiname des aktuellen Archivsegments gespeichert werden muss.

```
String.Compare(listOfWinccDBs(i).strName, My.Settings. _
               WinccActualSegment)
```

Ist eines der oben aufgeführten Fälle erfüllt, so wird eine Verbindung zur entsprechenden Datenbank hergestellt und die Meldungen in dem vorgegeben Zeitabschnitt ausgelesen und in einem DataTable-Objekt zwischengespeichert.

```
Private Function WinccGetData(ByVal startDate As Date, _
                              ByVal endDate As Date)
                              As DataTable
```

6.3.5 Interpretation der Platzhalter im AlarmLogging

In der Anlage sind meistens so viele Objekte (Ventile, Motoren, Frequenzumrichter usw.), dass man nicht für jedes einzelne eine Meldung projektieren kann. Dazu und um die Größe der Meldungsdatenbanken zu optimieren, enthalten die in WinCC auftretenden Meldungen Platzhalter. Dadurch kann gewährleistet werden, dass die statischen Informationen der Meldung nicht wiederholt abgespeichert werden müssen.

Die folgende Abbildung zeigt als Beispiel die drei projektierten Meldungen der Meldeklasse „*Bedienmeldung*“.

	Nummer	Klasse	Art	Herkunft	Bereich	Ereignis	Meldetext	Operator
	11	Bedienmeldung	Bediener Aktion	@1%-s@	@2%-s@	@3%-s@	@4%-s@	@5%-s@
	12	Bedienmeldung	Bediener Aktion			@1%-s@	@2%-20.20s@@3%-30.30s@@4%-20.20s@	
	13	Bedienmeldung	Bediener Aktion	@1%-s@	@2%-s@	@3%-s@	@4%-s@@5%s@: @6%s@ --> @7%s@	@8%-s@

Abbildung 6.10 Meldungsprojektierung unter WinCC mittels Platzhalter

Da diese Applikation dem Meldesystem von WinCC nachempfunden ist, müssen diese Platzhalter entsprechend interpretiert werden. Platzhalter, die vom Standard (siehe folgende Beschreibung) abweichen, können nicht ersetzt werden. Stattdessen werden sie unverändert in den zugehörigen Meldungsparameter (*Herkunft*, *Bereich*, *Ereignis*, *Meldetext* oder *Operator*) eingetragen.

Die Zusammensetzung dieser Platzhalter wird in der folgenden Tabelle anhand vom folgenden Beispiel erklärt: *@3%-s@*

Ausschnitt	Beschreibung
„@“	Eröffnendes Zeichen
„3“	Hier wird die Prozessgröße bestimmt, deren Wert eingefügt wird.
„%“	Trennzeichen
„-“	Hier wird die Formatierung des einzuführenden Prozesswertes festgelegt. In diesem Fall wird der Prozesswert linksbündig in den Anwendertextblock eingefügt.
„s“	Legt den Datentyp des einzufügenden Prozesswertes fest. In diesem Fall handelt es sich um einen Text.
„@“	Schließendes Zeichen

Tabelle 6 Interpretation der Platzhalter

Der Meldungsparameter kann aus mehreren Platzhaltern bestehen, wie beispielsweise *„@4%-s@@5%-s@:@6%-s@-->@7%-s@“*. Diese müssen dann zusammengefügt werden, um daraus aussagekräftige Texte für den entsprechenden Meldungsparameter ableiten zu können. Für dieses Beispiel gilt folgender Quellcode:

```
row_New ("Meldetext") = strProcessValueBlock(3) &
                        strProcessValueBlock(4) & " : " & _
                        strProcessValueBlock(5) & " --> " & _
                        strProcessValueBlock(6)
```

Sollte es sich um Meldungen handeln, die gar keinen Platzhalter enthalten. So werden die Meldungstexte direkt in den Anwendertextblöcken eingetragen. Man kann sie dann aus den Spalten (Text1, Text2, …, Text5) aus der Datenbanktabelle wieder auslesen.

```
If Not (row("Text2") Is DBNull.Value) Then
    row_New ("Bereich") = row("Text2")
Else
    row_New ("Bereich") = ""
End If
```

6.3.6 Behandlung eines Datenbankzugriffsfehler

Beim Zugriff auf die WinCC-Datenbanken, kann es manchmal zu Datenzugriffsfehlern kommen. Dies ist dann der Fall, wenn zum Beispiel die WinCC-Runtime abgestürzt ist. Bei diesen Fehlern handelt es sich darum, dass man keine Leseberechtigung auf die WinCC-Archivsegmente hat und somit wird jeder Zugriff darauf verweigert und folgende Fehlermeldung ausgegeben: ***„Es kann auf die Datei nicht zugegriffen werden, da sie von einem anderen Prozess verwendet wird“.***

