

DIPLOMARBEIT

Entwicklung eines Software-Systems

für die Qualitätssicherung von Oberflächen
im Automobilbau

*Datenbankaufbau und Entwicklung einer Datenbankschnittstelle
für die C++ Anwendung unter Verwendung
der ADO-Technologie*

Christos Agiovlassis

Konstanz, 31. Mai 2003

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

an der

Fachhochschule Konstanz
Hochschule für Technik, Wirtschaft und Gestaltung

Fachbereich
Informatik/Wirtschaftsinformatik

- Thema** : Entwicklung eines Software-Systems zur Qualitätssicherung von Oberflächen im Automobilbau: Datenbankaufbau und Entwicklung einer Datenbankschnittstelle für die C++ Applikation unter Verwendung der ADO-Technologie
- Diplomand** : Christos Agiovlassis, Heuriedweg 57, 88131 Lindau
- Betreuer** : Professor Dr. Wolfgang Arndt, Heiko Borho
- Firma** : VW/Audi do Brasil (Business Unit Curitiba)
- Eingereicht** : Konstanz, 31. Mai 2003

“ Wissen ist nur ein Teil des Verstehens.
Wirkliches Verstehen kommt erst mit praktischer Erfahrung,
(Dr. Seymour Papert)

Abstract

Heutzutage nimmt die Qualitätssicherung der Produkte bei immer mehr Betrieben einen zunehmend hohen Stellenwert ein.

Auch in der Automobilindustrie gelten mittlerweile hohe Qualitätsrichtlinien. Damit der Qualitätsstandard aber auch gewährleistet werden kann, muss die Qualität des Produkts ständig gemessen und beurteilt werden.

Um die manuelle Prüfung der Pressteile zu unterstützen wurde in den letzten Jahren bei VW/Audi do Brasil – BUC das Oberflächen-Inspektions-System DSight eingesetzt. Leider haben jedoch zahlreiche Messungen mit DSight ergeben, dass die Ergebnisse unzuverlässig und nicht reproduzierbar sind.

Aus diesem Grund wurde im Rahmen dieser Diplomarbeit ein neues Software-System zur Inspektion von Oberflächen im Automobilbau entwickelt, das die Anforderungen der Qualitätssicherung besser erfüllt.

Das neue System, das den Namen VisionMaster trägt, basiert auf dem Prinzip der Retro-Reflexion. Mit Hilfe implementierter Auswertelgorithmen werden aufgenommene Grauwertbilder der gepressten Teile analysiert. Die Ergebnisse der Inspektion werden anschließend in übersichtlicher Form in einem Protokoll ausgegeben. Um die Reproduzierbarkeit der Messungen zu garantieren und die Messergebnisse archivieren zu können wird zusätzlich eine Datenbank eingesetzt, auf die über die ADO-Technologie zugegriffen wird.

Zur Beschleunigung der Inspektion wird das neue System VisionMaster direkt in die Produktionshalle verlagert, um dort vor Ort Messungen an den gepressten Teilen durchführen zu können.

Inhaltsverzeichnis

	Seite
Abkürzungsverzeichnis	V
Darstellungsverzeichnis.....	VI
1 Einleitung	1
2 Das Volkswagen/Audi Werk.....	2
3 Qualitätssicherung im Automobilbereich.....	4
3.1 <i>Qualitätsstandards im Auto-mobilbereich.....</i>	<i>4</i>
3.2 <i>Oberflächen-Inspektions-Systeme im Einsatz.....</i>	<i>6</i>
3.2.1 ABIS – Automatic Body Inspection System	6
3.2.2 CEM - Chuo Electronic Measurement	11
3.2.3 ISRA – Car Paint Vision System.....	13
3.2.4 DSight – Surface Inspection System from LMI	15
4 Datenbanksysteme	20
4.1 <i>Datenbankaufbau.....</i>	<i>21</i>
4.2 <i>Modellierungskonzepte</i>	<i>22</i>
4.3 <i>Datenmodelle.....</i>	<i>23</i>
4.3.1 Das Hierarchische Datenmodell	24
4.3.2 Das Netzwerkmodell	25
4.3.3 Das Relationale Datenmodell	25
4.3.4 Das Objektorientierte Datenmodell	27
4.4 <i>Relationale Entwurfstheorie</i>	<i>28</i>
4.4.1 Funktionale Abhängigkeiten.....	28
4.4.2 Schlüssel	29
4.4.3 Bestimmung einer funktionalen Abhängigkeit.....	29
4.4.4 Normalisierung	31
4.4.5 Die dritte Normalform	33
4.5 <i>SQL.....</i>	<i>34</i>
4.6 <i>Relationale Datenbank APIs.....</i>	<i>36</i>
4.6.1 Überblick über Datenbank Client- Technologien.....	36
4.6.2 Vergleich der verschiedenen Datenbank Technologien	42
4.6.3 Microsoft Universal Data Access	43

4.7	<i>Datenbankapplikationen</i>	44
4.7.1	Oracle.....	44
4.7.2	Microsoft Access	44
5	Das System VisionMaster	46
5.1	<i>Systemüberblick</i>	46
5.2	<i>Hardwarekomponenten</i>	49
5.2.1	Der VisionMaster Rechner	49
5.2.2	Frame Grabber PX510.....	50
5.2.3	Kamera TVA 2100	52
5.2.4	Der Highlighter.....	54
5.2.5	Der Messtisch	56
5.3	<i>Softwarestruktur</i>	58
5.3.1	Grobarchitektur des Systems	58
5.3.2	Feinarchitektur.....	59
5.3.3	Das Modul DbInterface	69
5.3.4	Datenbankstruktur.....	71
5.4	<i>Erzeugen von Datenbank-Backups</i>	80
6	Optimierungsvorschläge	82
	Schlusswort.....	87
	Anhang – Die CADO Klassen.....	88
	Literaturverzeichnis	112

Vorwort

Immer mehr Unternehmen setzen in den letzten Jahren bei der Qualitätssicherung ihrer Produkte zusätzlich maschinelle Unterstützung ein.

Seit einigen Jahren wird im Unternehmen VW/AUDI BUC zur Unterstützung der Qualitätssicherung bei der Inspektion von Oberflächen im Presswerk das System DSight von LMI eingesetzt und erprobt. Leider haben eine Reihe von Messungen mit diesem System gezeigt, dass für die Auswertung von Grauwertbildern von den zur Verfügung stehenden Verfahren nur zwei Auswerteverfahren einsetzbar sind. Die anderen Auswerteverfahren haben bei kleinen Fehlern keine sinnvollen und reproduzierbaren Ergebnisse geliefert.

Um das System effektiv in der Produktion einsetzen zu können wurde über eine Erweiterung des vorhandenen Systems diskutiert. Im Rahmen einer Diplomarbeit sollte eine Erweiterung in erster Linie aussagekräftige und reproduzierbare Ergebnisse in einer relativ kurzen Zeit garantieren können.

Zahlreiche Tests und Messungen mit DSight in der Startphase der Diplomarbeit ergaben jedoch, dass eine sinnvolle Erweiterung dieses Systems nicht möglich ist bzw. nicht von großem Nutzen wäre. Aus diesem Grund wurde die Idee der Erweiterung des vorhandenen Systems wieder fallen gelassen, und die Entwicklung eines neuen Software-Systems mit dem Namen VisionMaster vereinbart, das den Ansprüchen der Qualitätssicherung besser gerecht wird.

Das Projekt VisionMaster umfasst im Groben die zwei großen Themengebiete Bildverarbeitung und Datenbanksysteme und wurde wegen Umfang und Komplexität des Projektes auf zwei Diplomarbeiten aufgeteilt. Durch eine enge Zusammenarbeit und die parallele Entwicklung konnten am Ende die zwei Einzelkomponenten des Systems problemlos unter einer gemeinsamen Benutzeroberfläche zu einem Gesamtsystem zusammengefügt werden.

Die Einführung von VisionMaster hat das alte System DSight vollständig ersetzt und wird im Moment bei VW/Audi BUC zur Oberflächeninspektion von Pressteilen erprobt.

Die vorliegende schriftliche Ausarbeitung der Diplomarbeit befasst sich schwerpunktmäßig mit dem zweiten Teil der Systementwicklung, der sich auf den Aufbau der Systemdatenbank und der Entwicklung der dazugehörigen Datenbankschnittstelle für die Anwendung konzentriert. Sie soll einen allgemeinen Überblick über Datenbanksysteme geben und Hintergründe bei der Entwicklung einer Datenbankschnittstelle erläutern, die als Bindeglied zwischen Anwendung und Datenbank agiert.

Christos Agiovlasis
Konstanz, 31. Mai 2003

Abkürzungsverzeichnis

ADO	:	ActiveX Data Objects
ANSI	:	American National Standards Institute
API	:	Application Programming Interface
CCD	:	Charge Coupled Device
COM	:	Component Object Model
DAO	:	Data Access Objects
DBMS	:	Database Management System
DDL	:	Data Definition Language
DML	:	Data Manipulation Language
ER	:	Entity-Relationship
funkt.	:	funktional(e)
ggf.	:	gegebenenfalls
i.d.R.	:	in der Regel
i. W.	:	im Weiteren
MFC	:	Microsoft Foundation Classes
MS	:	Microsoft
ODBC	:	Open Database Connectivity
OLE DB	:	Object Linking and Embedding Database
OODB	:	Objektorientiertes Datenbankmanagementsystem
RDBMS	:	Relationales Datenbankmanagementsystem
RDO	:	Remote Data Objects
SQL	:	Structured Query Language
s. S.	:	siehe Seite

Darstellungsverzeichnis

Abbildungen:

	Seite	
Abbildung 2.1	Business Unit Curitiba (BUC) in São José dos Pinhais	2
Abbildung 2.2	Das Presswerk	3
Abbildung 2.3	Das Kommunikationszentrum	3
Abbildung 3.1	VDA Richtlinien	5
Abbildung 3.2	Das ABIS-System	7
Abbildung 3.3	Das ABIS-System - Verwendung von spezifizierten Masken	8
Abbildung 3.4	Das ABIS-System – Anordnung der Videosensoren	8
Abbildung 3.5	Das ABIS System - Einzelbildaufnahme	9
Abbildung 3.6	Das ABIS-System-Entscheidungen bei der Fehlerklassifizierung	9
Abbildung 3.7	Das ABIS-System – Math. Verfahren bei der Fehlererkennung	10
Abbildung 3.8	Das CEM-System - Paint Defect Detecting Machine	11
Abbildung 3.9	Das CEM-System - Das Streifentriangulationsprinzip	11
Abbildung 3.10	Das CEM-System - Aufbau, Sensor und Kamerabild	12
Abbildung 3.11	Das CEM-System – Protokoll	12
Abbildung 3.12	Das ISRA-System - Gesamtansicht des Systems	13
Abbildung 3.13	Das ISRA-System - Erkennbare Fehler	13
Abbildung 3.14	Das ISRA-System - Aufgenommene Grauwertbilder	14
Abbildung 3.15	Das ISRA-System - Datenfluss des Systems	14
Abbildung 3.16	Das DSight-System – Komponenten	15
Abbildung 3.17	Das DSight-System - Positionierung der Komponenten	16
Abbildung 3.18	Das DSight System - Typischer Arbeitsplatz	17
Abbildung 3.19	Das DSight-System - Prinzip der Bildgewinnung	17
Abbildung 3.20	Das DSight-System - Oberfläche eines inspizierten Objektes	18
Abbildung 3.21	Das DSight-System – Signaturen von Defekten	18
Abbildung 3.22	Das DSight-System – Auswirkung des Blickwinkels	19
Abbildung 4.1	Abstraktionsschema eines Datenbanksystems	21
Abbildung 4.2	Beispiel eines ER-Diagramms	22
Abbildung 4.3	Schema eines hierarchischen Datenmodells	24
Abbildung 4.4	Schema eines Netzwerkmodells	25
Abbildung 4.5	Tabellen eines relationalen Datenmodells	26
Abbildung 4.6	Begriffsbildung und Tabellendarstellung im Relationenmodell	26
Abbildung 4.7	Tabellen in erster Normalform	32
Abbildung 4.8	Tabelle in zweiter Normalform	33
Abbildung 4.9	Format eines SQL-Anweisungsblocks	34
Abbildung 4.10	ODBC Architektur	37
Abbildung 4.11	DAO Architektur	38
Abbildung 4.12	OLE DB Consumer und Provider	39
Abbildung 4.13	OLE DB Hauptobjekte	39
Abbildung 4.14	ADO Objekt-Hierarchie	40
Abbildung 4.15	Abhängigkeit der verschiedenen Datenbank Technologien	42
Abbildung 4.16	Universal Data Access Architektur	43
Abbildung 4.17	Schemadefinition in Microsoft Access	45

Abbildung 5.1	Systemkomponenten von VisionMaster	46
Abbildung 5.2	Analyseprinzip von VisionMaster	47
Abbildung 5.3	Das Prinzip des Auswertungsalgorithmus PLOT	47
Abbildung 5.4	Prinzip der Retro-Reflexion	48
Abbildung 5.5	Frame Grabber PX510	51
Abbildung 5.6	Blockschaltbild der Frame Grabber Karte	51
Abbildung 5.7	Berechnung der Kamerahöhe	53
Abbildung 5.8	zu wenig Highlighter	53
Abbildung 5.9	zu viel Highlighter	54
Abbildung 5.10	korrekte Auftragung der Glanzflüssigkeit	54
Abbildung 5.11	Teile und Abmessungen des Messtisches	56
Abbildung 5.12	Positionierungsgitter für Messtisch und Kamera	57
Abbildung 5.13	Grobarchitektur des Systems VisionMaster	58
Abbildung 5.14	Feinarchitektur des Systems VisionMaster	59
Abbildung 5.15	Die Oberfläche des Hauptprogramms	60
Abbildung 5.16	Der Dialog AddNewModelView	60
Abbildung 5.17	Der Dialog AddNewPartView	61
Abbildung 5.18	Der Dialog AddNewRegionView	61
Abbildung 5.19	Der Dialog CamCapView	62
Abbildung 5.20	Der Dialog DBInvestigation	63
Abbildung 5.21	Der Dialog DeleteModelView	63
Abbildung 5.22	Der Dialog DeletePartView	64
Abbildung 5.23	Der Dialog DeletePlotView	64
Abbildung 5.24	Der Dialog DeleteRegionView	65
Abbildung 5.25	Der Dialog ModelChooserView	65
Abbildung 5.26	Der Dialog ModifyModelView	66
Abbildung 5.27	Der Dialog ModifyPartView	66
Abbildung 5.28	Der Dialog ModifyRegionView	67
Abbildung 5.29	Der Dialog PlotView	67
Abbildung 5.30	Der Dialog ProductionDataView	68
Abbildung 5.31	Die Protokollansicht	68
Abbildung 5.32	Hauptkomponenten des Moduls DbInterface	69
Abbildung 5.33	Mapping einer Datenbanktabelle im Programm	70
Abbildung 5.34	ER-Modell des Systems VisionMaster	71
Abbildung 5.35	VisionMaster Beziehungsdiagramm	78
Abbildung 5.36	Verzeichnisstruktur von VisionMaster	79
Abbildung 5.37	Erstellung und Importierung eines Datenbank-Backups	80
Abbildung 5.38	Der Speichern unter Dialog der DB-Backup Funktion	81
Abbildung 5.39	Der Öffnen Dialog der DB-Backup Funktion	81
Abbildung 6.1	Eingabemaske für die Presseparameter	84
Abbildung 6.2	Schema einer Mustererkennung und Klassifikation	85
Abbildung 6.3	lineare Klassifikation mit überlappenden Klassen	86
Abbildung 6.4	Klassifikation mit dem Bayes-Klassifizierer	86

Tabellen:

		Seite
Tabelle 4.1	Definition von Entity-Typen bei einem OODB	27
Tabelle 4.2	Die wichtigsten SQL-Statements	35
Tabelle 4.3	Stärken/Schwächen der Datenbank Technologien	42
Tabelle 5.1	Die wichtigsten Aufgaben der VisionMaster-Software	48
Tabelle 5.2	Spezifikation der Frame Grabber Karte PX510	50
Tabelle 5.3	Spezifikation der Kamera	52
Tabelle 5.4	Struktur der Tabelle „CrossbarSettings“ in der Datenbank	72
Tabelle 5.5	Struktur der Tabelle „CylinderSettings“ in der Datenbank	72
Tabelle 5.6	Struktur der Tabelle „Cylinders“ in der Datenbank	73
Tabelle 5.7	Struktur der Tabelle „InspectionInfo“ in der Datenbank	73
Tabelle 5.8	Struktur der Tabelle „Models“ in der Datenbank	73
Tabelle 5.9	Struktur der Tabelle „Parts“ in der Datenbank	74
Tabelle 5.10	Struktur der Tabelle „partOf“ in der Datenbank	74
Tabelle 5.11	Struktur der Tabelle „ProgramSettings“ in der Datenbank	74
Tabelle 5.12	Struktur der Tabelle „Plot“ in der Datenbank	75
Tabelle 5.13	Struktur der Tabelle „ProgramSettings“ in der Datenbank	75
Tabelle 5.14	Struktur der Tabelle „ProductionData“ in der Datenbank	76
Tabelle 5.15	Struktur der Tabelle „Stations“ in der Datenbank	76
Tabelle 5.16	Struktur der Tabelle „SystemSettings“ in der Datenbank	77
Tabelle 6.1	Anforderungen an den motor- gesteuerten Messtisch	83
Tabelle 6.2	Mögliche Klassenbildung für Pressteile	85

1 Einleitung

Bei der Beurteilung der Güte eines Inspektionssystems ist neben der Genauigkeit, Verlässlichkeit und Reproduzierbarkeit von Messungen, die Messdauer ein weiterer Faktor, dessen Berücksichtigung eine wichtige Rolle spielt.

Der Einsatz von DSight brachte einige Nachteile mit sich, aus denen die Notwendigkeit einer Neuentwicklung des Systems resultierte:

Einerseits ist die Hardware des alten Systems sehr veraltet (veraltete amerikanische Industrienorm) und eignet sich nicht optimal zum Zwecke der Bildverarbeitung bzw. zur Entwicklung von Software, zum anderen ist das System nur unter dem Betriebssystem Windows 95 lauffähig.

Sämtliche Auswertalgorithmen, die das System verwendet liefern mit Ausnahme des Algorithmus Plot keine nützlichen und reproduzierbaren Ergebnisse.

Außerdem unterstützt das System keine Datenbankanbindung, die für die Messdatenarchivierung und Protokollerstellung nötig wäre.

Zwar bietet DSight eine Makro-Sprache an, mit der bestimmte Messabläufe automatisiert werden können, aber eine Weiterentwicklung bzw. Erweiterung der Software ist nicht möglich.

Diese und weitere Schwachpunkte des DSight-Systems wurden bei der Entwicklung des neuen Systems VisionMaster berücksichtigt und umgesetzt.

Die Hauptziele von VisionMaster sind neben der Verkürzung der Inspektionsdauer, die Bereitstellung verlässlicher Auswertalgorithmen, die Reproduzierbarkeit der Messungen sowie eine einfache und benutzerfreundliche Bedienung des Systems.

2 Das Volkswagen/Audi Werk

Der Volkswagenkonzern ist der größte Automobilhersteller der brasilianischen Automobilindustrie und der einzige Konzern der Südhalbkugel, der fähig ist, Autos zu produzieren und zu entwickeln, die Akzeptanz in der ganzen Welt finden.

Unter den vielen Werken, die Volkswagen in Brasilien unterhält, ist das Werk in São José dos Pinhais eins der modernsten Werke der Volkswagengruppe und der Welt. Durch Investitionen in Höhe von 800 Millionen US-Dollar produziert das Werk ca. 450 Autos am Tag (Golf, Audi A3 und Saveiro) und beschäftigt 2600 Arbeitnehmer. Des Weiteren wurden seit der Einweihung des Standortes am 18. Januar 1999 mehr als 10 000 Arbeitsplätze durch Drittfirmer geschaffen.

Die Kapazität beläuft sich auf 700 Autos am Tag, was einer Jahresproduktion von ca. 200 000 Autos entspricht.



Abbildung 2.1 – Business Unit Curitiba (BUC) in São José dos Pinhais

Der Komplex überzeugt durch sein prägnantes Erscheinungsbild. Das Gebäude (überbaute Fläche von 210.000 m²) wurde sternförmig angelegt. In seinen Flügeln beherbergt es die Fertigungsbereiche des Präzisions-Karosseriebaus, der Lackiererei sowie der Fahrzeugmontage.

Das Herzstück des Werks ist das Kommunikationszentrum. Dort laufen alle planenden, steuernden sowie qualitätssichernden Funktionen zusammen.



Abbildung 2.2 – Das Presswerk

Das Herzstück des Presswerks ist eines der modernsten Pressen der Firma Schuler AG. Hier werden u.a. die Autoteile für den Golf gepresst.

Investition 121 Mio. R\$

Das Werk sticht nicht nur wegen der Benutzung von fortgeschrittener Technologie, wie das Laserschweißen und der Verwendung von Lackstoffen auf Wasserbasis heraus, sondern auch durch die Realisierung eines Industrieparks, in dem 14 der Zulieferer untergebracht wurden.

**Das Kommunikationszentrum:
Hier laufen sämtliche Fäden zusammen; nicht nur auf dem Papier, sondern auch in der Form des Komplexes.**



Abbildung 2.3 – Das Kommunikationszentrum

Volkswagen/Audi ist der einzige brasilianische Automobilhersteller, der den wettbewerbsfähigen nordamerikanischen Markt mit einer großen Bandbreite beliefert. Im Jahre 2001 wurden 43700 Golfs in die Vereinigten Staaten und Kanada exportiert. Letztes Jahr stellte das Werk ca. 97000 Fahrzeuge her, von denen 53900 für den externen Markt bestimmt waren.

Volkswagen/Audi pflegt die engen Kontakte zu den umliegenden Universitäten und Hochschulen, so dass unter den gesellschaftlichen Anlagen des Werkes in São José dos Pinhais, sich Abkommen mit 12 Universitäten und Hochschulen des Staates Paraná für die technische, wissenschaftliche und auszubildende Zusammenarbeit befinden.

Seit der Eröffnung des Werkes vor fünf Jahren hat Volkswagen/Audi verschiedene nationale und internationale Preise in unterschiedlichen Bereichen erhalten, darunter den Preis für den größten Exporteur des Staates Paraná, den Qualitätspreis Brasiliens und den Preis „Total Quality“ (für den Golf der in die Vereinigten Staaten exportiert wird).

3 Qualitätssicherung im Automobilbereich

Alle wichtigen Businessprozesse erhalten aufgrund starker Globalisierungsbestrebungen eine neue Bedeutung in den Unternehmensstrukturen. Früher waren Servicebereiche weitgehend lokal orientiert. Allein schon aus der Kosten- und Wettbewerbssituation heraus begründet, werden heute Infrastrukturen in Konzernen global definiert. Hieraus ergibt sich ein neues Verständnis über Servicegrade und Serviceleistungen. In der IT sind somit Themen wie Configuration Control, Change Management und Outsourcing mehr als nur Schlagworte. Folglich gewinnen auch Systeme wie Data Warehouse, Service Catalogue, Request Management und Helpdesk an zusätzlicher Bedeutung aufgrund neuer Anforderungen.

Neben diesen branchen-übergreifend erkennbaren Tendenzen gelten insbesondere im Automobilbereich hohe Qualitätsrichtlinien.

3.1 Qualitätsstandards im Automobilbereich

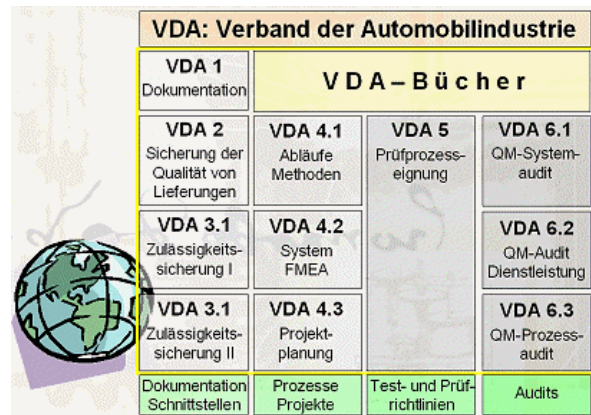
In der Automobilindustrie gelten hohe Qualitätsrichtlinien. Hierbei hat in Deutschland der VDA (Verband der Automobilindustrie) ein Qualitätssystem entwickelt, das neben den Herstellern auch die Automobilzulieferer und Dienstleistungsunternehmen im weitesten Sinne betrifft. Daneben fördert der VDA national und international die Interessen der gesamten deutschen Automobilindustrie auf allen Gebieten der Kraftverkehrswirtschaft (z.B. in der Wirtschafts-, Verkehrs- und Umweltpolitik), der technischen Gesetzgebung, der Normung und natürlich der Qualitätssicherung.

Die letzten Jahre zeigten, dass IT-Umgebungen dieser Unternehmen verstärkt im Fokus entsprechender Qualitäts-Audits stehen. Man begreift Informationssysteme zunehmend als strategische Komponenten in einer global organisierten Unternehmensstruktur.

Die VDA Richtlinien

Zur Sicherung der Qualität entstanden in der jüngeren Verbandsgeschichte entsprechende Richtlinien, die in den so genannten VDA-Büchern zusammengefasst sind. Hierbei lassen sich die Schwerpunkte Dokumentation und Schnittstellen, Prozesse und Projekte, Test- und Prüfrichtlinien sowie Audits erkennen.

Die Richtlinien des Verbandes der Automobilindustrie sind in so genannte VDA-Bücher festgehalten.



VDA: Verband der Automobilindustrie			
VDA 1 Dokumentation	VDA – Bücher		
VDA 2 Sicherung der Qualität von Lieferungen	VDA 4.1 Abläufe Methoden	VDA 5 Prüfprozess-eignung	VDA 6.1 QM-System-audit
VDA 3.1 Zulässigkeits-sicherung I	VDA 4.2 System FMEA		VDA 6.2 QM-Audit Dienstleistung
VDA 3.1 Zulässigkeits-sicherung II	VDA 4.3 Projekt-planung		VDA 6.3 QM-Prozess-audit
Dokumentation Schnittstellen	Prozesse Projekte	Test- und Prüf-richtlinien	Audits

Abbildung 3.1 – VDA Richtlinien

■ VDA 6.1 - QM-Systemaudit

Das Band enthält den Fragenkatalog zur Bewertung eines QM-Systems¹⁾ sowie das zugehörige Bewertungssystem. Die Anwendung ist in erster Linie in Unternehmen vorgesehen, die materielle Produkte herstellen. Der Inhalt des Fragenkataloges geht deutlich über die Forderungen der DIN EN ISO 9001 bzw. 9002 hinaus.

■ VDA 6.2 - QM-Systemaudit – Dienstleistung

Mangelnde Qualität bei Dienstleistungsunternehmen kann die größten Bemühungen der vorgelagerten Glieder einer Prozesskette unwirksam machen und deren auf Kundenzufriedenheit ausgerichtete Aktivitäten in Frage stellen. In diesem Band wird ein QM-System dargestellt, das speziell auf die Erfordernisse des Dienstleistungssektors in der Automobilindustrie ausgerichtet ist. Es bietet aber auch anderen Dienstleistungsbranchen Hinweise für die Ist-Analyse sowie den Aufbau und Auditierung ihres QM-Systems. Ebenfalls können Dienstleistungen, die im Rahmen der Herstellung eines materiellen Produkts erbracht werden (z.B. Marketing, Vertrieb, Kundens Schulung), nach diesem Fragenkatalog bewertet werden.

■ VDA 6.3 – Prozessaudit

Das Bindeglied zwischen dem System- und Produktaudit ist das Prozessaudit, das eine Aussage über die Fähigkeit von Prozessen bei der Planung und Herstellung von Produkten und der Dienstleistungserbringung macht. Prozessaudits dienen der Beurteilung und Qualitätsfähigkeit von Prozessen. Sie sollten zu fähigen beherrschten Prozessen führen, die robust gegenüber Störgrößen sind.

¹⁾ Qualitätsmanagement-System

3.2 Oberflächen-Inspektions-Systeme im Einsatz

Damit ein hoher Qualitätsstandard gewährleistet werden kann, müssen in der Praxis Systeme zum Einsatz kommen, die die Qualität des Produktes „messen“ und „bewerten“.

Am Ende der hochautomatisierten Karosserielackierung prüfen geübte Mitarbeiter die Oberfläche bestimmter Karosserie-segmente auf Lackfehler und beseitigen diese. Trotz ihrer zumeist langen Erfahrung und „geschulten Augen“ führen persönliche Toleranzgrenzen, Ermessensspielräume und die nicht immer gleiche menschliche „Tagesform“ dazu, dass einige Fehler nicht erkannt werden.

Die Folge daraus sind teure Nacharbeiten in einem späteren Stadium der Fertigung. Die Alternative zur anstrengenden und im Ergebnis nie optimalen Sichtprüfung durch Menschen sind Systeme zur Sicherung der Qualität von Oberflächen. Mit Hilfe mehrerer beweglich oder stationär arbeitender hochauflösender Kameras erkennen und orten diese Systeme u.a. Farbfehler, Staubeinschlüsse, Krater, Nadelstiche, Tropfen, Lackläufer, Kratzer und Orangenhaut.

Im Folgenden werden vier Systeme zur Oberflächenanalyse im Automobilbau vorgestellt, die heutzutage in verschiedenen Betrieben ihren Einsatz finden.

3.2.1 ABIS – Automatic Body Inspection System

Das ABIS-System wurde speziell für die Erkennung oberflächenrelevanter Fehler im Fahrzeugbau entwickelt. ABIS ermöglicht die automatische Überprüfung der gesamten Oberfläche innerhalb von 90 Sekunden. Es wurde in Zusammenarbeit mit der Audi AG Ingolstadt, Steinbichler Optotechnik Neubeuern, der TECHMATH/TECINNO GmbH, der Universität Erlangen-Nürnberg und der ITWM Kaiserslautern entwickelt.

Das integrierte Gesamtsystem besteht aus einem Messportal mit Positionierungseinrichtungen für mehrere optische Sensoren, einem schnellen netzwerk-gekoppelten Rechnerverbund zur Datenerfassung und anschließender asynchroner Auswertung sowie einer Fehlermarkieranlage.

Nachfolgende Abbildung zeigt das ABIS-System bestehend aus einem Identifikationssystem, einem Rechnerverbund, einem Messportal und einer Fehlermarkieranlage.

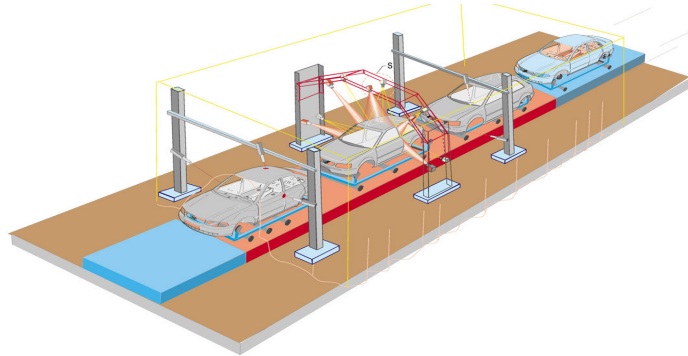


Abbildung 3.2 – Das ABIS-System

Die Systemsteuerung ermöglicht neben dem Automatikbetrieb der Anlage auch die manuelle Steuerung aller Einzelprozesse. Fehlerstellen können zusätzlich zur Markierung auf der Karosserie als Bilddaten oder Fehlerlisten ausgedruckt oder über das Netzwerk an Statistikprogramme weitergereicht werden.

Realisierung des ABIS-Systems

Die Arbeitsweise des Systems ist in der Abbildung 3.2 dargestellt. In einem ersten Schritt wird das einfahrende Auto automatisch erkannt. Je nach Fahrzeugtyp bewegen sich die Videosensoren zu vorgegebenen Positionen für die Bildaufnahme, während das Fahrzeug an ihnen vorbeifährt. Durch Demodulation werden die Bilddaten in Tiefenbilder umgerechnet, bei denen jeder Pixel den Abstand des entsprechenden Oberflächenpunktes zum Sensor beschreibt. Diese Bilder werden genauer analysiert und die Fehlerbereiche, sofern welche gefunden wurden, mit Hilfe eines CBR¹⁾-Systems klassifiziert.

Die letztendlich relevanten Fehlerstellen werden auf der Karosserie markiert, um anschließend eine Nachbearbeitung zu vereinfachen.

Im Folgenden werden die einzelnen Schritte im Detail erläutert.

■ Identifikation des Fahrzeugtyps:

Die Kamera und die Roboterpositionen werden individuell an das Fahrzeug, das in das Sensorenportal eintritt angepasst. Nur diejenigen Automodelle werden überprüft, die auch von dem System wieder erkannt werden. Dies ist aus diesem Grund wichtig, da jeder Bildposition eine individuelle Maske zugeordnet werden muss.

¹⁾ Case Based Reasoning

Diese Maske verdeckt stark gekrümmte Autoteile, wie z.B. eine Vertiefung des Türgriffes oder der Tanköffnung, die ansonsten störend auf den Algorithmus einwirken würden (siehe Abbildung 3.3).

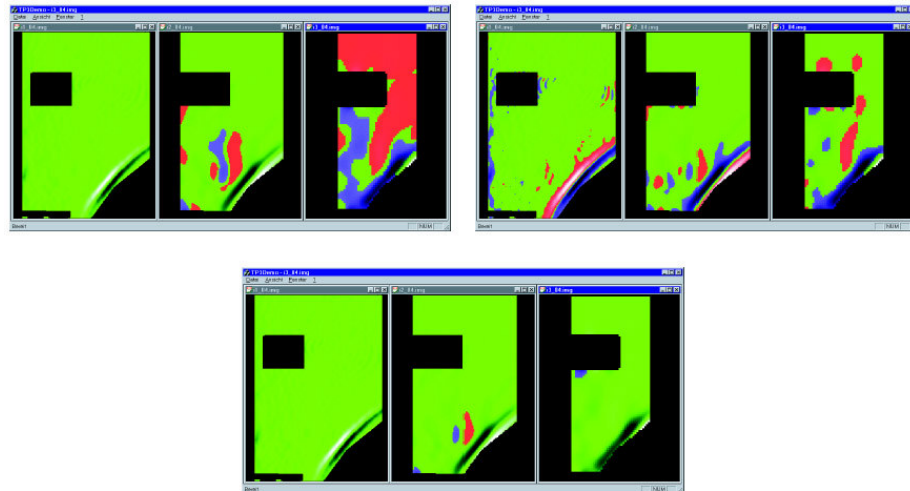


Abbildung 3.3 - Das ABIS-System - Verwendung von spezifizierten Masken

■ Datenerfassung:

20 bis 25 optische Sensoren tasten die Oberfläche des Fahrzeuges ab. Die Mängel, die von dem System erkannt werden sollen, sind nur einige Mikrometer groß, so dass eine sehr hohe Messgenauigkeit gegeben sein muss. Aus diesem Grund ist jeder Videosensor nur für eine Fläche der Größe 20-30 cm zuständig. Um das gesamte Fahrzeug zu erfassen, müssen pro Messung zwischen 400 und 600 Bilder aufgenommen werden.

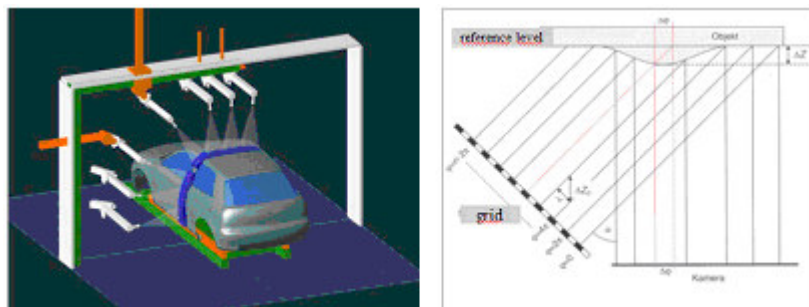


Abbildung 3.4 - Das ABIS-System – Anordnung der Videosensoren

■ Entdeckung von Fehlern:

Die riesige Datenmenge, die durch die Videosensoren aufgenommen wird, muss erst einmal reduziert werden. Dafür besitzt jeder einzelne Sensor eine Rechereinheit, die potentielle Fehler entdeckt.

Nur diejenigen Bilder, die potentielle Fehler enthalten, werden an die „Analyseeinheit“ weitergeleitet, um dann dort weiterverarbeitet zu werden.

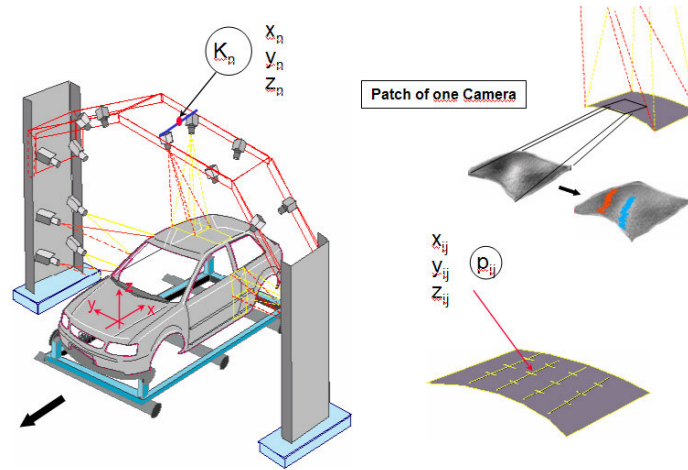


Abbildung 3.5 - Das ABIS-System - Einzelbildaufnahme

■ Defektanalyse und Klassifizierung:

Die „Analyseeinheit“ extrahiert zunächst die relevanten Daten aus den Bildern, die durch die Videosensoren aufgenommen wurden. Diese Merkmale werden mit Hilfe eines lernfähigen, CASE-basierten Algorithmus (CBR) ausgewertet. Das CBR-System entscheidet dann, ob die extrahierten Merkmale einen Fehler darstellen, und falls dies der Fall sein sollte, ob es ein relevanter Fehler ist (d.h. ein Fehler der Klasse **A** oder **B**).

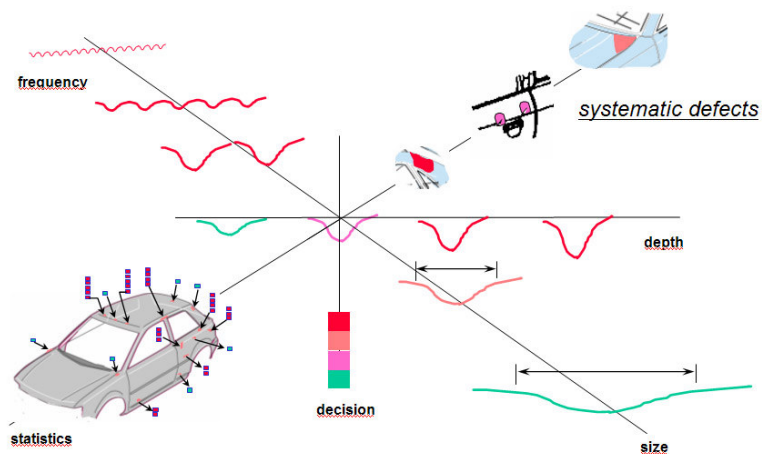


Abbildung 3.6 - Das ABIS-System-Entscheidungen bei der Fehlerklassifizierung

■ **Mathematische Verfahren:**

Aus den Tiefeninformationen wird zuerst mit Hilfe eines linearen Filters die Abweichung zu einer fehlerfreien Oberfläche geschätzt. Fehlerbereiche wie z.B. Dellen oder Beulen entsprechen im Gegensatz zu Rauschen einem relativ großen Wert in diesem Abweichungsbild. In einem zweiten Schritt werden mittels adaptiver Schwellwerte lokale Extrema bestimmt. Die auf diesem Wege erhaltenen Fehlerkandidaten werden anschließend mittels morphologischer Methoden geglättet. Letztlich werden für diese Fehlerkandidaten charakteristische Features für die Klassifikation bestimmt.

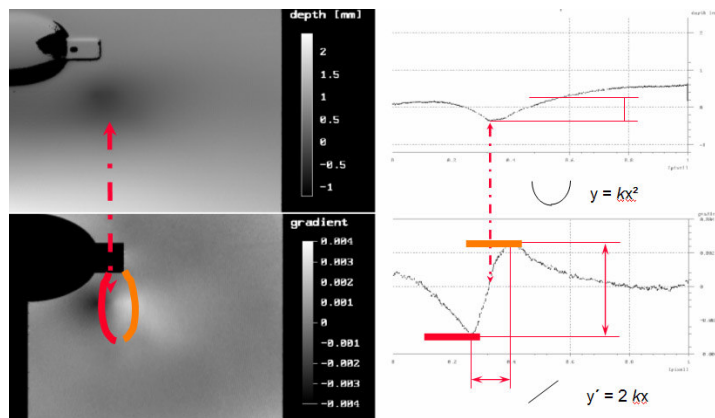


Abbildung 3.7 - Das ABIS-System – Math. Verfahren bei der Fehlererkennung

3.2.2 CEM - Chuo Electronic Measurement

Das System besteht aus einem Messportal mit ca. 40 optischen Sensoren, einem schnellen netzwerk-gekoppelten Rechnerverbund zur Datenerfassung und anschließender Auswertung sowie einer Fehlermarkieranlage. Der Aufbau des Systems ist somit beinahe identisch mit dem Aufbau des ABIS-Systems. Auch diese Anlage macht sich das Grundprinzip der Streifentriangulation zu nutze, indem „Streifenlicht“ auf das Fahrzeug projiziert wird, und das daraus resultierende reflektierte Licht durch optische Sensoren aufgefangen wird.