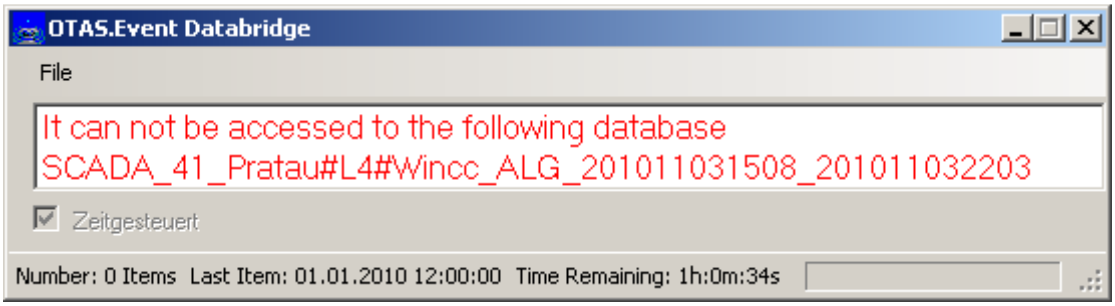

Abbildung 6.11 Anzeige eines Datenzugriffsfehler auf ein WinCC-Archivsegment

Um die aufgetretenen Fehler aufzuheben, wurde eine Funktion implementiert. Sie wird aufgerufen, indem man in der Menüleiste *File* den Eintrag *Remove Failure* auswählt. Sie dient dazu, die Verbindung mit den entsprechenden Datenbanken nochmal herzustellen und die fehlenden Einträge auszulesen. Ob dies erfolgreich ausgeführt wurde oder nicht, wird dann in einer entsprechenden MessageBox ausgegeben.

6.3.7 Selektion der Intouch-Datensätze

In diesem Kapitel wird erläutert, wie neue Datensätze in der Intouch-Alarmdatenbank ermittelt werden. Aufgrund der internen Datenstruktur von Intouch erfolgt das Kopieren der Meldungen nicht wie bei WinCC, wo der Zeitstempel des letzten Eintrags gespeichert wird. Stattdessen wird bei jedem Kopiervorgang einen arithmetischen Vergleich zwischen der Anzahl der Datensätze in der neuen Datenbank `iNumberOfOtasDbItems` und der in der Intouch-Alarmdatenbank `iNumberOfIntouchDbItems` durchgeführt. Dazu wird die SQL-Funktion Count verwendet.

Anzahl der Elemente in der neuen Datenbank ermitteln

Nachdem eine Verbindung zur OTAS Datenbank hergestellt wurde, wird mittels der SQL-Abfrage `SELECT COUNT(*) FROM <OtasEventDataTable>` die Anzahl der Zeilen in der neuen Datenbanktabelle `iNumberOfOtasDbItems` ermittelt.

Anzahl der Elemente in der Intouch-Datenbank ermitteln

Auf der gleichen Weise wird die Anzahl der Elemente in der Intouch Alarmdatenbank `iNumberOfIntouchDbItems` mittels des SQL-Befehls `SELECT COUNT(*) FROM v_AlarmHistory` ermittelt.

`iNumberOfOtasDbItems` **und** `iNumberOfIntouchDbItems` **vergleichen**

Nachdem die Anzahl der Elemente sowohl in der neuen Datenbank als auch in der Intouch-Datenbank ermittelt wurden, wird nun einen arithmetischen Vergleich durchgeführt. Besitzen die beiden Datenbanktabelle die gleiche Anzahl an Zeilen, so werden keine Elemente kopiert. Hat aber stattdessen die Intouch-Datenbanktabelle mehr Einträge, so werden die letzten Elemente ausgelesen und in die neue Datenbanktabelle geschrieben.

```
SELECT  TOP (iNumberOfIntouchDbItems – iNumberOfOtasDbItems)
EventStamp,Area,TagName,Description,Operator,Priority FROM
v_AlarmHistory
```

6.3.8 Aktualisierung der Statusleiste

In diesem Kapitel werden die einzelnen Felder in der Statusleiste erläutert.

Anzahl der kopierten Elemente

Hier wird dem Bediener zur Anzeige gebracht, wie viele Datensätze beim letzten Kopiervorgang in die neue Datenbank geschrieben worden sind.

```
txtNrOflLastItems.Text = "Number: " & dt.Rows.Count & " Items"
```

Zeitstempel des letzten kopierten Eintrags

Es ist ganz wichtig, wenn Meldungen in die neue Datenbank geschrieben werden, dass der Zeitstempel des letzten Eintrags gespeichert wird. Das hat den Vorteil, dass beim nächsten Kopiervorgang nur die neusten Meldungen aus der WinCC-Alarmdatenbank berücksichtigt werden. Bei Intouch ist der Zeitstempel irrelevant, Siehe Kapitel 5.3.6.

```
strSQL = "SELECT TOP (1) Datum FROM " & strOtasEventDataTable &
         " ORDER BY Datum DESC"
com = New SqlCommand(strSQL, conn)
dr = com.ExecuteReader()
While dr.Read
     My.Settings.LastRecordDate = dr.GetDateTime(0)
End While
dr.Close()
```

Dieser Zeitstempel wird dem Bediener als Text in der Statusleiste ausgegeben.

```
txtDateOfLastItem.Text = "Last Item: " & My.Settings. _
                                         LastRecordDate
```

Verbleibende Zeit bis zur nächsten Meldungsaktualisierung

In der Statusleiste erfolgt ebenfalls eine Zeitangabe. Damit wird der Bediener über die verbleibende Zeit bis zur nächsten Meldungsaktualisierung informiert. Hierzu wurde ein Timer implementiert, welchem ein Zeitintervall in Millisekunden übergeben werden muss.

```
TimerTextUpdate.Interval = 1000
```

Die verbleibende Zeit wird jede Sekunde neu berechnet und somit die Anzeige aktualisiert. Zuerst wird die vergangene Zeit seit der letzten Meldungsaktualisierung ermittelt und vom konfigurierten Zeitintervall abgezogen.

```
span = Now.Subtract(TimerLastMsgUpdate)
iTimeRemainingToUpdate = (TimerMsgUpdate.Interval / 1000) - _
                         (span.days*86400 + span.Hours*3600 + _
                          span.Minutes*60 + span.Seconds)
```

Die Variable `iTimeRemainingToUpdate` gibt die verbleibende Zeit in Sekunden an. Diese wird umgerechnet und als Text ausgegeben. Hat zum Beispiel die Variable einen Wert, der größer als 60 und kleiner als 3600 ist, so wird folgender Text ausgegeben.

```
TimeRemaining.Text = "Time Remaining: " & iMinute & "m:" & _
                                          iSecond & "s"
```

Progressbar

Damit der Benutzer bei einer großen Anzahl von Meldungen und einer ggf. langsamen Verbindung zur Datenbank sieht, dass die Anwendung noch arbeitet und nicht abgestürzt ist, wurde ein Fortschrittbalken implementiert. Er wird bei jedem Kopiervorgang der Meldungen aus der WinCC- bzw. Intouch-Alarmdatenbank in die neue Datenbank aktiv.
Für den Fortschrittbalken gibt es in Visual Basic .NET ein fertiges Steuerelement, welchem drei Parameter übergeben werden müssen: sein Minimum, sein Maximum und sein aktueller Wert, wobei das Minimum immer auf Null gesetzt wird.
Der Fortschrittbalken berechnet selbst, wie weit der Balken angezeigt werden soll. Wenn zum Beispiel als Maximum 10 eingegeben wird und als aktueller Wert 5 vorliegt, so zeigt das Steuerelement einen Balken an, der die Hälfte des Steuerelementfensters einnimmt.
Nachdem die Meldungen aus der Alarmdatenbank ausgelesen worden sind, werden sie in einem DataTable-Objekt zwischengespeichert. Vor seiner Übergabe an die neue Datenbanktabelle, wird seine Zeilenanzahl zuerst ermittelt. Diese wird dem Fortschrittbalken als Maximum übergeben.

```
ProgressBar1.Maximum = dt.Rows.Count
```

Der aktuelle Wert wird bei der Bearbeitung nach jeder Zeile inkrementiert.

```
ProgressBar1.Value += 1
```

6.4 Anzeige

Ziel dieser Softwarekomponente ist, Alarmmeldungen und Ereignisse aus der neu erstellten Datenbanktabelle auszulesen und in einem ausgewählten Zeitabschnitt mit Hilfe eines *Datagridview*-Steuerelements tabellarisch anzuzeigen. Um eine detaillierte und aussagekräftige Auswertung der Meldungen zu ermöglichen, verfügt die Anwendung über zahlreiche Filterfunktionen.

6.4.1 Aufbau der Applikationsfenster

Nachdem die Anwendung gestartet wurde, erscheint nach einem Begrüßungsbildschirm das folgende Programmfenster „OTAS.Event Viewer“.