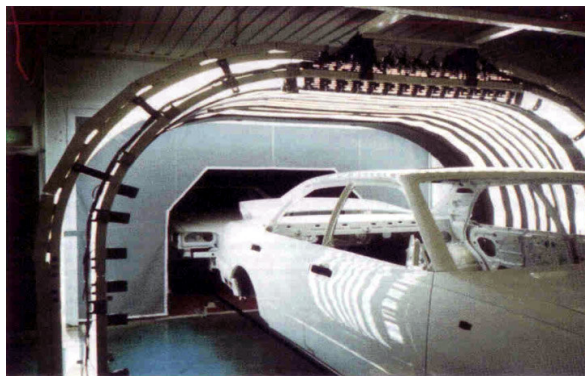


Abbildung 3.8 - Das CEM-System - Paint Defect Detecting Machine

Die Auswahl des Automodells geschieht anhand von 3D-Konstruktionszeichnungen. Dieses System ist jedoch nicht in der Lage verdeckte Partien des Fahrzeuges zu analysieren. Des Weiteren ist es nicht möglich, die Vorder- bzw. Rückseite eines Autos zu untersuchen, da die Sensoren des Systems fest positioniert sind.

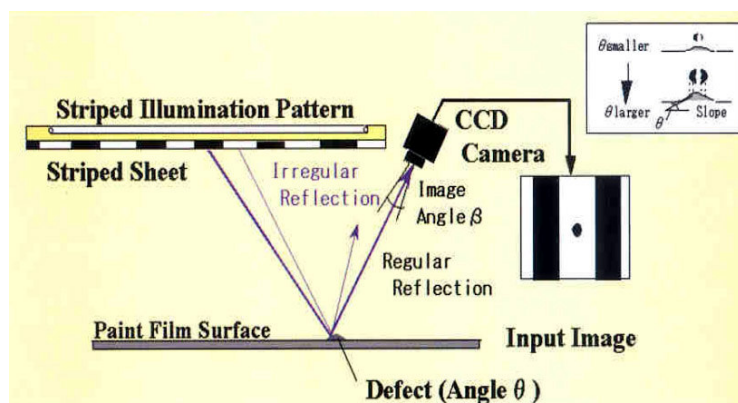


Abbildung 3.9 - Das CEM-System - Das Streifentriangulationsprinzip

Da Lackschäden normalerweise eine Größe von einigen Mikrometern haben, muss eine sehr hohe Messgenauigkeit gegeben sein. Daher nimmt jede Kamera nur eine Fläche der Größe 12 cm x 12 cm auf. Während sich das Fahrzeug in Bewegung befindet, nehmen die Videosensoren 20 Bilder pro Sekunde auf.

Nach Angaben von CEM findet das System beinahe 100% aller Defekte, die eine lackierte Oberfläche aufweist. Dabei ist die Art des Schadens irrelevant. Testreihen haben jedoch gezeigt, dass das System nur ca. 66% der Defekte ausfindig macht.

Die Fehler werden vom System erkannt und wie in der Automobilbranche üblich in Fehler der Klasse A (0.2 mm – 0.5 mm), Klasse B (0.5 mm – 1.0 mm), Klasse C (1.0 mm – 2.0 mm) oder Klasse D (2.0 mm – 5.0 mm) eingeteilt.

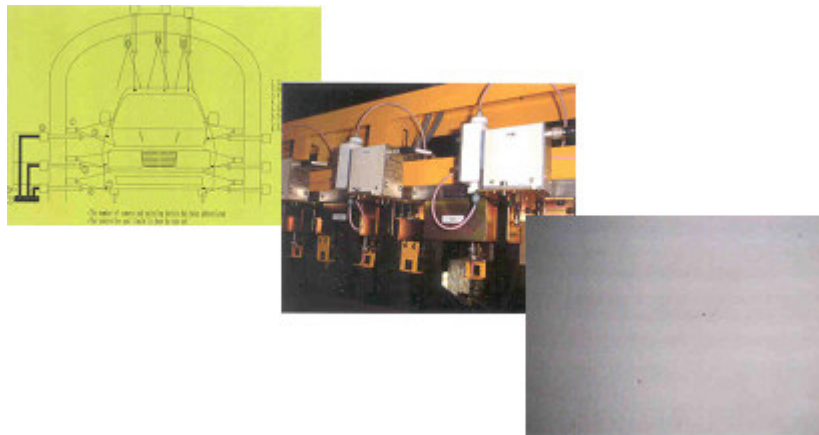


Abbildung 3.10 - Das CEM-System - Aufbau, Sensor und Kamerabild

Nach der Analyse eines Fahrzeuges liefert das System ein Protokoll, indem erkannte Defekte aufgezeichnet und klassifiziert sind. Außerdem werden auch diejenigen Stellen markiert, die von dem System nicht inspiziert wurden.

CHECKSHEET	TYPE	NUMBER	COLOR	OF DEFECT	PHOTO SENSOR		
DATE: 10/11/11 11:34	NAME: 1115	NO.:	TYPE: 111	色: 01	少ケテ: 10%	欠陥数: 1(1/10)	光電: 123
CLEAR							
1	1-1	111	111	0			
2	1-2	112	112	0			
3	1-3	113	113	0			
4	1-4	114	114	0			
5	1-5	115	115	0			
6	1-6	116	116	0			
7	1-7	117	117	0			
8	1-8	118	118	0			
9	1-9	119	119	0			
10	1-10	120	120	0			
11	1-11	121	121	0			
12	1-12	122	122	0			
13	1-13	123	123	0			
14	1-14	124	124	0			
15	1-15	125	125	0			
16	1-16	126	126	0			
17	1-17	127	127	0			
18	1-18	128	128	0			
19	1-19	129	129	0			
20	1-20	130	130	0			
21	1-21	131	131	0			
22	1-22	132	132	0			
23	1-23	133	133	0			
24	1-24	134	134	0			
25	1-25	135	135	0			
26	1-26	136	136	0			
27	1-27	137	137	0			
28	1-28	138	138	0			
29	1-29	139	139	0			
30	1-30	140	140	0			
31	1-31	141	141	0			
32	1-32	142	142	0			
33	1-33	143	143	0			
34	1-34	144	144	0			
35	1-35	145	145	0			
36	1-36	146	146	0			
37	1-37	147	147	0			
38	1-38	148	148	0			
39	1-39	149	149	0			
40	1-40	150	150	0			
41	1-41	151	151	0			
42	1-42	152	152	0			
43	1-43	153	153	0			
44	1-44	154	154	0			
45	1-45	155	155	0			
46	1-46	156	156	0			
47	1-47	157	157	0			
48	1-48	158	158	0			
49	1-49	159	159	0			
50	1-50	160	160	0			

Abbildung 3.11 - Das CEM-System – Protokoll

Zur Zeit wird diese Anlage bei BMW Dingolfing und bei Audi Ingolstadt zu Testzwecken eingesetzt.

3.2.3 ISRA – Car Paint Vision System

Das Car Paint Vision System von ISRA verwendet mehr als 50 Kameras um alle dominierenden Farbdefekte auf den lackierten Fahrzeugen zu erkennen und eine hohe Zuverlässigkeit zu garantieren. Die Koordinaten der Defekte werden an die Robotermarkierungs-Einheit weitergereicht, die mit Hilfe eines leicht zu entfernenden Sprays die Fehler auf der Fahrzeugoberfläche markiert.

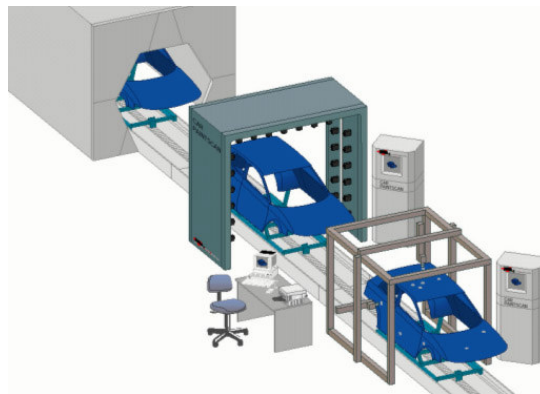


Abbildung 3.12 - Das ISRA-System - Gesamtansicht des Systems

Das System erkennt, wie auch die beiden vorherigen Systeme, Fehler mit einer Größe von einigen Mikrometern. Dies können sowohl Lackläufer, Staubeinschlüsse, Farbfehler, Kratzer oder Tropfen sein.

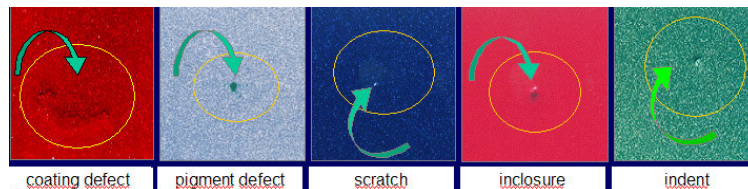


Abbildung 3.13 - Das ISRA-System - Erkennbare Fehler

Die Kameras werden individuell an den Fahrzeugtyp, der vom System inspiziert werden soll angepasst, indem aus einer Datenbank Positionsangaben zu vordefinierten, relevanten Bereichen ausgelesen werden. Nachdem die beweglichen optischen Sensoren eingestellt sind, werden mit den Kameras Bilder des Fahrzeugs aufgenommen, während sich dieses vorwärts bewegt. Die einzelnen Bilder ergeben zusammengefasst eine repräsentative Aussage über den Zustand der zu inspizierenden Oberfläche.

Der Unterschied zum ABIS-System besteht darin, dass das System die Farbe der Oberfläche kennen muss, damit die Auswertelgorithmen einwandfrei arbeiten können. Diese Informationen werden vor der Inspektion an das ISRA-System geliefert.

Die verwendeten Algorithmen arbeiten auf der Basis von Grauwertbildern¹⁾ und nicht anhand von Binärbildern²⁾.

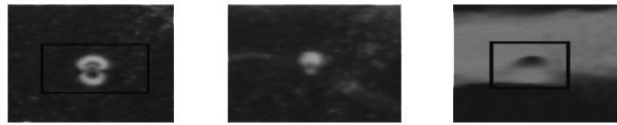


Abbildung 3.14 - Das ISRA-System - Aufgenommene Grauwertbilder

Wie auch beim ABIS-System, besitzt beim ISRA-System jeder Videosensor eine eigene Rechneinheit um die Datenmenge, die die Kameras liefern in Grenzen zu halten. Unterschiedlich ist jedoch der Verarbeitungsort der Daten.

Während beim ABIS-System die Bilder auf den einzelnen Rechneinheiten nur auf Fehler überprüft und dann an die „Analyseeinheit“ weitergeleitet werden, werden die Bilder beim ISRA-System vollständig auf den autonomen Rechneinheiten analysiert. Lediglich das Ergebnis dieser Analyse wird einer zentralen Recheneinheit zugesendet. Dieser zentrale Rechner zeigt die Ergebnisse der einzelnen Bildanalysen an und leitet die Positionierungsdaten an die Markierungseinheit weiter.

Anhand der aufgenommenen Bilder und der gespeicherten Referenzdaten ist es dem System möglich Oberflächendefekte zu erkennen und in einem weiteren Arbeitsschritt zu markieren.

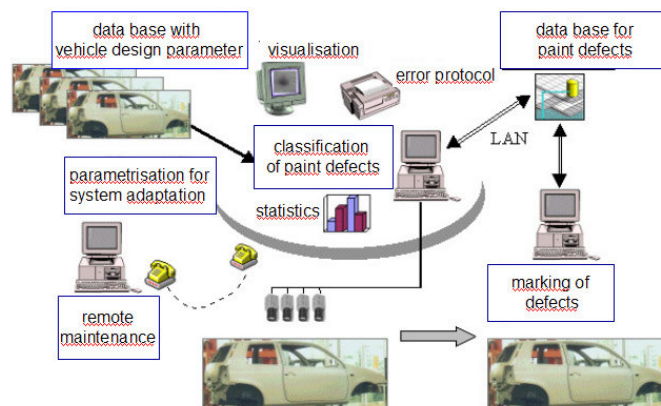


Abbildung 3.15 - Das ISRA-System - Datenfluss des Systems

Im Großen und Ganzen sind sich aber die beiden Systeme ABIS und ISRA sehr ähnlich. Beide Systeme weisen einen ähnlichen Systemaufbau auf und arbeiten mit ähnlichen Auswertalgorithmen. Leider gibt es aber keine Vergleichsstudie zwischen den beiden Inspektionssystemen.

1) Grauwertbilder sind Bilder, die aus 256 verschiedenen Graustufen aufgebaut sein können.

2) Binärbilder sind Bilder, die nur aus zwei Farben bestehen (z.B. Schwarz/Weiß).

3.2.4 DSight – Surface Inspection System from LMI

Im Gegensatz zu den drei vorher dargestellten Systemen basiert das DSight-System der Firma LMI auf einem anderen Prinzip der Bildaufnahme.

Während das ABIS-, das CEM- und das ISRA-System Inline-Systeme repräsentieren (d.h. in die Produktionslinie integriert werden können), ist DSight ein Offline-Inspektionssystem (die Fahrzeuge bzw. Fahrzeugteile müssen aus der Produktionslinie herausgenommen werden).

Dies stellt auch den Nachteil des Ganzen dar, denn wie in allen Branchen, so gilt auch in der Automobilindustrie „time is money“, und Inline-Systeme sind nun mal zeitsparender, da die Fahrzeuge nie zum Stillstand kommen und nicht aus der Produktionslinie herausgenommen werden müssen.

Andererseits ist das DSight-System so kostengünstig wie kein anderes. Es benötigt sehr wenig Raumfläche, da es nur eine Kamera und eine retro-reflektierende Wand zur Bildaufnahme verwendet.

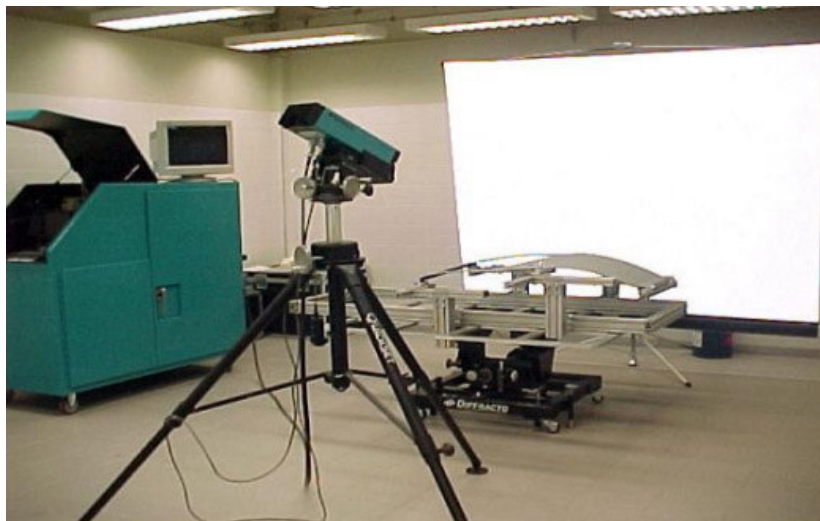


Abbildung 3.16 - Das DSight-System – Komponenten

Das LMI DSight-System wurde entwickelt, um Qualitätsansprüchen auf Oberflächen wie z.B. Metalle, Plastik und Glas gerecht zu werden. Es setzt sich aus der Kamera, der retro-reflektierenden Wand und dem Rechner zusammen.

Bevor jedoch mit dem System Inspektionen durchgeführt werden können, muss vorher eine Kalibrierung durchgeführt werden.

Die durch die Kamera aufgenommenen Bilder werden auf dem Rechner mit einer Vielzahl von Algorithmen analysiert (Surface, FREQB, Feature, Metall, OPEEL, LTW). Nach der Analyse liefert das System detaillierte Informationen zu den gefundenen Defekten.

■ **Die Sensoren des DSight Systems:**

Für das DSight-System bietet LMI drei verschiedene Ausführungen der Kamera an. Das Modell TVA 2000 enthält eine einzelne, punktförmige Lichtquelle, das Modell TVA 2100 beinhaltet mehrere punktförmige Lichtquellen und das Model TVA 3000 ist eine automatische Kamera.

Jede Kamera beinhaltet einen CCD-Chip mit einer Auflösung von 754 x 488 Bildpunkten und eine eingebaute Lichtquelle, die das zu analysierende Objekt ausleuchtet. Die Kameras werden auf einem speziellen Stativ ausgeliefert, das eine genaue Ausrichtung der Kamera in Abhängigkeit des zu inspizierenden Objektes ermöglicht.

■ **Die retro-reflektierende Wand:**

Die Größe der Oberfläche, die bei der Verwendung der Kamera von einer einzelnen Position aus inspiziert werden kann, hängt größtenteils von der Größe der retro-reflektierenden Wand ab.

Im allgemeinen gilt: je größer die Oberfläche die analysiert werden soll, desto größer die benötigte retro-reflektierende Wand. Das System wird mit einer 3,7 m breiten, gekrümmten Wand ausgeliefert. Obwohl es dem Beobachter nicht sofort offensichtlich ist, ist die retro-reflektierende Wand ein wesentlicher Bestandteil des optischen DSight-Systems. Aus diesem Grund sollte die Wand mit genau der gleichen Vorsicht behandelt werden wie die Linse einer Kamera.

■ **Das Rechnersystem:**

Der Computer kontrolliert sämtliche Funktionen der Audit Station mit Hilfe des DSight Softwarepaketes, das auf Windows 98 läuft. Die installierten Framegrabberkarten haben eine maximale Auflösung von 640 x 480 Bildpunkten.

■ **Aufbau des Systems:**

Die folgende Abbildung zeigt wie die einzelnen Komponenten des DSight-Systems positioniert werden müssen, damit eine Inspektion durchgeführt werden kann.

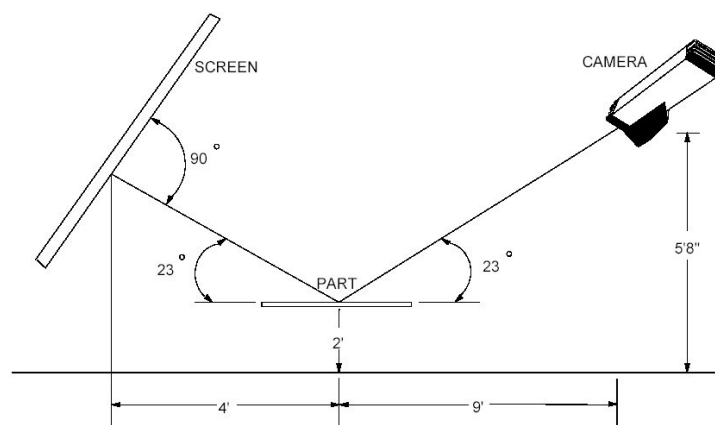


Abbildung 3.17 - Das DSight-System - Positionierung der Komponenten

Dieser Aufbau ermöglicht jedoch nur die Analyse von kleinen, flachen Teilen. Die Winkel, die in der Abbildung dargestellt werden, können in der Praxis um ± 2 Grad abweichen.

Das folgende Bild repräsentiert einen typischen DSight Inspektions-Arbeitsplatz. In der linken unteren Ecke des Bildes sehen wir ein Autoteil; in diesem Fall die Motorhaube, die auf dem Präzisionsmesstisch liegt. Der große „Gegenstand“, der hinter dem zu inspizierendem Teil ist, entspricht der retro-reflektierenden Wand. Auf der rechten Seite können wir das Präzisionsstativ inklusive der Kamera betrachten.