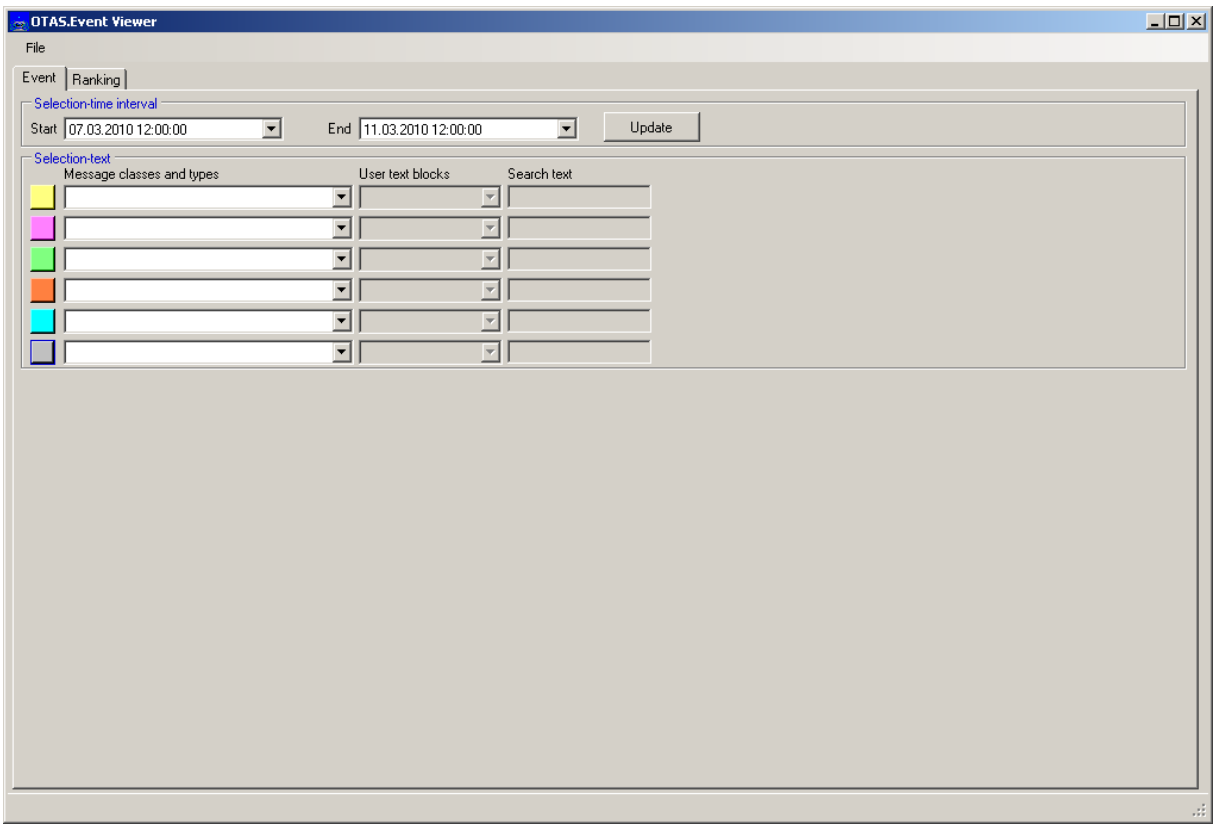

Abbildung 6.12 Event-Ansicht beim Start der Anwendung

Das Programm besteht aus einem Funktions- und Anzeigebereich. Im Folgenden werden die einzelnen Objekte erläutert.

Menüleiste

An dieser Stelle kann der Anwender die Daten sowohl aus der Event- als Ranking-Ansicht in eine Excel-Datei exportieren. Außerdem besteht hier die Möglichkeit, eingestellte Filterfarben zu speichern.

Gruppe „Selection-time interval“

Die Objekte dieser Gruppe dienen der Vorgabe eines Zeitintervalls, das als Filterkriterium zur Abfrage der archivierten Daten verwendet wird. Die Zeit wird im lokalen Uhrzeitformat vorgegeben.

Schaltfläche „Update“

Mit dem Klick auf die Schaltfläche „Update“ werden die Meldungen in dem eingestellten Zeitabschnitt aus der Datenbank ausgelesen. Nach dem Auslösen des Lesebefehls wird zunächst die SQL-Abfrage erstellt und ausgeführt. In dieser Zeit ist die Applikation nicht reaktionsfähig, da sie auf eine Antwort des SQL-Servers warten muss. Je nach Anzahl der angeforderten Daten, kann dies bis zu einigen Minuten in Anspruch nehmen. Während dessen kann aus Sicherheitsgründen die Schaltfläche nicht mehr bedient werden.
Nach dem Auslesen der Daten wird die Anzahl der ausgelesenen Einträge als Text in der Statusleiste angezeigt und die Bedienung der Schaltfläche wird wieder freigegeben.

Gruppe „Selection-text“

Die Objekte dieser Gruppe dienen dazu, Filtereinstellungen vorzunehmen. Dadurch können die zurück gelieferten Daten auf diejenigen reduziert werden, wofür man sich bezüglich Art oder Herkunft der Meldungen zum Beispiel tatsächlich interessiert.

Registermappe

Es steht die Registerseite „Event“ für eine chronologische Anzeige sowie die Registerseite „Ranking“ für eine statistische Anzeige (Häufigkeit) zur Verfügung. Jede Ansicht hat ihren eigenen Funktions- und Anzeigebereich, so dass man sie unterschiedlich einstellen kann. Dadurch werden die Ergebnisse unterschiedlicher Ansichten nicht voneinander beeinflusst.

6.4.2 Filterfunktionen

Für die beiden Ansichten stehen Filterfunktionen zur Verfügung. Sie dienen dazu, den Suchbereich einzugrenzen, um dann einfache Datenauswertung durchführen zu können.

Filterinitialisierung

Hierfür müssen zuerst die Filterkriterien festgelegt werden. Dabei handelt es sich hauptsächlich um zwei Arten von Filtern:

1- Filter zum Auswählen eines Anwendertextblockes

2- Filter zum Auswählen einer Meldeklasse bzw. Meldeart

Bei dem ersten Filter werden die fünf Anwendertextblöcke in die entsprechenden Checkbox-Objekte direkt eingetragen.

Bei dem zweiten Filter werden die filterspezifischen Einstellungen aus der Konfigurationsdatei ausgelesen und in einem DataTable-Objekt zwischengespeichert.

```
dt.Columns.Add(New DataColumn("Meldeart",System.Type.GetType("System.String")))
dt.Columns.Add(New DataColumn("aktiv",System.Type.GetType("System. Boolean ")))
dt.Columns.Add(New DataColumn("Nummer",System.Type.GetType("System.String")))
```

Dann werden nur die Meldeklassen bzw. Meldearten als Filterkriterien angezeigt, deren Parameter „*aktiv*“ gleich *True* ist.

```
For Each row In dt.Rows
    If (row("aktiv") = True) Then
        For i As Integer = 1 To 15
            CType(Me.gbxEventFilter.Controls("cbxEventFilter" & _
                    i.ToString), ComboBox).Items.Add(row("Meldeart"))
        Next
    End If
Next
```

Filtereinstellung

Nachdem die Filter initialisiert wurden, können sie für die Datensuche verwendet werden.

Es muss mindestens eine Meldeklasse bzw. Meldeart ausgewählt werden, um dann die weiteren zugehörigen Filter für den Anwendertextblock oder den Suchtext freizugeben.

Jedes Filter ist mit einem Button-Objekt versehen, um die Filterfarbe einzustellen.

Abbildung 6.13 Filtereinstellung

Zunächst wird eine SQL-Abfrage für den ungefilterten Fall erstellt. Dabei handelt es sich nur Meldungen in dem ausgewählten Zeitintervall. Die Ergebnisse werden in dem DataTable-Objekt *dtEvent* gespeichert.

```
strSQL = "SELECT * FROM " & strOtasEventDataTable & " WHERE " & _
         "Datum >= '" & dtpFromEvent.Value & "' AND " & _
         "Datum <= '" & dtpToEvent.Value & "'"
```

Sind Filterfunktionen aktiv, so wird für jeden aktiven Filter eine SQL-Abfrage gebildet. Mit Hilfe der *Rowfilter*-Methode werden die Daten aus *dtEvent* gefiltert und in einer neuen Tabelle gespeichert *dtEventF*. Sie hat die gleiche Struktur wie die von *dtEvent*, bis auf eine zusätzliche Spalte für Farbe, damit bei jedem Eintrag in die Tabelle gleich die entsprechende Farbe hineingeschrieben wird. Sonst würde man zum Schluss die Daten keinem Filter zuordnen.

```
dvEvent = New DataView(dtEvent)      ‘ DataView für Filterfunktionen
dtEventF = dtEvent.Clone             ‘ Datenstruktur kopieren

                                     ‘ neue Spalte hinzufügen
dtEventF.Columns.Add(New DataColumn("Farbe",
                     System.Type.GetType("System.Integer")))
. . .
dvEventF.RowFilter = strRowFilter(i)‘ SQL-Abfrage des i.ten Filterns


'Ergebnisse in dtEventF speichern und Filterfarbe eintragen
For k As Integer = 0 To dvEventF.ToTable.Rows.Count - 1
    dtEventF.ImportRow(dvEventF.ToTable.Rows(k))
    dtEventF.Rows(dtEventF.Rows.Count - 1)("Farbe") = iFilterColor
Next k
```

Um die Daten dem DataGridView-Control zu übergeben, ist folgende Zeile notwendig

```
dgvEvent.DataSource = dtEventF
```

Dabei müssen die einzelnen Zeilen anhand ihrer Quelle eingefärbt werden.

```
For i As Integer = 0 To dgvEvent.Rows.Count - 1
    dgvEvent.Rows(i).DefaultCellStyle.BackColor = Color.FromArgb _
                                                  (dtEventF.Rows(i)("Farbe"))
Next
```

6.4.3 Berechnung der Meldungshäufigkeit

In diesem Kapitel geht es darum, wie die Häufigkeit der Alarmmeldungen für die Ranking-Ansicht berechnet wird. Zunächst werden die Daten nach *Ereignis* und *Meldetext* sortiert. Dies ist ausreichend, um gleiche Einträge zu eliminieren und die Häufigkeit zu berechnen. Danach wird wird jede Zeile mit der nächsten auf *Ereignis* und *Meldetext* verglichen. Sind die beiden gleich, so wird der erste Eintrag ignoriert und die Häufigkeit um eins erhöht, bis zwei verschiedene Einträge detektiert werden. In diesem Fall wird die entsprechende Zeile mit der zugehörigen Häufigkeit in ein neues DataTable-Objekt eingetragen.

```
iCounter = 1
dtRkgToGrid = dtRankingF.Clone
dvRankingF = New DataView(dtRankingF)
dvRankingF.Sort = "Ereignis,Meldetext"

For i As Integer = 0 To dvRankingF.ToTable.Rows.Count - 1
        If (dvRankingF.ToTable.Rows(i)("Ereignis") = _
            dvRankingF.ToTable.Rows(i + 1)("Ereignis") And _
            dvRankingF.ToTable.Rows(i)("Meldetext") =
            dvRankingF.ToTable.Rows(i + 1)("Meldetext")) Then

                iCounter = iCounter + 1
        Else
            dtRkgToGrid.ImportRow(dvRankingF.ToTable.Rows(i))
            dtRkgToGrid.Rows(dtRkgToGrid.Rows.Count - 1)("Häufigkeit") = iCounter
            iCounter = 1
        End If
Next
```

6.4.4 Steuerelementfelder

Wenn man ein Steuerelement zu einem Formular hinzufügt, so bekommt es zusätzlich zu seiner Bezeichnung automatisch eine Nummer, wie beispielsweise: *Steuerelement**1***. Falls noch ein zweites hinzukommt, so bekommt er den Namen: *Steuerelement**2***. Dies setzt sich so weiter fort, bis man N Steuerelemente hat. Sollte es durch das Klicken eines dieser Steuerelemente immer das gleiche Ereignis ausgelöst werden, so müssten N gleiche Ereignisse geschrieben werden.

*Steuerelement**1**.Ereignis*
*Steuerelement**2**.Ereignis*
.
.
*Steuerelement**N**.Ereignis*

Es gibt aber noch eine zweite Art, wie man gleiche Ereignisse zusammenfassen kann. Der Vorteil dabei ist, dass deutlich weniger Quellcode geschrieben werden muss. Die Syntax dafür wird anhand eines Beispiels gezeigt, wo durch das Klicken eines Buttons ein Color-Dialogfenster aufgerufen wird, um die Hintergrundfarbe des aufrufenden Objektes einzustellen.

```
Private Sub btnColorRkgFilter_Click(ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles btnColorRkgFilter1.Click, _
                                            btnColorRkgFilter2.Click, _
                                            btnColorRkgFilter3.Click, _
                                            btnColorRkgFilter4.Click, _
                                            btnColorRkgFilter5.Click

    Dim btn As Button
    btn = CType(sender, Button)

    If (ColorDialog1.ShowDialog() = Windows.Forms.DialogResult.OK) Then
        btn.BackColor = ColorDialog1.Color
    End If

End Sub
```

6.4.5 Testergebnisse

In den folgenden Abbildungen sind Testergebnisse der beiden Ansichten.