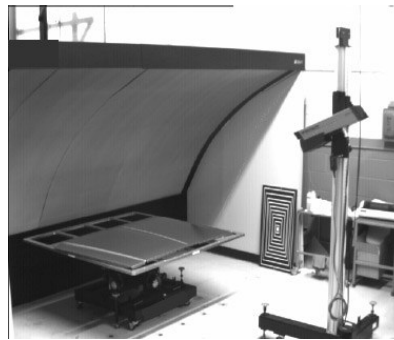


Abbildung 3.18 - Das DSight-System - Typischer Arbeitsplatz

■ Das Inspizieren von Oberflächen:

In einem DSight System existieren immer drei bedeutende optische Bestandteile: die Kamera, das zu inspizierende Objekt und die retro-reflektierende Wand. Diese Komponenten müssen so angeordnet werden, dass das Licht, das von der Kamera abgestrahlt wird, von dem Objekt auf die Wand reflektiert wird. Aufgrund der optischen Eigenschaften der Wand, wird dieses Licht zurück auf das Objekt reflektiert und von dort wieder zurück in die Linse der Kamera.

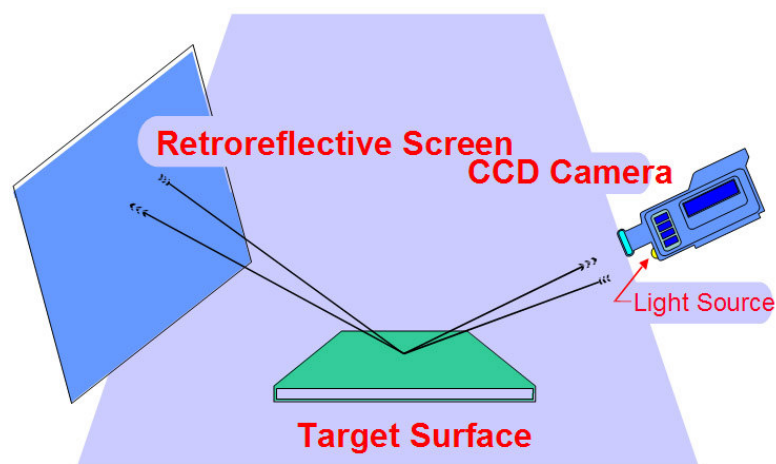


Abbildung 3.19 - Das DSight-System - Prinzip der Bildgewinnung

In dieser Anordnung erzwingt das System Neigungsvariationen der Oberfläche als Variationen in der Intensität des reflektierten Lichtes zu erscheinen. Daraus ergeben sich Abweichungen der Kontur im aufgenommenen Bild.

Wenn z.B. eine vollkommene spiegelnde Stahloberfläche mit dem DSight System analysiert werden würde, würde die Oberfläche in dem aufgenommenen Bild als gleichmäßiger Grauton erscheinen. An den Rändern des Objektes hätte es aber den Anschein, als würde dieser Grauton heller werden. Dies lässt sich mit dieser Technik nicht verhindern, stört das Inspizieren von Oberflächen jedoch nicht, da der Übergang gleichmäßig erfolgt. Defekte an der Oberfläche eines Objektes verursachen lokale Extrema, d.h. der Farbübergang ist abrupt.

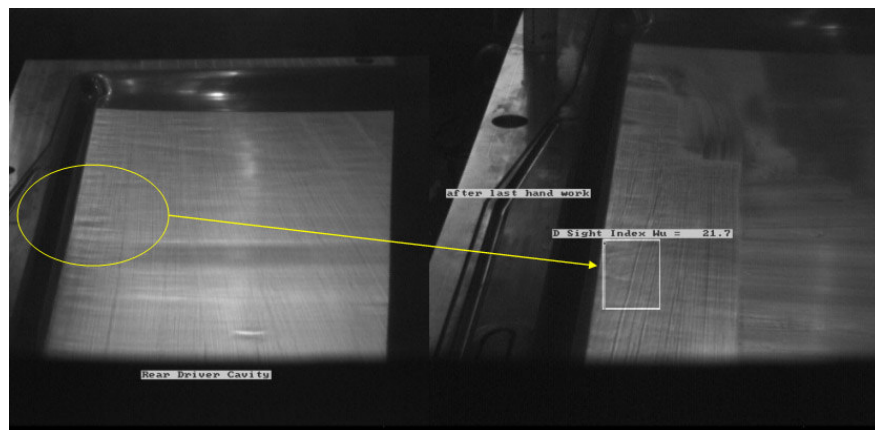


Abbildung 3.20 - Das DSight-System - Oberfläche eines inspizierten Objektes

Jeder Defekt, der auf der Oberfläche erkannt wird, besitzt seine eigene Signatur, d.h. man kann erkennen, ob es sich um eine Delle, einer Welligkeit oder um eine Kerbe handelt.

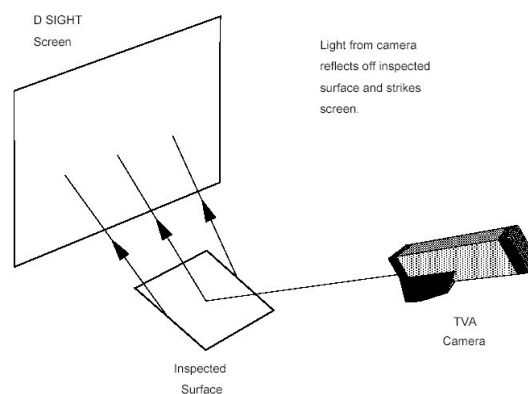


Abbildung 3.21 - Das DSight-System – Signaturen von Defekten

Jedoch hängt diese Signatur von dem Betrachtungswinkel der Kamera ab. So ist es möglich, dass ein Defekt auf einer Oberfläche kaum erkennbar ist, dreht man jedoch das Objekt in einen anderen Winkel, so wird der Fehler deutlich erkennbar.

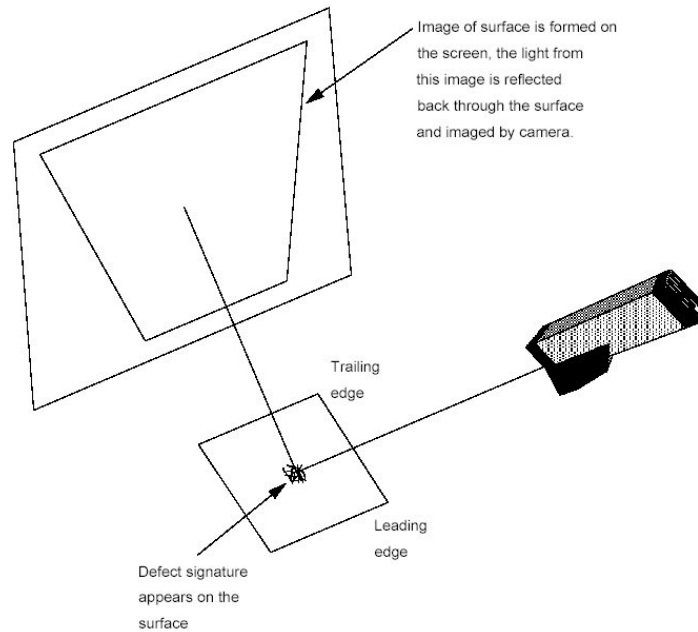


Abbildung 3.22 - Das DSight-System – Auswirkung des Blickwinkels

Somit lassen sich auch schwer erkennbare Fehler mit Hilfe dieses Systems feststellen. Da es sehr flexibel ist, bietet es teilweise mehr Möglichkeiten als ein fest installiertes System.

4 Datenbanksysteme

Eine Datenbank enthält eine Menge von Informationsdarstellungen (Daten) aus einem abgegrenzten Themenbereich. Die Daten werden dabei von einem Datenbankverwaltungssystem (DBMS) so verwaltet, dass sie im Weiteren unabhängig von den Anwendungsprogrammen sind, die sie benutzen.

Definition:

Ein Datenbankverwaltungssystem ist ein rechnergestütztes System, bestehend aus einer Datenbasis¹⁾ zur Beschreibung eines Ausschnitts der Realwelt sowie Programmen zum geregelten Zugriff auf diese Datenbasis²⁾.

Datenbankverwaltungssysteme sind heutzutage in der Datenverarbeitung nicht mehr wegzudenken. Durch die immer größer werdende Datenflut wird der Einsatz von Datenbanksystemen notwendig, um die Verbindung von Datenbeständen und Anwendungsprogrammen herzustellen, und die Verwendung unterschiedlicher Dateiformate zu ermöglichen.

Als Motivation für die Entwicklung und den Einsatz solcher Datenbankverwaltungssysteme können folgende Punkte angesehen werden:

1. **Redundanz:** Dieselbe Information wird doppelt gespeichert.
2. **Inkonsistenz:** Gleiche Information liegt in unterschiedlichen Versionen vor.
3. **Integritätsverletzung:** Die Einhaltung komplexer Integritätsbedingungen wird erschwert bzw. nicht eingehalten.
4. **Verknüpfungseinschränkungen:** Logisch verwandte Daten sind nur schwer miteinander zu verknüpfen, wenn sie in isolierten Dateien vorliegen.
5. **Mehrbenutzerprobleme:** Auftreten von Anomalien durch gleichzeitiges Editieren derselben Datei.
6. **Verlust von Daten:** Außer einem kompletten Backup wird kein Recovery-Mechanismus unterstützt.
7. **Sicherheitsprobleme:** Abgestufte Zugriffsrechte können nicht implementiert werden.
8. **Hohe Entwicklungskosten:** Fragen zur Dateiverwaltung müssen für jedes einzelne Anwendungsprogramm neu bedacht werden.

Solche Probleme treten z.B. durch die separate Abspeicherung von teilweise miteinander in Beziehung stehenden Daten durch verschiedene Anwendungen auf.

1) „Sammlung von logisch zusammengehörenden Tabellen, die eine auf die Dauer und für flexiblen Gebrauch ausgelegte Datenorganisation des Datenbestandes und der dazugehörigen Datenverwaltung umfasst“. (Zehnder 1987; in Küng 1994: 18)

2) vgl. Vornberger, O./Müller, O.: Datenbanksysteme, Vorlesung im SS 2001, achte Auflage, Osnabrück 2001, S.11

4.1 Datenbankaufbau

Im allgemeinen legt das Datenbankschema die Struktur der abgespeicherten Daten im Speicher fest, sagt aber nichts über die individuellen Daten aus.

Grundsätzlich lassen sich Datenbanksysteme in drei Abstraktionsebenen einteilen:

- Konzeptuelle Ebene
- Externe Ebene
- Interne Ebene

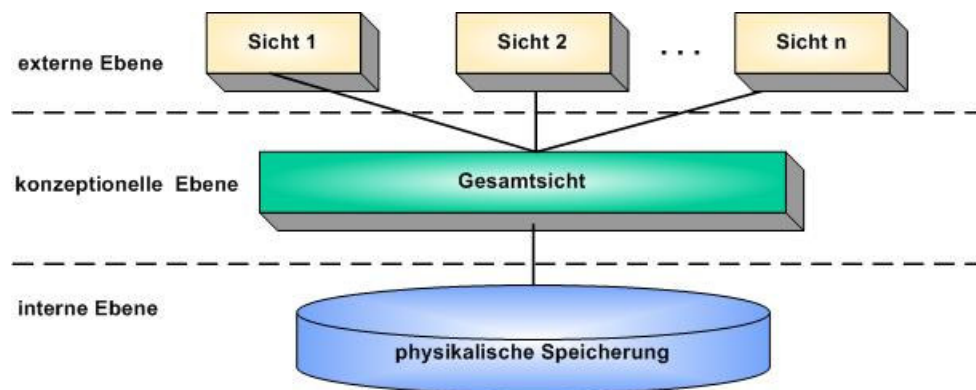


Abbildung 4.1 – Abstraktionsschema eines Datenbanksystems

Auf der konzeptuellen Ebene wird unabhängig von allen Anwenderprogrammen, die Gesamtheit aller Daten, ihre Strukturierung und ihre Beziehung untereinander beschrieben. Die Formulierung erfolgt mittels einer DDL. Das Ergebnis ist dabei das so genannte Datenbankschema (konzeptuelle Schema).

Auf der externen Ebene wird für jede Benutzergruppe eine anwendungsbezogene Sicht der Daten (VIEW) spezifiziert. Die Beschreibung erfolgt dabei ebenfalls mittels einer DDL. Der Umgang vom Benutzer erfolgt jedoch durch eine DML. Die DML ist für den Anwender interessant und besteht aus einer Anfragesprache, mit deren Hilfe die Informationen aus Dateien verknüpft werden können.

Die interne Ebene legt fest, wie die logisch beschriebenen Daten im Speicher abgelegt werden sollen. Es werden unter anderem Datensatzaufbau, Darstellung der Datenbestandteile, Dateiorganisation und Zugriffspfade geregelt.

Die Verbindungen zwischen den Ebenen werden durch Transformationsregeln definiert. Diese Regeln legen fest, wie die Objekte der einzelnen Ebenen aufeinander abgebildet werden. Die drei Ebenen gewährleisten einen bestimmten Grad an Datenunabhängigkeit:

- **physikalische Datenunabhängigkeit:** Die Modifikation der physikalischen Speicherstruktur verlangt nicht die Änderung der Anwenderprogramme.
- **logische Datenunabhängigkeit:** Modifikation der Gesamtsicht verlangt nicht die Änderung der Benutzersichten.

4.2 Modellierungskonzepte

Zur Modellierung der konzeptionellen Ebene verwendet man das Entity-Relationship-Modell (ER-Modell), welches einen Ausschnitt der Realwelt unter Verwendung von Entitäten (Entities) und Beziehungen (Relationships) beschreibt.

Definitionen:

Entity: Individuelles und identifizierbares Exemplar eines Objekts, einer Person eines Begriffes bzw. eines Ereignisses der realen oder der Vorstellungswelt. Entitäten werden repräsentiert durch eine Menge von Attributen, die gewisse Attributwerte annehmen können.

Entity-Typ: Oberbegriff für eine Menge von Entitäten der Miniwelt, die gleiche Attribute besitzen. Jeder Entity-Typ hat einen eindeutigen Namen.

Relationship: Beziehung zwischen den Entitäten. Sie drücken die Zusammenhänge zwischen Entitäten aus.

Die graphische Darstellung eines ER-Modells erfolgt durch das ER-Diagramm:

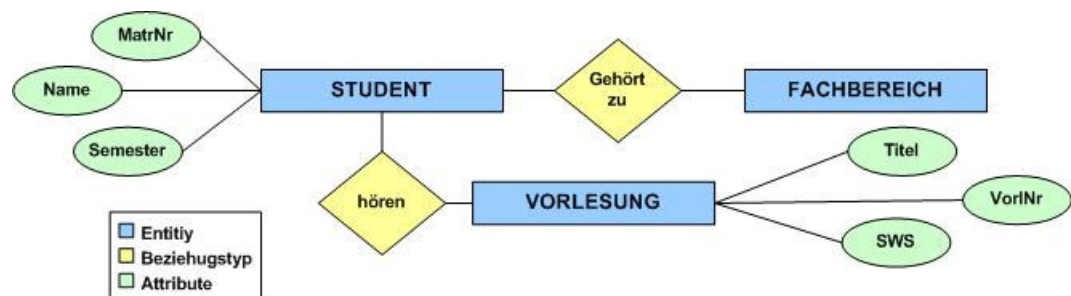


Abbildung 4.2 – Beispiel eines ER-Diagramms

Entity-Typen werden durch Rechtecke, Beziehungen durch Rauten und Attribute durch Ovale dargestellt.

Jedes Entity wird durch seine Attribut-Wert-Kombination eindeutig bestimmt. Eine minimale Menge von Attributen, welche die zugeordnete Entität innerhalb aller Entitäten gleichen Typs identifiziert, wird als Schlüssel oder Schlüsselkandidat bezeichnet. Treten mehrere Schlüsselkandidaten auf, wird einer als Primärschlüssel ausgewählt.

Die Schlüsselattribute werden häufig durch Unterstreichen gekennzeichnet.

Bsp.:

STUDENT = ({ MATRNR, NAME, GEB_DAT, WOHNORT })

Ein Beziehungstyp R zwischen den Entity-Typen E_1, E_2, \dots, E_n kann als Relation im mathematischen Sinn verstanden werden.

Es gilt:
$$R \subset E_1 \times E_2 \times E_3 \dots E_n$$

Mit n wird der Grad der Beziehung R beschrieben. Ein Element $(e_1, e_2, \dots, e_n) \in R$ beschreibt eine Instanz des Beziehungstyps.

Beziehungstypen können anhand ihrer Funktionalität charakterisiert werden:

■ **1:1-Beziehung (one-to-one):** Jeder Entität e_1 aus E_1 ist höchstens eine Entität e_2 aus E_2 zugeordnet und umgekehrt.

■ **1:N-Beziehung (one-to-many):** Einem Objekt des Typs A sind beliebig viele Objekte des Typs B zugeordnet.

■ **N:1-Beziehung (many-to-one):** Analog zu 1:N-Beziehung

■ **N:M-Beziehung (many-to-many):** Jede Entität aus E_1 kann mit beliebig vielen Entitäten aus E_2 in Beziehung stehen und umgekehrt. Es bestehen keinerlei Restriktionen.

4.3 Datenmodelle

Je nach Anforderung werden Datenbankmanagementsysteme von unterschiedlichen Datenmodellen gebildet, die durch die logische Ebene definiert sind. Sie beinhalten die Festlegung der Datenstrukturen zur Verwaltung der Daten und die dazu gehörenden Operatoren.

Unter einer Reihe von entwickelten Datenmodellen haben sich insbesondere folgenden Datenmodelle durchgesetzt:

1. Hierarchisches Datenmodell
2. Netzwerkmodell
3. Relationales Datenmodell
4. Objektorientiertes Datenmodell

Das hierarchische Modell und das Netzwerkmodell haben heute nur noch eine historische Bedeutung.

Relationale Datenbanksysteme dagegen sind inzwischen marktbeherrschend. In der Praxis hat sich gezeigt, dass das relationale Datenmodell in den meisten Fällen den entsprechenden Anforderungen am ehesten genügt und so am häufigsten zum Einsatz kommt.

Objektorientierte Systeme fassen strukturelle und verhaltensmäßige Komponenten in einem Objekttyp zusammen. Sie gelten als die nächste Generation von Datenbanksystemen.

In Abhängigkeit von dem zu verwendeten Datenbanksystem und den Anforderungen an das System wählt man zur rechnergerechten Umsetzung des ER-Modells in der Regel eines der oben genannten Datenmodelle.

4.3.1 Das Hierarchische Datenmodell

Datenbanksysteme, die auf dem hierarchischen Datenmodell basieren, haben nach heutigem Stand nur eine eingeschränkte Modellierfähigkeit. Sie verlangen außerdem vom Anwender eine gute Kenntnis der internen Ebene.

Dieses Modell unterstützt nur baumartige Beziehungen. Die Daten werden in hierarchische Abhängigkeiten gesetzt. Operationen navigieren durch die Baumstruktur.

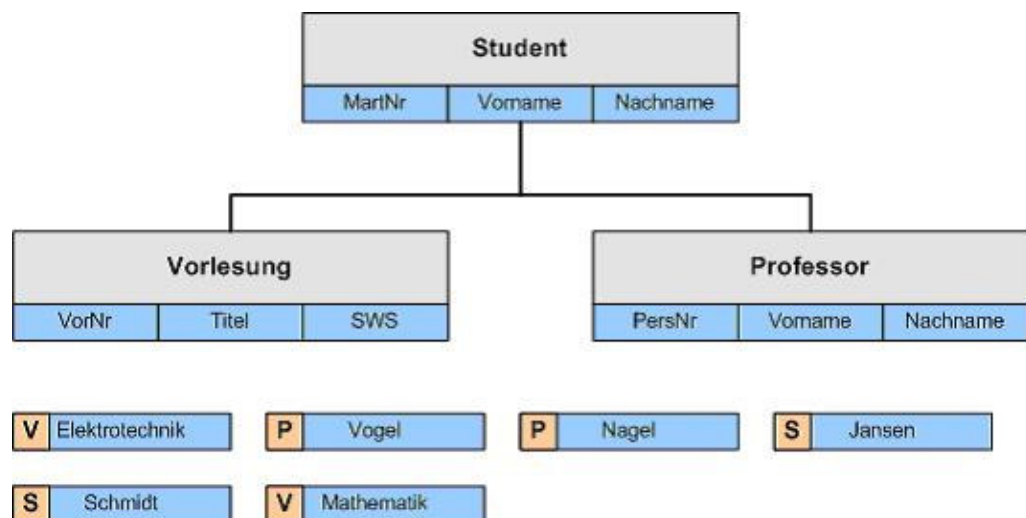


Abbildung 4.3 – Schema eines hierarchischen Datenmodells

Diese Struktur unterstützt relativ einfach die 1:N-Beziehung, während eine N:M-Beziehung nur durch redundante Speicherung von Information modelliert werden kann. Diese Komplexität erschwert erheblich den Umgang mit der Datenbank.