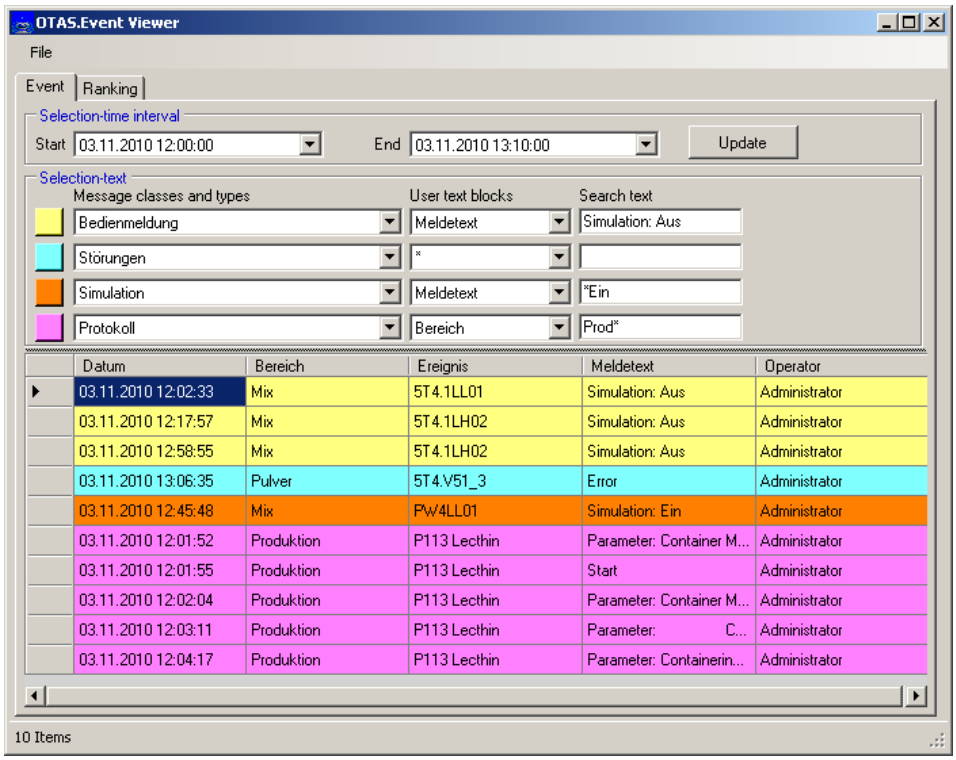

Abbildung 6.14 Testergebnisse der Event-Ansicht

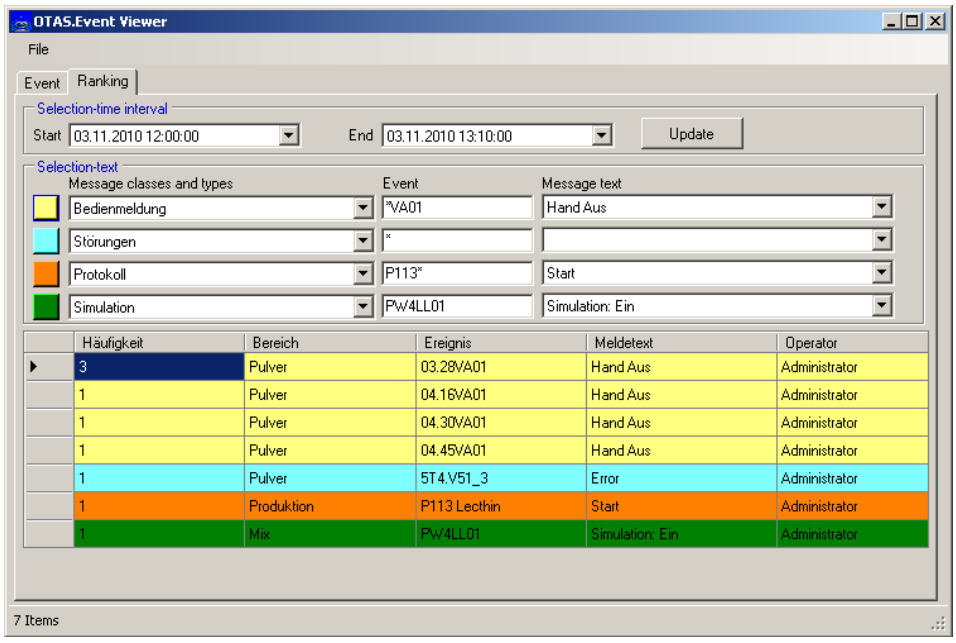

Abbildung 6.15 Testergebnisse der Ranking-Ansicht

6.4.6 Export der Daten in eine CSV-Datei

In diesem Kapitel wird darauf eingegangen, wie sich der gesamte Inhalt eines DataGridView-Controls in ein neues Excel-Dokument einfügen lässt. Dies ist für Dokumentationszwecken sowie für Druckfunktionen sehr nützlich. Nachteil dabei ist, dass der Anwender Microsoft Office auf dem Rechner haben muss.

Durch das Auswählen in der Menüleiste den Eintrag „*Export*“, dann *EventData* oder *RankingData,* wird die Export-Funktion mit den erforderlichen Parametern aufgerufen.

```
Call Export(DataGridView, Filepath, Delimiter)
```

Innerhalb der Funktion haben folgende Zeichen eine Sonderfunktion zur Strukturierung der Daten.

- Trennzeichen

  Dieses Zeichen wird zur Trennung von Datenfeldern (Spalten) innerhalb der Datensätze benutzt. Allgemein wird dafür ein Semikolon oder ein Komma verwendet.
- Qualifizierer

  Hierbei handelt es sich um eine Zeichenfolge, die um Ergebnisse gesetzt werden soll, um Sonderzeichen innerhalb der Daten nutzen zu können (z.B. Komma in Fließkommazahlen). Normalerweise ist diese Zeichenfolge das Anführungszeichen ″ Wenn der Qualifizierer selbst in den Daten enthalten ist, wird dieses Datenfeld einfach verdoppelt.

Im Folgenden Beispiel wird zunächst gezeigt, wie die Spaltenüberschriften in die csv-Datei exportiert werden.

```
Dim sWriter As StreamWriter

Qualifier = """"
strValue = ""
strRowValue = ""

For i As Integer = 0 To DataGridView.Columns.Count - 1
    strValue = DataGridView.Columns(i).Name
    strValue = strValue.Replace(Qualifier, Qualifier & Qualifier)

    strRowValue = strRowValue & Qualifier & strValue & Qualifier
    If i < (DataGridView.Columns.Count - 1) Then
        strRowValue = strRowValue & Delimiter
    End If
Next

sWriter.Write(strRowValue)
sWriter.WriteLine()
```

Auf der gleichen Art lässt sich der Inhalt des DataGridView-Steuerelements nach Excel exportieren.

Sollte beim Zugriff auf die Excel-Datei einen Fehler auftreten, weil die Datei zum Beispiel geöffnet oder schreibgeschützt ist, so wird eine Fehlerroutine ausgeführt.

```
Try
    sWriter = New StreamWriter(FullPath, False, enc)
Catch ex As Exception
    MsgBox("The process can not access the file, because it is " & _
           "being used by another process")
    End
End Try
```

Ist kein Fehler aufgetreten, so werden die Daten wie oben beschrieben in die Excel-Datei geschrieben. Der Dateiname setzt sich wie folgt zusammen:

```
strFile = "EventView" & _
          Format(Me.dtpFromEvent.Value, "yyyyMMddHHmmss") & "_" & _
          Format(Me.dtpToEvent.Value, "yyyyMMddHHmmss") & ".csv"
```

Zum Beispiel: „EventView20100603120000_20100703223000“. Das heißt, die Datei enthält Daten mit einem Zeitstempel von „03. Juni 2010 12:00:00“ bis „03. Juli 2010 22:30:00“.

Die folgende Abbildung zeigt einen Ausschnitt einer Excel-Datei mit Event-Daten.

	A	B	C	D	E
1	Datum	Bereich	Ereignis	Meldetext	Operator
2	03.11.2010 12:02:33	Mix	5T4.1LL01	Simulation: Aus	Administrator
3	03.11.2010 12:17:57	Mix	5T4.1LH02	Simulation: Aus	Administrator
4	03.11.2010 12:58:55	Mix	5T4.1LH02	Simulation: Aus	Administrator
5	03.11.2010 13:06:35	Pulver	5T4.V51_3	Error	Administrator
6	03.11.2010 12:45:48	Mix	PW4LL01	Simulation: Ein	Administrator
7	03.11.2010 12:01:52	Produktion	P113 Lecthin	Parameter: Container MDR-Nr. : 0 --> 8108600	Administrator
8	03.11.2010 12:01:55	Produktion	P113 Lecthin	Start	Administrator
9	03.11.2010 12:02:04	Produktion	P113 Lecthin	Parameter: Container MDR-Nr. : 0 --> 8108600	Administrator
10	03.11.2010 12:03:11	Produktion	P113 Lecthin	Parameter: Chargennr. : 0 --> 1950	Administrator
11	03.11.2010 12:04:17	Produktion	P113 Lecthin	Parameter: Containerinhalt bei Start [kg] : 0 --> 622,0	Administrator

Abbildung 6.16 Ausschnitt einer Excel-Datei mit Event-Daten

7 Zusammenfassung und Ausblick

Ziel dieser Bachelorthesis war, ein übersichtliches Meldesystem zur Alarmaufzeichnung und –Auswertung zu entwickeln, welches leicht zu konfigurieren ist.
Durch die erläuterte Softwareentwicklung wurde gezeigt, die das Gesamtsystem in einfache überschaubare Bausteine unterteilt wurde, wo jeder Baustein eine Teilfunktion des gesamten Prozesses erfüllt hat. Dabei handelt es sich um die Softwarekomponenten: Konfiguration, Datenerfassung und Datenanzeige.
An dieser Stelle sind die Anforderungen der vorliegenden Arbeit erreicht worden. Einige Verbesserungsvorschläge sollen hier kurz erläutert werden:

- Da aus zeitlichen Gründen ausreichende Tests nicht möglich waren, sollte das Programm weitgehend getestet werden, um so den Nutzen der Software bei der GEA TDS GmbH sicher zu stellen.
- Es sollte noch eine Option konfigurierbar sein. Hierbei handelt es sich darum, wie lange die Meldungen in der neuen Datenbank archiviert werden müssen. Dadurch wird gewährleistet, dass nicht zu alte Meldungen gespeichert werden und die Datenbank zu groß wird. Sondern es werden nur die Meldungen beibehalten, die sich in dem letzten konfigurierten Zeitabschnitt befinden.
- Um die angezeigten Daten ausdrucken zu können, sollte ein Report mit dem SQL Server Reporting Services erstellt werden. Zusätzlich zu zahlreichen anderen Eigenschaften stellt er eine Druck- und Druckvorschau-Funktion zu Verfügung.
- Um die Software weltweit einsetzen zu können, sollte eine Sprachumschaltung realisiert werden.