Aus diesem Grund hat sich das hierarchische Datenmodell nicht durchsetzen können.

4.3.2 Das Netzwerkmodell

Das Netzwerkmodell baut auf dem hierarchischen Modell auf. Zusätzlich werden Beziehungen durch so genannte Pointer schon bei dem Entwurf der Datenbank festgelegt. Es werden jedoch nicht die Segmente, sondern die logischen Beziehungen dargestellt. Durch die Einführung von logischen Pointern wird das Problem der Redundanz gelöst. Durch die Erstellung aller denkbaren Pointer ergibt sich die Möglichkeit einer mehrdimensionalen Auswertung der Datenbank.

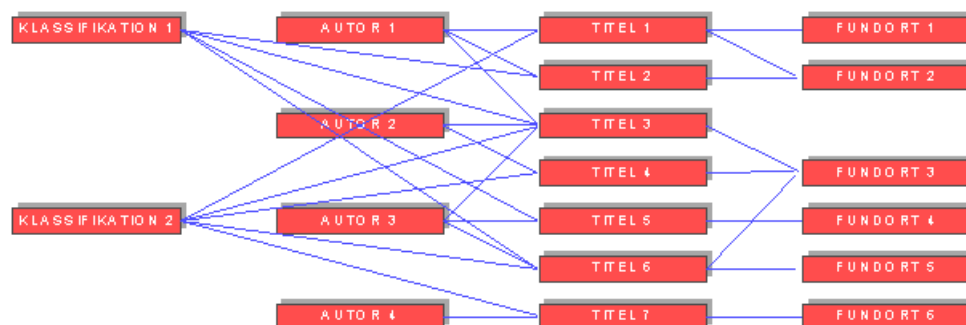


Abbildung 4.4 – Schema eines Netzwerkmodells

4.3.3 Das Relationale Datenmodell

Der wichtigste Unterschied des relationalen Datenmodells zum Netzwerkmodell und zum hierarchischen Modell ist, dass nur Daten aber keine Beziehungen dargestellt werden.

Die Daten werden in Tabellen, den sog. Relationen¹⁾ gespeichert und durch Operatoren miteinander verknüpft, die wiederum als Werte in Tabellenform abgelegt sind.

Relationale Datenbanken bieten verschiedene Möglichkeiten Daten mehrerer Tabellen zueinander in Beziehung zu setzen. Grundsätzlich muss in beiden Tabellen ein Schlüsselfeld mit jeweils identischem Inhaltstyp und zumindest teilweise identischen Werten in beiden Tabellen zur Verfügung stehen, über das die beiden Tabellen verbunden werden können.

Die typischen Operationen auf eine relationale Datenbank sind:

- Selektion
- Projektion
- Verbund

¹⁾ Menge von Tupeln (Daten und Beziehungen zwischen Daten, deren Umsetzung in einer Tabellenform untergebracht ist), die durch Angabe des Namens und der beteiligten Attribute festgelegt ist

Bei der Selektion werden alle Tupel einer Relation mit gewissen Attributeigenschaften ausgewählt.

Die Projektion hingegen filtert nur gewisse Spalten heraus. Es werden nur diejenigen Tupel aus mehreren Relation ausgewählt, die bezüglich gewisser Spalten übereinstimmen.

MatrNr	Vorname	Nachname	VorNr	Titel	Umfang
264287	Hans	Bertel	1	Algebra	4 SWS
264231	Thomas	Dürer	2	Stochastik	2 SWS
264341	Peter	Böck	3	Elektronik	4 SWS
265176	Gerda	Semmel	4	Robotik	1 SWS
265233	Fritz	Berger	5	Netzwerktechnik	4 SWS
266765	Anita			DB-Systeme	2 SWS
...

MatrNr	VorNr
264287	6
264231	2
266765	3
...	...

Abbildung 4.5 – Tabellen eines relationalen Datenmodells

Die erste Zeile einer Tabelle gibt jeweils die Struktur einer Tabelle an (Anzahl und Benennung der Spalte). Diese Strukturinformation wird als Relationenschema bezeichnet.

Die weiteren Einträge einer Tabelle stellen eine Relation zu diesem Schema dar. Eine einzelne Zeile in der Tabelle wird als Tupel bezeichnet. Da die Spaltenüberschriften auch als Attribute bezeichnet werden, ist ein einzelner Eintrag in einer Attributspalte einer Spalte der so genannte zugehörige Attributwert dieses Tupels.

Die Konventionen sollen in Abbildung 4.6 noch einmal verdeutlicht werden:

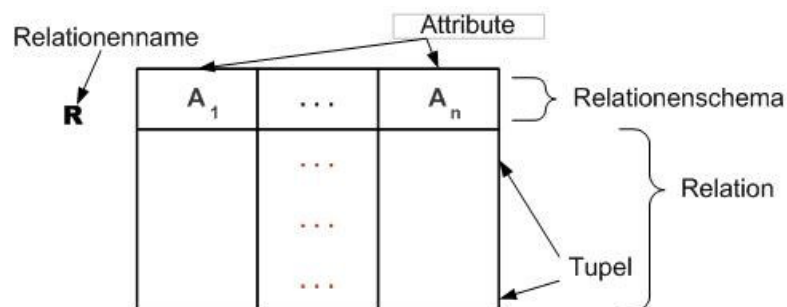


Abbildung 4.6 – Begriffsbildung und Tabellendarstellung im Relationenmodell

4.3.4 Das Objektorientierte Datenmodell

Der Nachteil des relationalen Datenmodells ist, dass es sich sehr schlecht für den räumlichen Zugriff eignet, da zwar die Koordinaten eines Punktes in Form von Tabellen gespeichert werden können, die Beziehungen dieses Punktes jedoch zu anderen Punkten nicht mit in diese Tabelle eingehen können.

Diese Unzulänglichkeiten des relationalen Datenmodells bei komplexeren Anwendungen, wie z.B. bei wissenschaftlichen oder multimedialen Zwecken führten zur Entwicklung der objektorientierten DBMS.

Ein OODB ist ein DBMS mit einem objektorientierten Datenmodell. Der Unterschied zu dem RDBMS ist die Wertabhängigkeit eines Objektes.

Während bei einem RDBMS miteinander verknüpfte Daten innerhalb eines Wertebereiches liegen müssen, werden bei dem OODB Attribute zu einem Objekt zusammengefasst und mit eigenen Operatoren versehen. Gleichartige Objekte werden zu Klassen zusammengefasst, die mit jeweils gleichen Operatoren behandelt werden.

Eine Klasse repräsentiert einen Entity-Typ zusammen mit darauf erlaubten Operationen. Die Attribute müssen nicht atomar sein, sondern bestehen ggf. aus Tupeln, Listen und Mengen. Die Struktur einer Klasse kann an eine Unterklasse vererbt werden.

Folgende Tabelle zeigt die Definition eines Entity-Typen Person mit seiner Spezialisierung Student inklusiv der Beziehung „hoert“.

```

class Person
  type tuple ( name      : String,
               geb_datum : Date,
               kinder    : list(Person))
end;

class Student inherit Person
  type tuple ( matr_nr   : Integer,
               hoert     : set(Vorlesung))
end;

class Vorlesung
  type tuple ( titel     : String,
               gehoert   : set(Student))
end;

```

Tabelle 4.1 – Definition von Entity-Typen bei einem OODB

Das Objekt ist also eine kleine Teilwelt für sich. Das Objekt ist isoliert definierbar und obliegt nicht einer allgemeinen Datenmanipulation. Es kann also mit speziellen Operatoren manipuliert werden. Daher erscheint das objektorientierte Datenmodell vom Konzept gut geeignet, raumbezogene Problemstellungen zu bearbeiten.

4.4 Relationale Entwurfstheorie

Ein guter Datenbankentwurf bringt wie schon erwähnt eine Menge von Vorteilen mit sich. Ein wichtiger Teilschritt beim Entwurf eines relationalen Datenbankmodells ist die Normalisierung. Die Normalisierung dient dem Verfeinern des logischen Entwurfs (vgl. Kapitel 4.3.3), wie z.B. der Vermeidung von Redundanzen durch Aufspalten von Relationenschemata.

Die Redundanzvermeidung wird durch die Anwendung der Normalformen erreicht, die mit Hilfe von Abhängigkeiten zwischen Attributen definiert werden.

4.4.1 Funktionale Abhängigkeiten

Die am häufigsten eingesetzten Abhängigkeiten sind die funktionalen Abhängigkeiten.

Eine funktionale Abhängigkeit gilt genau dann innerhalb einer Relation zwischen Attributmengen α und β , wenn jedem Tupel der Relation der Attributwert unter den α -Komponenten der Attributwert der β -Komponenten vorliegt.

Gehen wir von einem Relationenschema \mathfrak{R} mit der Ausprägung \mathbf{R} aus. Die funktionale Abhängigkeit wird mathematisch dargestellt durch:

$$\alpha \rightarrow \beta$$

Dabei sind nur solche Ausprägungen zugelassen, für die gilt:

$$\forall r, t \in \mathfrak{R} : r.\alpha = t.\alpha \Rightarrow r.\beta = t.\beta$$

Das Überprüfen einer vermuteten funktionalen Abhängigkeit in einer Relation kann im allgemeinen mit folgendem einfachen Algorithmus erfolgen:

1. Sortiere \mathbf{R} nach α -Werten
2. Weisen alle Gruppen bestehend aus Tupeln mit gleichen α -Werten auch gleiche β -Werte auf, dann gilt $\alpha \rightarrow \beta$, sonst nicht

4.4.2 Schlüssel

Gegeben sei ein Relationenschema \mathfrak{R} .

$\alpha \subseteq R$ ist in diesem Relationenschema ein Superschlüssel, falls gilt: $\alpha \rightarrow \mathfrak{R}$.

Dieser Superschlüssel legt fest, dass alle Attribute von α abhängen. Es wird aber noch nicht klar, ob α eine minimale Menge von Attributen enthält.

Eine funktionale Abhängigkeit von β zu α ist dann gegeben, wenn gilt:

1. $\alpha \rightarrow \beta$
2. $\forall A \in \alpha : \alpha - \{A\} \not\rightarrow \beta$

In diesem Falle wird α als Schlüsselkandidat bezeichnet.

Aus der Menge dieser Schlüsselkandidaten wird einer als ausgezeichnete Schlüssel (Primärschlüssel) ausgewählt.

4.4.3 Bestimmung einer funktionalen Abhängigkeit

Die Bestimmung einer funktionalen Abhängigkeit aus einem vorgegebenen Relationenschema soll anhand eines Beispiels verdeutlicht werden.

Folgendes Relationenschema ist vorgegeben:

PROFESSOREN : {[PNR, NAME, RANG, RAUM, ORT, STR, PLZ, VORWAHL, BLAND, LANDREG]}

PNR : eindeutige Personalnummer des Professors
NAME : Name des Professors
RANG : Rang des Professors
RAUM : Arbeitsraum des Professors
ORT : eindeutiger Wohnort (Stadt) des Professors
STR : Straßename des zugehörigen Wohnortes
PLZ : Postleitzahl des zugehörigen Wohnsitzes
VORWAHL : Telefonvorwahl des Ortes
BLAND : Name des Bundeslandes
LANDREG : eindeutige Partei des Ministerpräsidenten

Dabei gelten folgende Abhängigkeiten:

1. **{PNR}** → {PNR, NAME, RANG, RAUM, ORT, STR, PLZ, VORWAHL, BLAND, LANDREG}
2. **{ORT, BLAND}** → {VORWAHL}
3. **{PLZ}** → {BLAND, ORT}
4. **{ORT, BLAND, STR}** → {PLZ}
5. **{BLAND}** → {LANDREG}
6. **{RAUM}** → {PNR}

Daraus ergeben sich weitere Abhängigkeiten:

- {RAUM}** → {PNR, NAME, RANG, RAUM, ORT, STR, PLZ, VORWAHL, BLAND, LANDREG}
- {PLZ}** → {LANDREG}

Bei einer vorgegebenen Menge M von funktionalen Abhängigkeiten über der Attributmenge A ist die Menge M^+ aller aus F ableitbaren funktionalen Abhängigkeiten interessant. M^+ wird auch als Hülle von M bezeichnet.

Zur Bestimmung dieser Hülle reichen folgende Inferenzregeln (Armstrong Axiome) aus:

- Reflexivität** : Aus $\beta \subseteq \alpha$ folgt : $\alpha \rightarrow \beta$
- Verstärkung**: Aus $\alpha \rightarrow \beta$ folgt : $\alpha\gamma \rightarrow \beta\gamma$ für $\gamma \subseteq A$
- Transitivität** : Aus $\alpha \rightarrow \beta$ und $\beta \rightarrow \gamma$ folgt : $\alpha \rightarrow \gamma$

Die Armstrong-Axiome sind korrekt und vollständig. Das bedeutet, dass nur solche funktionale Abhängigkeiten abgeleitet werden, die von jeder Ausprägung für die M erfüllt ist (korrekt).

Außerdem lassen sich alle Abhängigkeiten ableiten, die durch F logisch impliziert werden.

Weitere Axiome, die sich ableiten lassen:

- Vereinigung** : aus $\alpha \rightarrow \beta$ und $\alpha \rightarrow \gamma$ folgt : $\alpha \rightarrow \beta\gamma$
- Dekomposition** : aus $\alpha \rightarrow \beta\gamma$ folgt : $\alpha \rightarrow \beta$ und $\alpha \rightarrow \gamma$
- Pseudotransitivität** : aus $\alpha \rightarrow \beta$ und $\beta\gamma \rightarrow \delta$ folgt $\alpha\gamma \rightarrow \delta$

Aus den funktionalen Abhängigkeiten 1-6 für das Relationenschema PROFESSOREN lässt sich zeigen: $\{\text{PLZ}\} \rightarrow \{\text{LANDREG}\}$

- $\{\text{PLZ}\} \rightarrow \{\text{BLAND}\}$ (Dekomposition von der funkt. Abhängigkeit Nr.3)
- $\{\text{BLAND}\} \rightarrow \{\text{LANDREG}\}$ (funkt. Anhängigkeit Nr.6)
- $\{\text{PLZ}\} \rightarrow \{\text{LANDREG}\}$ (Transitivität)

Oft ist die Menge von Attributen α^+ interessant, die von α gemäß der Menge M von funktionalen Abhängigkeiten bestimmt werden.

$$\alpha^+ := \{\beta \subseteq A \mid \alpha \rightarrow \beta \in M^+\}$$

Weiterhin gilt:

$$\alpha \rightarrow \beta \text{ folgt aus den Armstrong-Axiomen genau dann wenn } \beta \in \alpha^+$$

Die Menge α^+ kann aus einer Menge M von funktionalen Abhängigkeiten und einer Menge von Attributen α wie folgt bestimmt werden:

$$X^0 := \alpha$$

$$X^{i+1} := X^i \cup \gamma \text{ falls } \beta \rightarrow \gamma \in M \wedge \beta \subseteq X^i$$

Von einer Abhängigkeit, deren linke Seite in der Lösungsmenge enthalten ist, wird die rechte Seite hinzugefügt. Dieser Algorithmus wird dann beendet, wenn keine Veränderung mehr zu erreichen ist.

Es gilt dann:

$$X^{i+1} := X^i$$

4.4.4 Normalisierung

Unter Normalisierung versteht man die Zerlegung eines Relationenschemas \mathcal{R} in die Relationenschemata $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_n$, die jeweils nur eine Teilmenge der Attribute von \mathcal{R} aufweisen ($\mathcal{R}_i \subseteq \mathcal{R}$).

Folgende Regeln müssen aber bei dieser Zerlegung beachtet werden:

1. **Verlustlosigkeit:** Die in der ursprünglichen Ausprägung R des Relationenschemas \mathcal{R} enthaltenen Informationen müssen aus den Ausprägungen der neuen Schemata $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_n$ rekonstruierbar sein.
2. **Abhängigkeitserhaltung:** Die für \mathcal{R} geltenden funktionalen Abhängigkeiten müssen auf die Schemata $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_n$ übertragbar sein.

Als Beispiel soll das Relationenschema \mathfrak{R} in zwei Relationenschemata zerlegt werden. Dafür muss gelten: $\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2$

Für die Ausprägung R von \mathfrak{R} definieren wir die Ausprägungen R_1 und R_2 von \mathfrak{R}_1 und \mathfrak{R}_2 wie folgt:

$$R_1 := \Pi_{R_1}(R); \quad R_2 := \Pi_{R_2}(R);$$

Wir bezeichnen eine Zerlegung als verlustlos, wenn für jede gültige Ausprägung R von \mathfrak{R} gilt:

$$R_1 \bowtie R_2$$

4.4.4.1 Die erste Normalform

Ein Relationenschema \mathfrak{R} ist in der ersten Normalform, wenn alle Attribute atomare Wertebereiche haben. Zusammengesetzte oder mehrwertige Domänen sind verboten.

Beispiel:

ELTERN		
Vater	Mutter	Kind
Hans	Maria	{Andrea,Martha}
Hans	Monika	{Tobi,Robert}
Kurt	Claudia	Jens

nicht in 1. Normalform !!!

ELTERN in 1. Normalform		
Vater	Mutter	Kind
Hans	Maria	Andrea
Hans	Maria	Martha
Hans	Monika	Tobi
Hans	Monika	Robert
Kurt	Claudia	Jens

Abbildung 4.7 – Tabellen in erster Normalform

4.4.4.2 Die zweite Normalform

Ein Relationenschema \mathfrak{R} ist in zweiter Normalform falls gilt:

1. \mathfrak{R} ist in der ersten Normalform
2. Jedes Nichtprimär-Attribut¹⁾ $N \in \mathfrak{R}$ ist voll funktional abhängig von jedem Schlüsselkandidaten.

¹⁾ Ein Attribut heißt Primärattribut, wenn es in mindestens einem Schlüsselkandidaten vorkommt, andernfalls ist es ein Nichtprimärattribut.

Beispiel:

Vorlesung	Dozent	Termin	Raum
Elektrotechnik	Dolbe	Mo, 9:30	G201
Statistik	Lorenz	Di, 14:00	G100
Statistik	Lorenz	Mi, 11:15	G100
BWL	Lorenz	Fr, 8:00	G100

Abbildung 4.8 - Tabelle in zweiter Normalform

Die Schlüsselkandidaten lauten:

- {Vorlesung, Termin}
- {Dozent, Termin}
- {Raum, Termin}

Da alle Attribute in mindestens einem Schlüsselkandidaten vorkommen gibt es keine Nichtprimärattribute. Diese Relation ist daher in der zweiter Normalform.

4.4.5 Die dritte Normalform

Seien X, Y, Z Mengen von Attributen eines Relationenschemas \mathcal{R} mit Attributmenge U .

Z heißt transitiv abhängig von X , falls gilt:

$$X \cap Z = \emptyset$$

$$\exists Y \subset U : X \cap Y = \emptyset, Y \cap Z = \emptyset$$

$$X \rightarrow Y \rightarrow Z, Y \not\rightarrow X$$

Ein Relationenschema ist in dritter Normalform falls gilt:

1. \mathcal{R} ist in der zweiten Normalform
2. Jedes Nichtprimärattribut ist nicht transitiv abhängig von jedem Schlüsselkandidaten

Es existieren auch höhere Normalformen (4NF, 5NF, DKNF), die versuchen dem Verlust an Information vorzubeugen. Durch die Anwendung höherer Normalformen auf ein Relationenschema resultieren aber immer mehr spezialisierte Tabellen. Dies kann unter Umständen einen negativen Effekt auf die Leistung haben, da sich die steigende Anzahl der Tabellen auf die Komplexität der SQL-Statements auswirkt.

Im Normalfall sollte versucht werden eine Relation mit der höchst möglichen Normalform in Einklang zu bringen. Ein Verstoß gegen die Normalformen sollte wenn möglich vermieden werden. Allerdings muss erwähnt werden, dass ein optimales Datenbankdesign sehr oft leicht denormalisiert ist.

4.5 SQL

Die Structured Query Language ist eine Anfragesprache, mit deren Hilfe die Informationen verknüpft werden können.

Abfragesprachen bilden die grundlegenden Voraussetzungen für einen einfachen Umgang mit den Daten.

Es existieren verschiedene Konzepte für formale Sprachen zur Formulierung einer Anfrage an ein relationales Datenbanksystem:

1. **Relationenalgebra**
2. **Relationenkalkül**
3. **SQL**
4. **Query by Example**

Die gebräuchlichste standardisierte DML ist jedoch die Structured Query Language.

SQL geht zurück auf den von IBM Anfang der 70er Jahre entwickelten Prototyp *System R* mit der Anfragesprache *Sequel*. Anfang der 80er Jahre begann das American National Standards Institut (ANSI) mit einer Standardisierung. Die aktuellste Standard wurde 1992 als SQL92 (SQL2) verabschiedet. Zur Zeit ist die Diskussion um den SQL3 Standard, der das Trigger-Konzept und objektorientierte Konzepte berücksichtigen soll.

SQL wurde entwickelt, um mit Datensätzen umzugehen und ist nicht nur eine Anfragesprache, sondern eine vollständige Datenbanksprache. Dabei können die Datensätze aus einer oder mehreren Tabellen eines relationalen Datenbanksystems entnommen werden.

Der Umgang mit dieser Sprache ist deshalb relativ unkompliziert, weil er sehr nah an der natürlich-sprachlichen Formulierung eines Befehls liegt.

Das Grundkonzept dieser Anfragesprache beruht auf dem Begriff „Abbildung“.

Im allgemeinen weist ein SQL-Anweisungsblock folgendes Format auf:

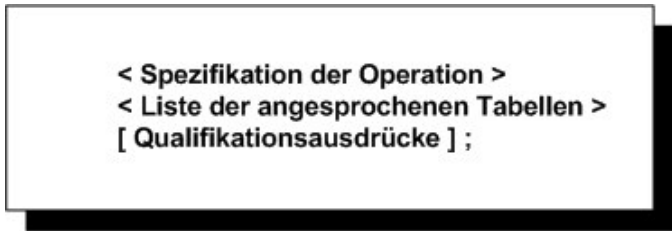
Das Diagramm zeigt ein rechteckiges Feld mit einer doppelten schwarzen Rahmenlinie. Innerhalb des Feldes sind drei Zeilen Text in einer hierarchischen Struktur angeordnet: Die erste Zeile beginnt mit einem schließenden Winkelgeschweifchen '>', gefolgt von 'Spezifikation der Operation'. Die zweite Zeile beginnt mit einem schließenden Winkelgeschweifchen '>', gefolgt von 'Liste der angesprochenen Tabellen'. Die dritte Zeile beginnt mit einem öffnenden eckigen Klammer '['', gefolgt von 'Qualifikationsausdrücke', und endet mit einer schließenden eckigen Klammer ']' und einem semikolon ';'.

Abbildung 4.9 – Format eines SQL-Anweisungsblocks

Im Sprachumfang von SQL werden unter anderem folgende Aspekte in Bezug auf ein Datenbanksystem berücksichtigt:

1. **Datenbank-Anfrage** (Query Facility)
2. **Datenbank-Updates** (Data Manipulation Facility)
3. **Datendefinition und Integrität** (Data Definition Facility)
4. **Zugriffsüberwachung** (Data Control Facility)
5. **Kopplung mit einer Wirtssprache** (Host Language Coupling)

Die wichtigsten Befehle, die SQL bereitstellt werden in nachfolgender Tabelle aufgelistet:

Befehl	Verwendung
COMMIT	Schließt eine Transaktion ab und macht alle Änderungen permanent
CREATE INDEX	Erstellt einen Index
CREATE TABLE	Erstellt eine Tabelle
CREATE VIEW	Erstellt eine logische Sicht anhand eines SELECT-Statements
DELETE	Löscht Tabellenzeilen
DROP	Löscht Strukturelemente, z.B. DROP TABLE
GRANT	Erteilt Berechtigungen
INSERT	Fügt einen oder mehrere Datensätze ein. Die Daten werden dem INSERT entweder in einer VALUES-Klausel mitgegeben oder stammen aus einem SELECT
REVOKE	Entzieht Berechtigungen
ROLLBACK	Macht eine Transaktion komplett rückgängig und verwirft alle Änderungen
SELECT	Liest Daten aus der Datenbank. Kann mehrere Tabellen verknüpfen, die Ausgabe sortieren, Daten gruppieren und aggregieren (Summe, Maximum, Minimum, Durchschnitt, Anzahl)
UPDATE	Ändert Daten. Die betroffenen Sätze werden durch eine WHERE-Klausel angegeben.

Tabelle 4.2 – Die wichtigsten SQL-Statements

4.6 Relationale Datenbank APIs

Relationale Datenbank APIs stellen eine Schnittstelle zwischen einer Programmiersprache (wie z.B. C++) und SQL dar. Sie bieten also den Anwendungsprogrammen die Möglichkeit mit relationalen Datenbanken zu kommunizieren.

Einige Datenbank APIs sind datenbankspezifisch, d.h. sie sind dazu entwickelt worden, um nur mit einem bestimmten Datenbanktyp eines bestimmten Herstellers zu kommunizieren. Andere (wie z.B. ODBC) versuchen eine offene Schnittstelle für alle relationalen Datenbanken bereitzustellen.

Welche Datenbank API für eine Applikation gut geeignet ist, hängt von der Wahl der Datenbank ab. Für die meisten relationalen Datenbanken haben sich OLE DB und ADO als die modernsten und robustesten APIs bewährt.

4.6.1 Überblick über Datenbank Client-Technologien

Eine Datenbank ist ein sehr komplexer Teil an Software. Die Kommunikation einer Anwendung mit einer Datenbank über deren natürliche Schnittstelle kann unter Umständen sehr aufwendig und kompliziert werden. Datenbank Client-Technologien versuchen diesen Vorgang zu vereinfachen, indem sie eine weniger komplexe Schnittstelle zur Verfügung stellen. Im allgemeinen versuchen diese Datenbank Client-Technologien ihre Schnittstelle als eine allgemeine Schnittstelle zu verschiedenen Datenbanksystemen zu verwirklichen.

Die wichtigsten Datenbankschnittstellen der Windows Plattform sind:

- ODBC
- DAO
- RDO
- OLE DB
- ADO

4.6.1.1 Open Database Connectivity (ODBC)

ODBC wurde in den späten 80er bzw. frühen 90er Jahren entwickelt, um eine allgemeine Schnittstelle zur Entwicklung von Client-Software für relationale Datenbanken zur Verfügung zu stellen. ODBC ist eine einzelne, einfache API für Client-Anwendungen, um mit verschiedenen Datenbanken arbeiten zu können. Die Anwendungen, die eine ODBC API verwenden können mit jeder beliebigen Datenbank kommunizieren, die einen ODBC Treiber besitzt.

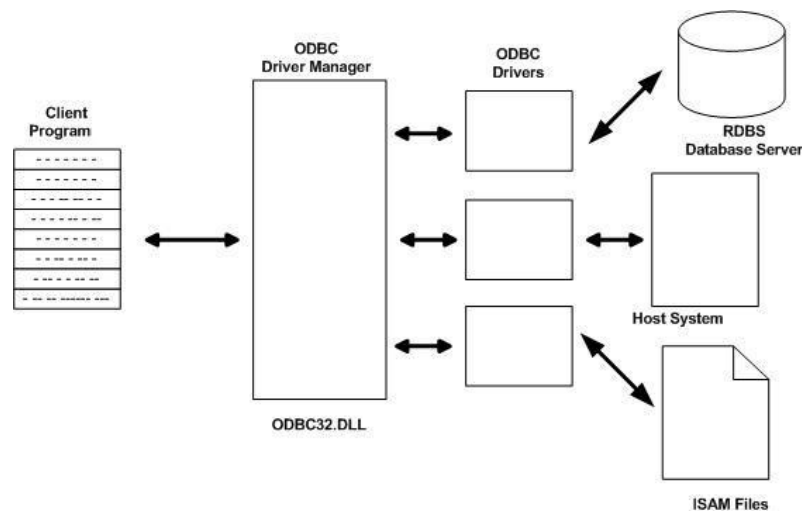


Abbildung 4.10 – ODBC Architektur

ODBC ist mittlerweile sehr populär und wurde als Standard-Interface für relationale Datenbanken akzeptiert.

Der Nachteil dieser Technologie ist, dass sie ausschließlich auf relationale Datenbanken beschränkt ist.

4.6.1.2 Data Access Objects (DAO)

DAO ist eine Sammlung von (COM-) Automationsschnittstellen für die Microsoft Access/Jet Database Engine. DAO kann dadurch mit Access/Jet Datenbanken direkt kommunizieren. Über die Jet Engine kann DAO aber auch mit anderen Datenbanken kommunizieren.

Die COM-basierte Automationsschnittstelle von DAO ist mehr als eine funktionsbasierte API. Sie liefert ein Objektmodell für die Datenbankprogrammierung.

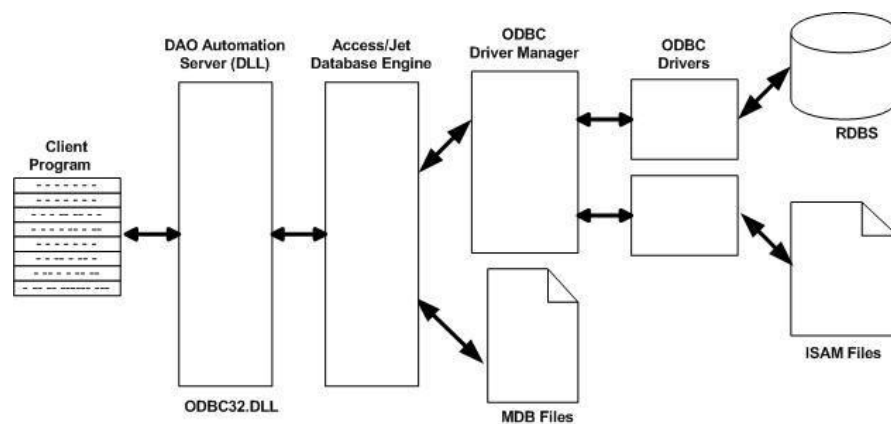


Abbildung 4.11 – DAO Architektur

DAO besitzt einen Satz an Objekten für die Verbindung zur Datenbank und Operationsausführung auf den Daten. Diese Objekte können relativ einfach in den Quellcode einer objektorientierten Anwendung integriert werden.

Der Nachteil dieser Technologie ist, dass alle Datenbankaufrufe und alle Daten aus der Datenbank die Access/Jet Engine passieren müssen. Unter Umständen kann dies zu einem „Flaschenhals“ für die Anwendung werden, die auf diese Datenbank zugreift.

4.6.1.3 Remote Data Objects (RDO)

RDO wurde ursprünglich als eine Abstraktion der ODBC API für Visual Basic Programmierer entwickelt. Dadurch ist RDO sehr nahe an ODBC und Visual Basic gebunden. RDO ist erheblich einfacher als ODBC, verliert jedoch im Vergleich dazu an Funktionalität was den Zugriff auf untere Ebenen angeht.

Da RDO direkt die ODBC API aufruft, kann eine gute Leistungsqualität für eine Applikation erreicht werden

4.6.1.4 OLE DB

OLE DB stellt eine Erweiterung zu ODBC dar. OLE DB besitzt eine COM-Schnittstelle für die Datenbankprogrammierung und ermöglicht dabei relationale und nicht-relationale Datenbanken anzusprechen.

Die COM-Schnittstelle ist robuster und flexibler als andere traditionelle Datenbankschnittstellen.

Folgende Abbildung demonstriert das Verhältnis zwischen Konsumenten (Consumers) und OLE DB Provider:

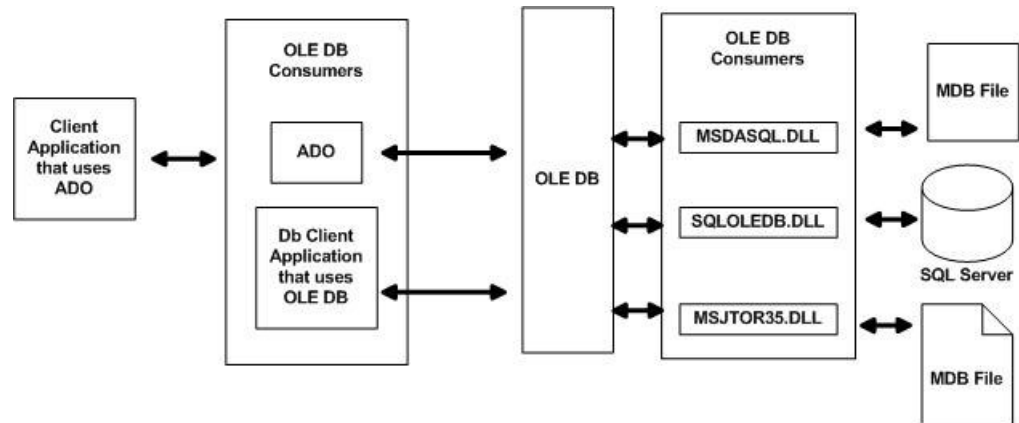


Abbildung 4.12 – OLE DB Consumer und Provider

OLE DB ist die Programmierschnittstelle von Microsoft, die auf der Systemebene agiert. Sie definiert eine Hierarchie von vier Hauptobjekten:

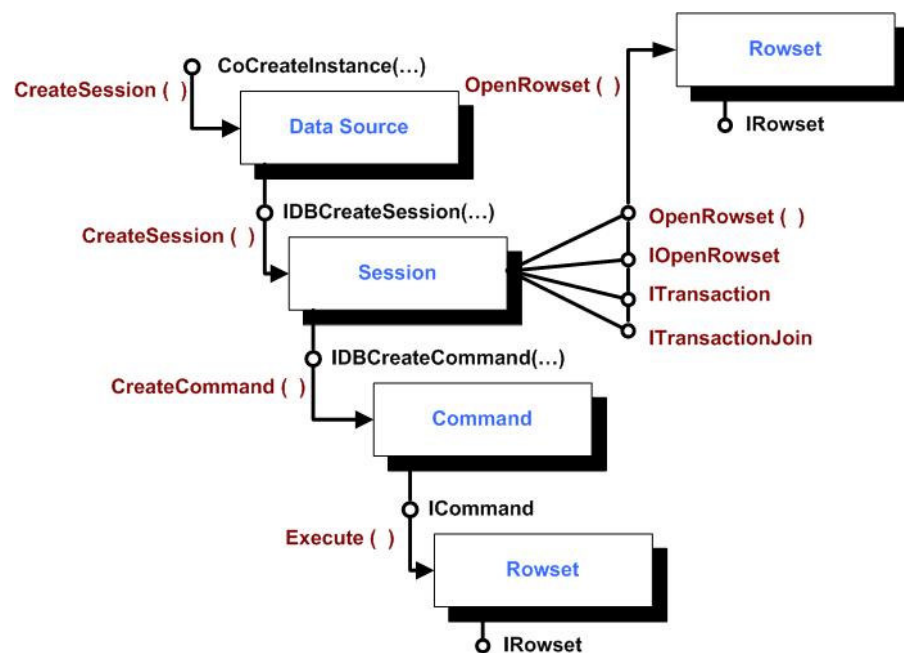


Abbildung 4.13 – OLE DB Hauptobjekte

Das Data Source Object enthält Funktionen, die einen bestimmten Data Provider identifizieren. Anschließend verifiziert es, ob ein Benutzer die entsprechenden Rechte hat, um eine Verbindung zu diesem Data Provider herzustellen und initialisiert diese anschließend.

Das Session Object definiert dialogorientierte Arbeitsschritte, die bei einer Verbindung anfallen.

Das Command Object enthält all diejenigen Funktionen, die es einem Verbraucher ermöglichen die Ausführung von Datendefinitions- oder Datenmanipulationsbefehlen (z.B. Datenbankabfragen) einzuleiten.

Rowset Objekte stellen eine allgemeine Repräsentation der Daten dar. Sie werden normalerweise entweder direkt von einer Session oder als Resultat einer Befehlsausführung erzeugt. Sie können aber auch als ein Ergebnis anderer Methoden (Funktionen) erzeugt werden, die Daten oder Metadaten zurückliefern.

4.6.1.5 ActiveX Data Objects (ADO)

ADO stellt ein Sammlung von Schnittstellen über den OLE DB Daten dar. Während OLE DB eine Programmierschnittstelle auf Systemebene darstellt, ist ADO eine Programmierschnittstelle auf der Anwendungsebene, d.h. dass sie über der OLE DB Schicht sitzt. Im Vergleich zu anderen Datenbank Client-Technologien ist ADO noch relativ neu.

Die Vorteile von ADO im Vergleich zu anderen Datenbank Client-Technologien sind:

- ADO-Programmierung ist einfacher
- Neutralität im Bezug auf Programmiersprache (ADO kann z.B. mit Visual Basic, Java, C++, VBScript, JavaScript usw. eingesetzt werden)
- Provider-neutral (ADO kann auf Daten jeder OLE DB Quelle zugreifen)
- Erhaltung der OLE DB-Funktionalität (auf die unterliegenden OLE DB Schnittstelle kann zugegriffen werden)

ADO besteht aus sieben verschiedenen Objekten:

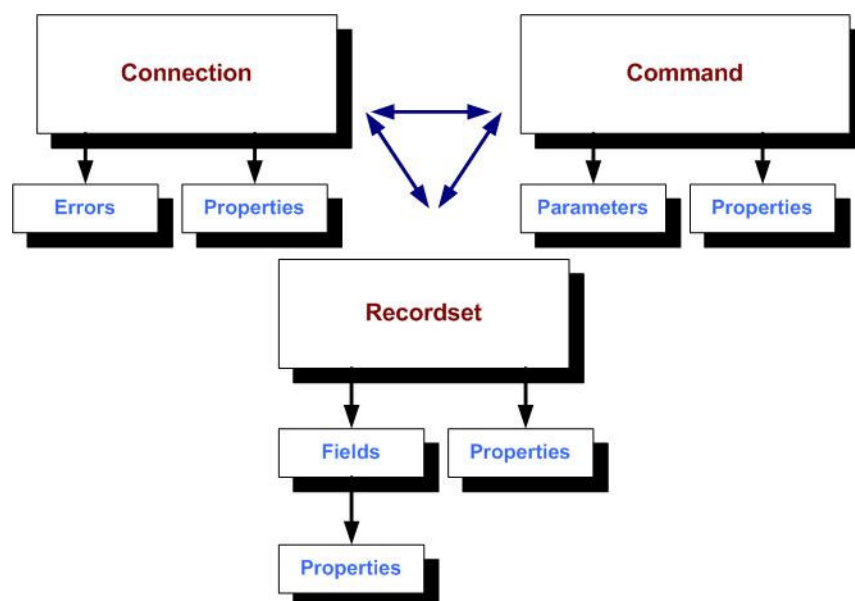


Abbildung 4.14 - ADO Objekt-Hierarchie

Das Connection, Command und Recordset Objekt sind Objekte der höheren Ebene. Sie können unabhängig von den anderen Objekten erzeugt und wieder gelöscht werden.

Obwohl das Parameter Objekt auch unabhängig von einem Command Objekt erzeugt werden kann, muss es jedoch zuerst mit einem Kommando in Verbindung gebracht werden, bevor es benutzt wird.

Das Field, Error und Property Objekt können nicht unabhängig von den Hauptobjekten erzeugt werden, sondern existieren nur in Verbindung mit einem dieser Elternobjekte.

Das ADO Connection Objekt beinhaltet die OLE DB Data Source und Session Objekte. Es repräsentiert ein einzelne Session mit der Datenquelle. Es definiert außerdem die Eigenschaften der Verbindung, weist den Bereich der lokalen Transaktionen zu, stellt eine zentrale Lokalität zur Fehlerabfrage bereit und ermöglicht die Ausführung von Schemaabfragen.

Das ADO Command Objekt setzt das OLE DB Command Objekt um. Es spezifiziert die Datendefinitions- oder Datenmanipulations-Statements, die ausgeführt werden sollen. Im Falle eines relationalen Datenproviders ist dies ein SQL-Statement. Dieses Objekt ermöglicht es Parameter zu spezifizieren und das Verhalten eines auszuführenden Statements anzupassen.

Das ADO Recordset Object setzt das OLE DB Rowset Objekt um. Es stellt die eigentliche Schnittstelle zu den Daten dar. Das Recordset Objekt ermöglicht die Kontrolle über den verwendeten Zugriffsmechanismus, den verwendeten Cursor-Typ, die Anzahl der Datensätze auf die gleichzeitig zugegriffen werden kann usw. Dieses Objekt zieht eine Sammlung von Field Objekten mit sich, welche die Metadaten über die Spalten eines Datensatzes (z.B. Name, Typ, usw.) wie auch die eigentlichen Datenwerte selber enthalten.

Jedes der drei ADO Hauptobjekte enthält ein Sammlung von Property Objekten. Das Property Objekt erlaubt ADO die Fähigkeiten eines spezifischen Providers zu durchleuchten. Da nicht alle Data Provider die gleiche Funktionalität anbieten, ist es für das ADO Modell sehr wichtig den Zugriff auf provider-spezifische Einstellungen auf einem dynamischen Art und Weise zu unterstützen.