8 Schlussbemerkung

An dieser Stelle möchte ich mich ganz herzlich bei allen bedanken, die mich in der Bachelorarbeitszeit unterstützt haben. Als erstes sehr großen Dank an meine Eltern, die mir dieses Studium überhaupt ermöglicht haben. Ein großes Dankeschön habe ich auch an meine Schwester und meine Freunde zu richten, die meine Launen und meinen Zeitmangel während des Studiums ertragen mussten und immer vollstes Verständnis dafür hatten. Als Letztes gilt mein Dank meinen Arbeitskollegen aus der Automatisierungsabteilung bei Tuchenhagen Dairy Systems GmbH, die mir bei Verständnisproblemen immer mit Rat und Tat zur Seite standen.

Literaturverzeichnis

Monadjemi MONADJEMI, Peter: *Jetzt lerne ich Visual Basic .NET*. München: Markt+Technik Verlag, 2004

Rottmann ROTTMANN, Heinrich : *Visual Basic .NET mit Methode*. Wiesbaden: Friedr. Vieweg & Sohn Verlag, 2003

Doberenz/Gewinnus DOBERENZ, Walter und GEWINNUS, Thomas: *Datenbankprogrammierung mit Visual Basic 2005*. Unterschleißheim: Microsoft Press Deutschland, 2006

Kühnel/Leibbrandt KÜHNEL, Andreas und LEIBBRANDT, Stephan : *Visual Basic 2008*. Bonn: Galileo Press, 2009

Stähr STÄHR, Mathias: *Entwicklung eines graphischen Frontends mit Visual Basic 6 zur Chargenverfolgung in der Lebensmittelindustrie*. Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit 2005

Grosche GROSCHE, Igor: *Konzeption und prototypische Realisierung einer Komponente zur dynamischen Verwaltung von Assoziationen zwischen Business Components*. Technische Universität München, Diplomarbeit 2002

Favre_Bulle FAVRE-BULLE, Bernard : *Automatisierung komplexer Industrieprozesse*. Wien: Springer Verlag, 2004

Wikipedia WIKIPEDIA FREIE ENZYKLOPÄDIE
http://de.wikipedia.org/wiki/ADO.NET
Abruf am 28.12.2010
http://de.wikipedia.org/wiki/Xml
Abruf am 01.01.2011
http://de.wikipedia.org/wiki/CSV_%28Dateiformat%29
Abruf am 04.01.2011

Selfhtml http://de.selfhtml.org/xml/regeln/baumstruktur.htm
Abruf 5.12.2010

Jähnichen *Vorgehensmodell für komponentenbasierte Softwareentwicklung*
http://www.google.de/url?sa=t&source=web&cd=1&ved=0CBgQFjA
A& url=http%3A%2F%2Fwww.swt.tu- berlin.de%2Ffileadmin%2Ffg130
%2Flehre%2FSeminarSS09%2FSenyuava_Cennet_CBSE.pdf&rct=j&q
=Zwar%20hat%20die%20objektorientierte%20Programmierung%20die%
20Programmierung%20ver%C3%A4ndert%2C%20 dennoch%20wurde%
20die%20Softwareentwicklung%20nur%20zum%20Teil%20 erleichtert&ei=
wMEuTeHnHoqRjAfn8aiDBQ&usg=AFQjCNEHi3g-jGu2exS8avcS-
6IDEgVMww&cad=rja

Siemens Support *Wie können in WinCC ausgelagerte Archivsegmente von Alarm-Logging oder Tag-Logging in Runtime verbunden oder getrennt werden?*
http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.
csinfo&objId=40347325&objAction=csOpen&nodeid0=10805588&
lang=de&siteid=cseus&aktprim=0&extranet=standard&viewreg=WW
Abruf am 14.12.2010

CadFamily *SIMATIC HMI Systemuebersicht*
www.cadfamily.com/Download.asp?ID=230436
Abruf am 20.12.2010

Msdn *XML-Dateien lesen und schreiben mit VB.NET*
http://msdn.microsoft.com/de-de/library/bb979298.aspx
Abruf am 20.12.2010

SQL Commands *SQL Commands*
http://www.sqlcommands.net/
Abruf am 05.10.2010

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

Liste der Abkürzungen

ADO	ActiveX Data Objects
API	Application Programming Interface
UTC	Universal Time Coordinated
SQL	Structured Query Language
DB	Database
CSV	Comma Separated Values
XML	eXtensible Markup Language
OLE	Object Linking and Embedding

Inhalt der CD-ROM

Auf der CD befinden sich ergänzende Daten zur Bachelorthesis. Dies sind neben dem Programmquellcode die benutzten und im Literaturverzeichnis angegebenen Internetseiten.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

……

Ort, Datum Unterschrift