Weitere Information zu ADO findet sich unter:

<http://msdn.microsoft.com/library/en-us/ado270/htm/mdaobj01.asp>

4.6.2 Vergleich der verschiedenen Datenbank Technologien

Für die Entwicklung einer Datenbank-Client-Anwendung stehen wie schon erwähnt verschiedene Technologien zur Verfügung, die untereinander folgende Abhängigkeiten aufweisen:

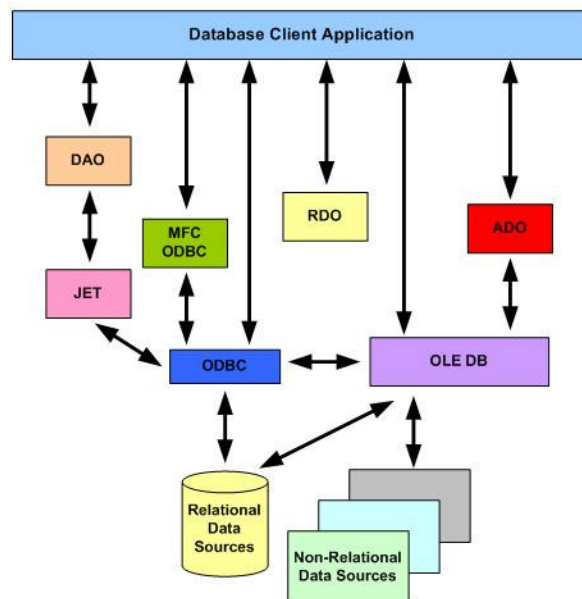


Abbildung 4.15 – Abhängigkeit der verschiedenen Datenbank Technologien

Die nachfolgende Tabelle fasst die Stärken bzw. Schwächen der einzelnen Datenbank Client-Technologien zusammen:

	ODBC	MFC/ ODBC	DAO	RDO	OLE DB	ADO
Object model	-	+	+	+	+	++
Non-relational data sources	-	-	-	-	+	+
Low-level control	+		-	-	+	
Code-to-functionality ratio	-		+		-	+
Performance	+		-		++	

Tabelle 4.3 – Stärken/Schwächen der Datenbank Technologien

Ein positives Vorzeichen (+) in der Tabelle entspricht einer Stärke, zwei (++) entsprechen einer besonderen Stärke und ein Minus (-) repräsentiert eine Schwäche der entsprechenden Technologie. Ist das Feld mit keinem Zeichen besetzt, so wird dieser Technologie in diesem Bereich keine besondere Schwäche oder Stärke zugeordnet.

4.6.3 Microsoft Universal Data Access

Microsoft Universal Data Access ist eine Plattform für die Entwicklung von Anwendungen, die Zugriff auf diverse relationale bzw. nicht-relationale Datenquellen benötigen. Universal Data Access besteht aus einer Sammlung von Software-Komponenten, die miteinander kommunizieren, indem sie gemeinsame Schnittstellen benutzen, die durch OLE DB definiert sind.

Folgende Abbildung demonstriert die Architektur von Universal Data Access:

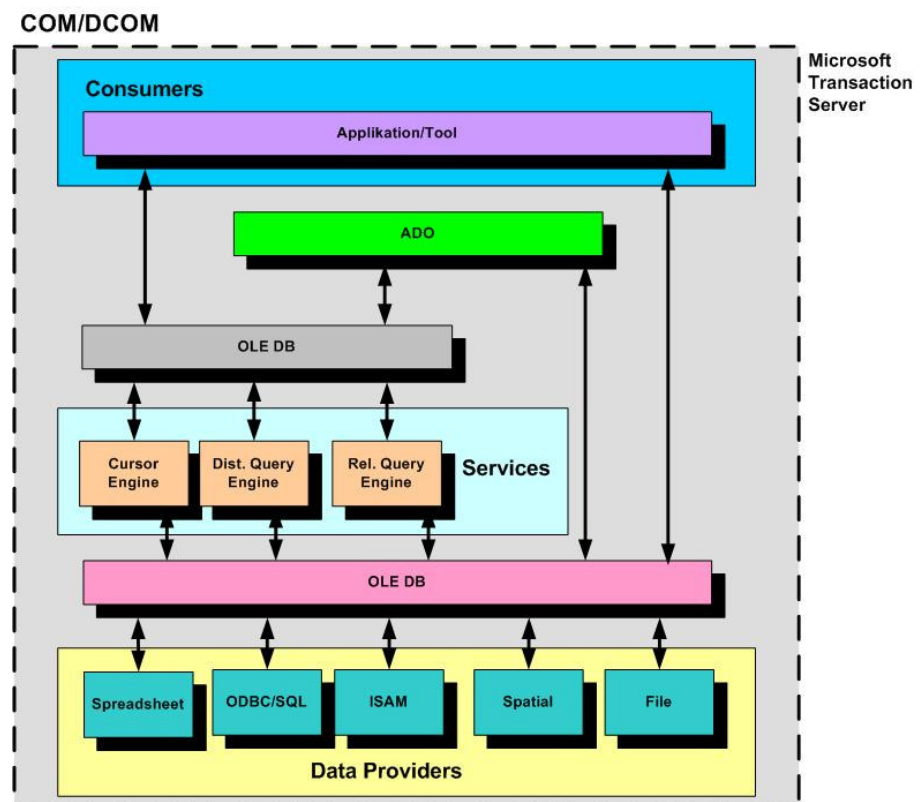


Abbildung 4.16 – Universal Data Access Architektur

Data Providers sind Komponenten, die diverse Datenquellen repräsentieren.

Services sind Komponenten, welche die Funktionalität der „Data Providers“ ausweiten, indem sie erweiterte Schnittstellen implementieren, die nicht standardmäßig vom Datenspeicher unterstützt werden.

Consumers sind Komponenten, die OLE DB Daten konsumieren.

Alle Interaktion zwischen den verschiedenen Komponenten, die in Abbildung 4.16 durch die dicken schwarzen Pfeile gekennzeichnet sind, können über Prozess- und Maschinengrenzen hinweg durch Netzwerkprotokolle ablaufen.

4.7 Datenbankapplikationen

4.7.1 Oracle

Oracle ist ein Datenbankmanagementsystem, das heute sehr weit verbreitet ist und vor allem seine Anwendung in administrativen Verwaltungsbereichen findet.

Oracle wurde mit dem Ziel entwickelt, Datenbanken mit wirklich großen Datenmengen (> 10 Gigabyte) manipulieren zu können, die mit einem herkömmlichen relationalen DBMS aufgrund von Problemen der effizienten Datenspeicherung und Geschwindigkeit zu großen Schwierigkeiten führen würde.

Mit Hilfe geeigneter Tools können bei Oracle kurze vordefinierte Transaktionen getätigt werden. Oracle Datenbanken können mit unterschiedlichen Datenbanksystemen und Betriebssystemen arbeiten. Verschiedene anwendungsbezogene Abfragesprachen ermöglichen umfassende Analysemöglichkeiten.

Der große Nachteil dieses System ist der sehr hohe Speicherbedarf.

4.7.2 Microsoft Access

Microsoft Access ist ein relationales Datenbanksystem der Firma Microsoft, das als Einzel- sowie auch als Mehrplatzsystem unter dem Betriebssystem Windows läuft.

Es erfüllt im wesentlichen alle Eigenschaften eines relationalen Datenbankmodells.

Es eignet sich besonders für Datenbanksysteme, die über die ODBC-Schnittstelle angesprochen werden können.

Tabellen stellen bei diesem System die eigentlichen Relationen dar. Sie werden lediglich einmal beim Erstellen der Datenbank definiert. Die Referenzen zwischen den Tabellen werden in Form von Beziehungen visualisiert.

Die wichtigsten Funktionen, die Access bereitstellt sind:

- Formulare
- Abfragen
- Berichte

Formulare definieren Eingabemasken, die das Erfassen und Updaten von Tabellendaten vereinfachen.

Die Abfragen erfolgen über die SELECT-Statements der SQL-Abfragesprache oder können menügesteuert abgesetzt werden.

Die Berichte, die erstellt werden können, fassen Tabelleninhalte und Abfrageresultate in formatierter Form zusammen.

Ein mächtiges Feature von MS Access sind die Assistenten-Funktionen. Sie stellen gerade für den weniger erfahrenen Benutzer eine große Unterstützung bei der Erstellung von allen Datenbank-Objekten von Access. Für den fortgeschrittenen Benutzer existieren eine SQL-Sicht der Abfragen und eine Schnittstelle zur VB-Programmierung.

Die Schemaentwurf bei MS Access erfolgt ebenfalls menügesteuert:



Abbildung 4.17 – Schemadefinition in Microsoft Access

MS Access bietet außerdem die Möglichkeit Routineaufgaben, die immer wiederkehren durch Makros zu erledigen.

Alle diese Bestandteile des Systems werden zentral als Microsoft Access-Datenbank in einer einzelnen Datei mit der Endung **.mdb** gespeichert.

5 Das System VisionMaster

5.1 Systemüberblick

VisionMaster ist ein System zur Oberflächeninspektion von Pressteilen im Automobilbau. Es ist ausschließlich dazu gedacht Aufgaben der Qualitätssicherung zu unterstützen und zu beschleunigen.

Mit Hilfe dieses Systems können im allgemeinen Oberflächen von Pressteilen in einer Zeit von ca. 20 Minuten analysiert, und die Ergebnisse der Inspektion protokolliert werden.

Das Gesamtsystem setzt sich aus folgenden Komponenten zusammen:

- Personalcomputer mit Frame Grabber Karte
- Kamera TV 2100
- Messtisch
- retro-reflektierende Leinwand
- VisionMaster-Software
- Microsoft Access Datenbank

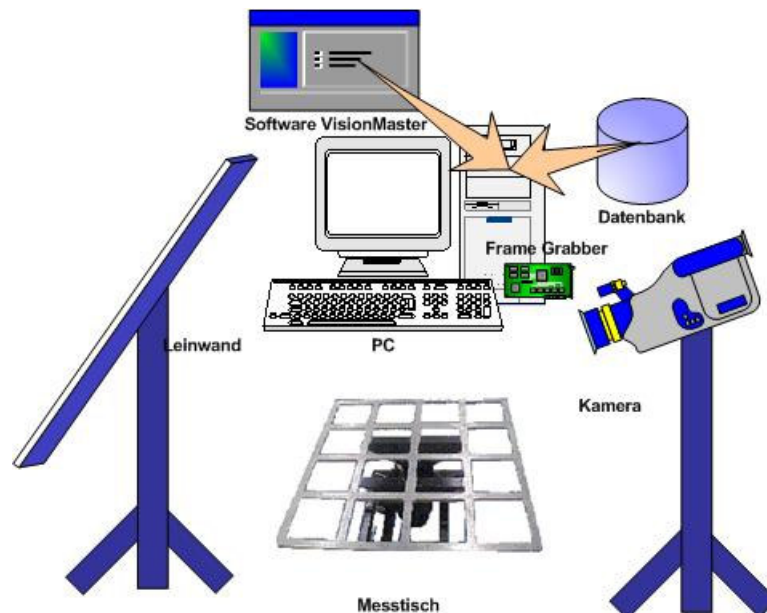


Abbildung 5.1 – Systemkomponenten von VisionMaster

Das Prinzip der Analyse basiert auf den Vergleich von zwei Grauwertbildern:

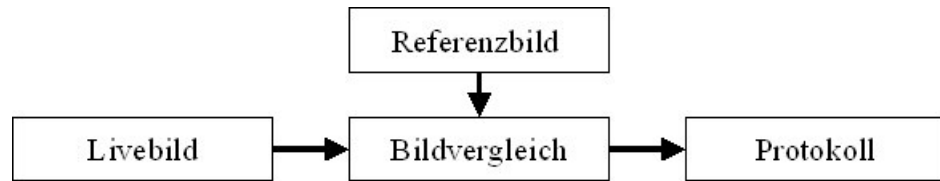


Abbildung 5.2 – Analyseprinzip von VisionMaster

Das aktuelle Bild des zu prüfenden Autoteils („Livebild“), das von der Kamera aufgenommen wird, wird mit einem vorher aufgenommenen Referenzbild¹⁾ des gleichen Teils verglichen, welches in der Datenbank gespeichert ist.

Die Auswertung der beiden Bilder erfolgt durch den Auswertungsalgorithmus PLOT, der die Grauwertistogramme der beiden Bilder erzeugt und gegenüberstellt.

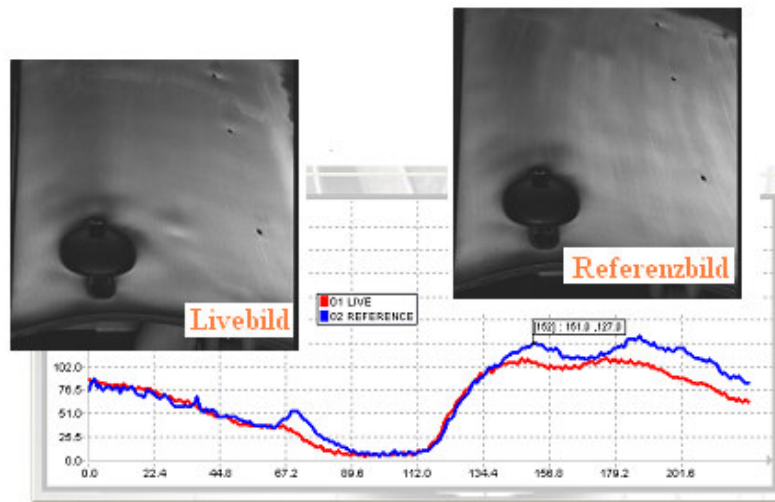


Abbildung 5.3 – Das Prinzip des Auswertungsalgorithmus PLOT

Zur Unterstützung der Inspektion ist das System VisionMaster an eine Datenbank angebunden. Die Datenbank enthält im allgemeinen Informationen, die für die Analyse eines Pressteils notwendig sind (z.B. Modelldaten, Pressteildaten, Systemparameter, Initialisierungswerte, Pressedaten usw.) und bietet außerdem die Möglichkeit zur Archivierung von Inspektionsergebnissen (Koordinaten der Messkurven, zugehörige Presseparameter usw.).

Nach der Analyse des aktuellen Bildes wird ein Protokoll generiert, indem alle relevanten Informationen der Inspektion enthalten sind.

¹⁾ Die Beurteilung eines Autoteils erfolgt immer in Bezug auf das zugehörige Referenzteil. Aus diesem Grund sollte bei der Aufnahme des Referenzbildes ein Autoteil ausgewählt werden, das möglichst keine Defekte aufweist.

Die Aufnahme der Grauwertbilder basiert auf dem Prinzip der Retro-Reflexion:

Die von der Kamera ausgesendeten Lichtstrahlen treffen auf das Messobjekt und werden von dessen Oberfläche reflektiert. Die reflektierten Lichtstrahlen erreichen die Leinwand und werden von dieser ebenfalls reflektiert. Über die Oberfläche des Objekts gelangen sie wieder zur Kamera, wo sie erfasst werden.

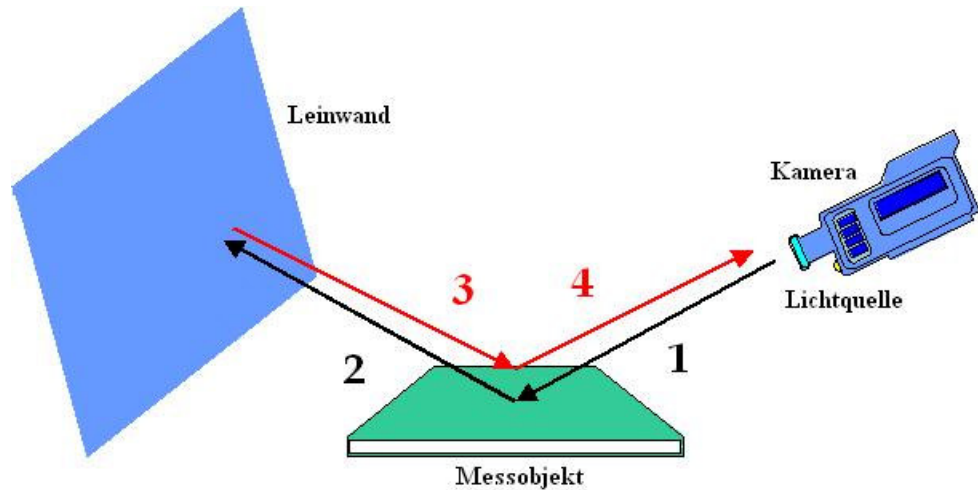


Abbildung 5.4 – Prinzip der Retro-Reflexion

Die Leinwand besitzt natürlich eine spezielle Beschichtung, die Lichtstrahlen reflektiert. Damit die Lichtstrahlen von der Oberfläche des Messobjektes abgelenkt werden können, muss eine spezielle Flüssigkeit (Glänzer¹) auf dessen Oberfläche aufgetragen werden.

Die VisionMaster-Software stellt für den Benutzer (Inspektor) eine komfortable Schnittstelle zur Datenbank und Kamera dar. Die wichtigsten Funktionen sind:

<p>1. Datenbankverwaltung Export/Import von MS Access Datenbanken und Erstellung/Einspielung von Backups</p>	<p>2. Datenbankverwaltung Einspielen/Ändern/Löschen von Referenzdaten</p>
<p>3. Bildaufnahme Tools zur Einstellung der Kamera (Focus, Iris, Zoom)</p>	<p>4. Datenbankrecherche Auflistung von Protokollen nach bestimmten Suchkriterien</p>
<p>5. Sonderinspektionen Vergleich verschiedener Produktionswochen bzw. Produktionsstadien</p>	<p>6. Erstellung von Protokollen</p>

Tabelle 5.1 – Die wichtigsten Aufgaben der VisionMaster-Software

¹) spezielle Flüssigkeit die Lichtstrahlen reflektiert

5.2 Hardwarekomponenten

5.2.1 Der VisionMaster Rechner

Der im System integrierte Rechner ist ein herkömmlicher Personalcomputer, der die Anforderungen an Hard- und Software erfüllt

- Pentium IV Prozessor,1,80GHz
- 256 MB Arbeitsspeicher
- zwei Festplatten mit je 40 GByte Kapazität
- Grafikkarte
- 21 Zoll Farbmonitor
- Diskettenlaufwerk
- Tastatur und Maus
- CD RW 32x10x40x;
- DVD Rom 16x
- zwei serielle Schnittstellen
- parallele Schnittstelle
- USB-Schnittstelle
- 3Com Etherlink XL 10/100 Netzwerkkarte

Zur Entwicklung des Software-Systems **VisionMaster** wurde der Rechner mit folgender Software ausgestattet:

- Microsoft Windows XP Professional 2002
- Visual C++.NET
- Intel Image Processing Library (IPL)
- Microsoft Access 2000

5.2.2 Frame Grabber PX510

Die Imagenation PX Familie an Frame Grabber Karten unterstützt die Bildaufnahme für Anwendungen, die eine hohe Genauigkeit und Fehlerfreiheit im Graustufenbereich voraussetzen. Das Design der PX-Hardware gewährleistet diese Genauigkeit und zeichnet die PX Frame Grabber Karten gerade für den Einsatz im Industrie- und Forschungsbereich aus.

Die wesentlichen Merkmale der PX-Hardware werden in der nachfolgenden Tabelle aufgelistet:

Video Eingansformat	:	Monochrom, RS-170 (NTSC) oder CCIR/PAL
Video Eingangssignal	:	1 V, 75 Ohm
Auflösung	:	640 x 480 Pixel (RS-170) bzw. 768 x 576 Pixel (CCIR/PAL) - 256 Graustufen (8 Bits)
Sampling Jitter	:	max. ± 3 ns
Aufnahmezeit	:	1/30 s/Frame (RS-170 (NTSC)), 1/25 s/Frame (CCIR/PAL)
LUTs	:	programmierbar - 256 Byte
Schutzvorkehrungen	:	alle Ein-/Ausgänge haben Schutzdioden
Form	:	PCI short card: 174.6 x 106.7 mm 6.875 x 4.2 in. PC/104 Plus Module: 91.4 x 96.5 mm 3.4 x 3.6 in. Compact PCI Module: 100 x 160 mm 3.94 x 6.3 i
Verzerrung (Video)	:	≤ 0.7 LSB
Versorgung	:	+5 VDC, 650 mA
Video Multiplexer	:	4 Video-Eingänge - RS-170 (NTSC) - CCIR/PAL
Betriebstemperatur	:	0° C to 60° C.

Tabelle 5.2 – Spezifikation der Frame Grabber Karte PX510

Alle angegebenen Merkmale wurden auf einem PCI Bus für den Realzeitbetrieb konzipiert.



Abbildung 5.5 – Frame Grabber PX510

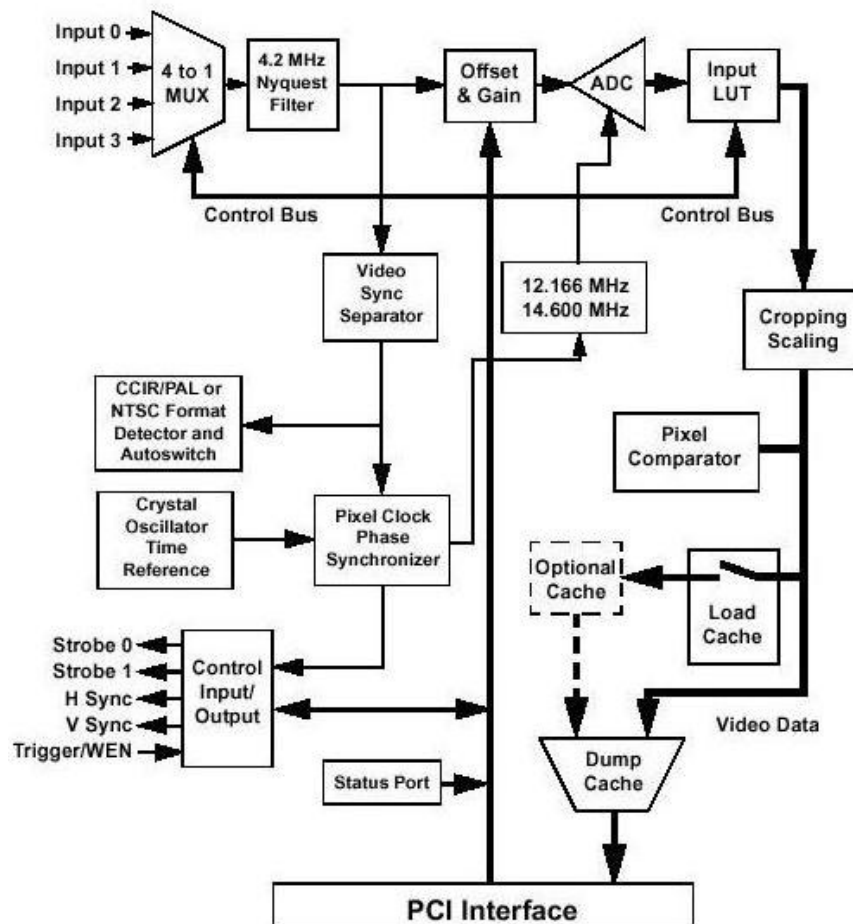


Abbildung 5.6 – Blockschaltbild der Frame Grabber Karte

Weitere Informationen finden sich im Benutzerhandbuch der Frame Grabber Karte (siehe *Imagination PX Family of Precision Frame Grabbers User's Guide for the PX510 and PX610*).

5.2.3 Kamera TVA 2100

Beim Kameramodel TVA 2100 handelt sich um ein solides CCD-basiertes Gerät mit einer Auflösung von 754 x 488 Pixel und einer integrierten Lichtquelle.



Die Lichtquelle dient zur Optimierung der Kameraempfindlichkeit bezogen auf verschiedene Oberflächentypen.

Die Kamera wird auf einem speziellen Stativ montiert, um so eine entsprechend genaue Ausrichtung und Einstellung der Kamera bezogen auf den Anwendungsbereich zu unterstützen.

Beim Kamerastativ handelt es sich um ein Zwei-Achsen-Gerät mit Sperrvorrichtungen.

Eine Achse erlaubt die vertikale Rotation der Kamera in 3,75 Grad Schritten, die andere Achse ist für die lineare vertikale Ausrichtung der Kamera verantwortlich.

Abmessungen (L x B x H)	: 53,3 cm x 12,7 cm x 15,2 cm
Gewicht (ohne/mit Stativ)	6,8 kg/ 9,1kg
Datenrate	: 30 Frames/s
Power Input	: 115V AC @ 60 Hz, max. 1 A
Ambientes Licht	: 0 bis 400 Lux (empfohlen 0-150 Lux)
Lebensdauer der Lampen	: ca. 2000 Stunden

Tabelle 5.3 – Spezifikation der Kamera

Es wird empfohlen die Kamera in einem Abstand zwischen 2,0 - 2,7 m vom Zentrum des zu untersuchenden Objektes zu positionieren. Bei einem kleineren Abstand zwischen Kamera und Objekt wird die Distanz zwischen Kameralinsen und Lampen um einen bestimmten Faktor verfälscht. Wird die Kamera zu weit vom Objekt aufgestellt, wird die Auflösung beeinträchtigt.

Für die meisten Oberflächeninspektionen sollte die Kamera auf einen Winkel von ungefähr 23 Grad fixiert werden. Im allgemeinen können jedoch Winkel zwischen 20 und 60 Grad je nach Bedarf eingestellt werden.

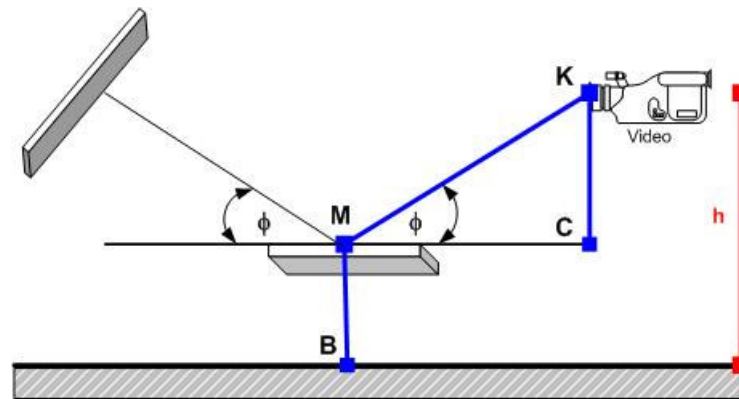


Abbildung 5.7 – Berechnung der Kamerahöhe

Die Höheneinstellung der Kamera hängt vom Abstand d der Kamera zum Objekt, wie vom Kamerawinkel ϕ ab.

$$(1) \quad d = \overline{MC}$$

$$(2) \quad h = \overline{BM} + \overline{CK} = \overline{BM} + (\tan(\phi) * \overline{MC})$$

5.2.4 Der Highlighter

Wie schon zu Anfang dieses Kapitels erwähnt wurde, basiert die Aufnahme der Grauwertbilder auf dem Prinzip der Lichtreflexion. Deswegen ist es zwingend notwendig, dass die Oberfläche des zu vermessenden Objektes ein Spiegelverhalten aufweist, d.h. die einfallenden Lichtstrahlen zurückwirft.

Aus diesem Grund wird das Objekt mit einer speziellen Flüssigkeit (Highlighter¹⁾) überzogen, die eine Reflexion der Lichtstrahlen ermöglicht.

Die Reflexion der einfallenden Lichtstrahlen durch das Objekt beeinträchtigt erheblich die Genauigkeit des Systems. Ein korrektes Auftragen dieser Flüssigkeit ist deshalb von sehr großer Bedeutung:

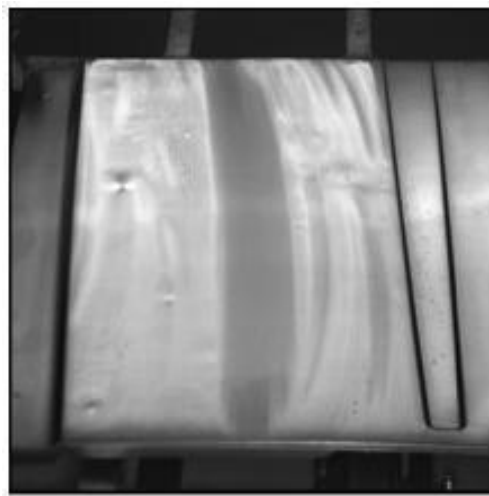


Abbildung 5.8 - zu wenig Highlighter

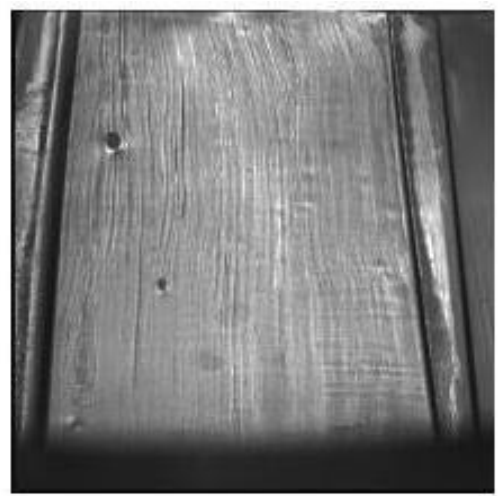


Abbildung 5.9 - zu viel Highlighter

Zu wenig oder unregelmäßig aufgetragene Glanzflüssigkeit ist durch dunkle Flecken auf der Oberfläche erkennbar. Im Gegensatz dazu bilden sich bei zu viel aufgetragener Flüssigkeit lange Streifen an der Oberfläche und können vom ungeübten Beobachter fälschlicherweise als Defekte interpretiert werden.

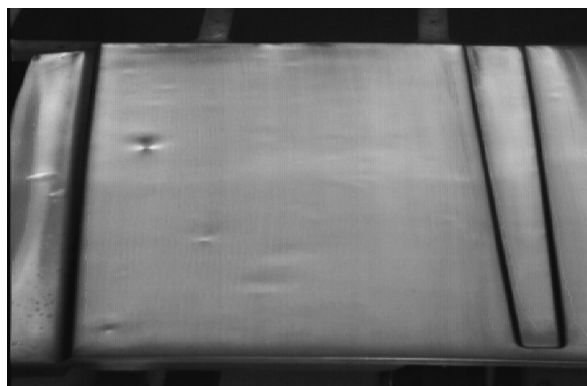


Abbildung 5.10 – korrekte Auftragung der Glanzflüssigkeit

¹⁾ licht- reflektierende Flüssigkeit

Da wir uns in unserem Fall auf die Analyse von Metalloberflächen beschränken, wird der Metal Highlighter M110 (PN 50200850) eingesetzt.

Für die Inspektion anderer Materiale wie z.B. Plastik sollten andere geeignetere Highlighter gewählt werden (z.B. PN 50200852).

Bevor die Flüssigkeit auf der Oberfläche des Objekts aufgetragen wird, sollte diese von jeglichem Staub und Schmutz gereinigt werden.

Für das Auftragen sollten ein sauberes, staubfreies Tuch oder Papierstreifen verwendet werden.

Im Normalfall wird die Flüssigkeit von Vorne nach Hinten bzw. von Hinten nach Vorne aufgetragen.

Nach dem Auftragen sollte das Teil einige Minuten (3-10) ruhen, damit die Flüssigkeit einwirken kann.

Normalerweise stehen für die Inspektion eines Teiles mindestens 20 Minuten zur Verfügung, bevor die reflektierende Eigenschaft der aufgetragenen Flüssigkeit langsam beginnt an Wirkung zu verlieren.

5.2.5 Der Messtisch

Der Messtisch ist ein Zwei-Achsen-Gerät, das auf beiden horizontalen Achsen bis ± 45 Grad (in $7\frac{1}{2}$ Grad-Schritten) rotieren kann.

Auf der Oberflächenplatte ($60'' \times 60'' \times \frac{3}{4}''$) des Tisches können spezielle Halterungen für die Befestigung der zu vermessenden Teile angebracht werden.

Für die Befestigung der Halterungen auf der Tischoberfläche werden $\frac{1}{4}''$ Schrauben verwendet.

Der Tisch hat ein Gewicht von 227 kg, besitzt vier Laufrollen, ist auf mehreren Höhenstufen verstellbar und kann auf verschiedene Grundpositionen ausgerichtet werden. Seine Höhe beträgt $26\frac{11}{16}''$ (mit Laufrollen).

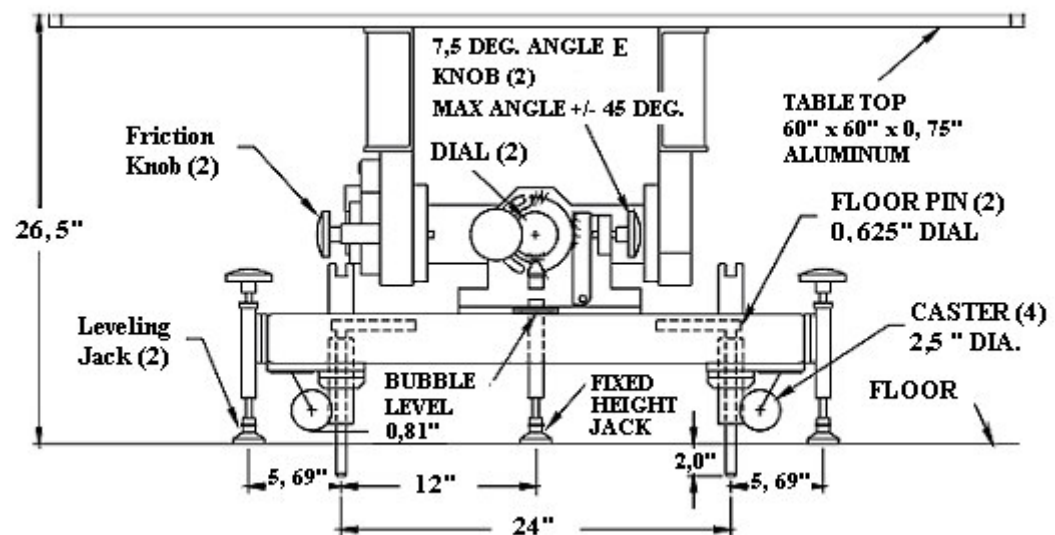


Abbildung 5.11 –Teile und Abmessungen des Messtisches

Die Tischbasis hat zwei Bolzen von $\frac{5}{8}''$ Durchmesser, die dazu benutzt werden um den Tisch auf dem Boden in extra dafür eingerichtete Bodenlöcher zu befestigen.

Abhängig von der Position des Messtisches und der Platzierung des zu vermessenden Objektes, gelten einige Einschränkungen für die Tragkraft des Tisches:

1. Die Last des Objektes sollte nicht über 18 kg liegen, wenn sie auf der linken unteren Seite der Tragfläche liegt und der Tisch auf den max. Drehwinkel von 45 Grad steht.
2. Sie sollte auch nicht über 28 kg liegen, wenn das Objekt in der Mitte der Tragfläche steht und der Tisch auf den max. Drehwinkel von 45 Grad steht.
3. Niemals sollte eine Last über 90 kg auf dem Tisch platziert werden, wenn dieser sich in seiner Grundposition befindet.
4. Die max. Last am Rand der Tragfläche des Tisches sollte nicht mehr als 45 kg betragen, wenn dieser sich in der Grundposition befindet.

Um eine reproduzierbare geometrische Position für den Messtisch bzw. der Kamera zu gewährleisten wird ein Positionierungsgitter eingerichtet, indem Löcher in einer bestimmten Anordnung in den Boden gebohrt werden.

Folgende Anordnung sollte bei der Einrichtung des Positionierungsgitters gewählt werden:

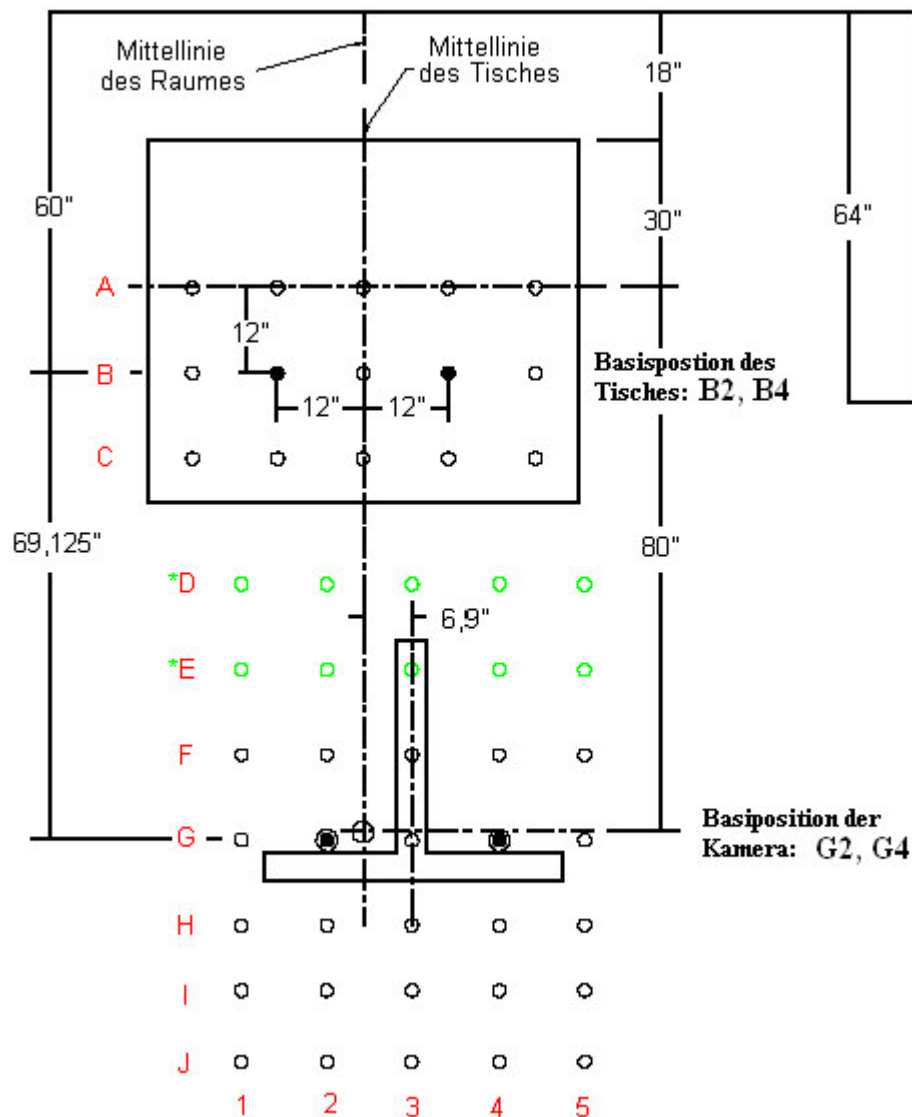


Abbildung 5.12 – Positionierungsgitter für Messtisch und Kamera

Das Gitter A1-C5 bezieht sich auf die Positionierung des Messtisches während das Gitter F1-J5 für die Kamera gedacht ist.

Das zweite Gitter kann bei Bedarf um zwei Reihen erweitert werden (Reihe D, E). Die Löcher müssen parallel zueinander in den Boden gebohrt werden.

Der Messtisch wird so angebracht, dass die Vorderseite des Tisches auf die Kamera gerichtet ist.

5.3 Softwarestruktur

Bei dem Software-System VisionMaster handelt es sich um eine MFC-Anwendung. Das System wurde in Visual C++ mit der Entwicklungsumgebung Visual.NET programmiert. Bei der Entwicklung der Software wurde gezielt darauf geachtet das Programm modular zu halten, um eine und Erweiterbarkeit, Wiederverwendbarkeit und Austauschbarkeit der Komponenten zu ermöglichen. Jede funktionale Einheit des Programms (z.B. Auswertemodul, Eingabemasken, DB-Schnittstelle usw.) wurde als separates Visual C++ Projekt entwickelt. Vielmehr wurde bis auf das Hauptmodul jedes Modul als erweiterbare MFC-DLL implementiert.

5.3.1 Grobarchitektur des Systems

Das Gesamtsystem gliedert sich in drei Modulschichten:

- **Benutzeroberfläche / GUI**

Diese Schicht stellt die graphische Benutzeroberfläche dar und ist die Schnittstelle zwischen dem Anwender und dem System.

- **Anwendungsschicht**

Die Funktionen in dieser Ebenen rufen eine oder mehrere Funktionen in einer oder mehreren Modulen der Ausführungsschicht auf, und übergeben ihr die Daten. Nach Veränderung der Daten werden in dieser Schicht die Funktionen zur Aktualisierung der GUI aufgerufen.

- **Ausführungsschicht**

Diese Schicht ist die Schnittstelle zwischen der Anwendungsschicht und den Daten. Sie organisiert den Zugriff auf die Datenbank und die Kamera. Um den Zugriff auf die Daten zu schützen, müssen alle oberen Schichten mit dieser Schicht über die fest definierten Schnittstellen kommunizieren, denn nur die in dieser Schicht befindlichen Funktionen können direkt auf die Datenbank zugreifen.



Abbildung 5.13 – Grobarchitektur des Systems VisionMaster

5.3.2 Feinarchitektur

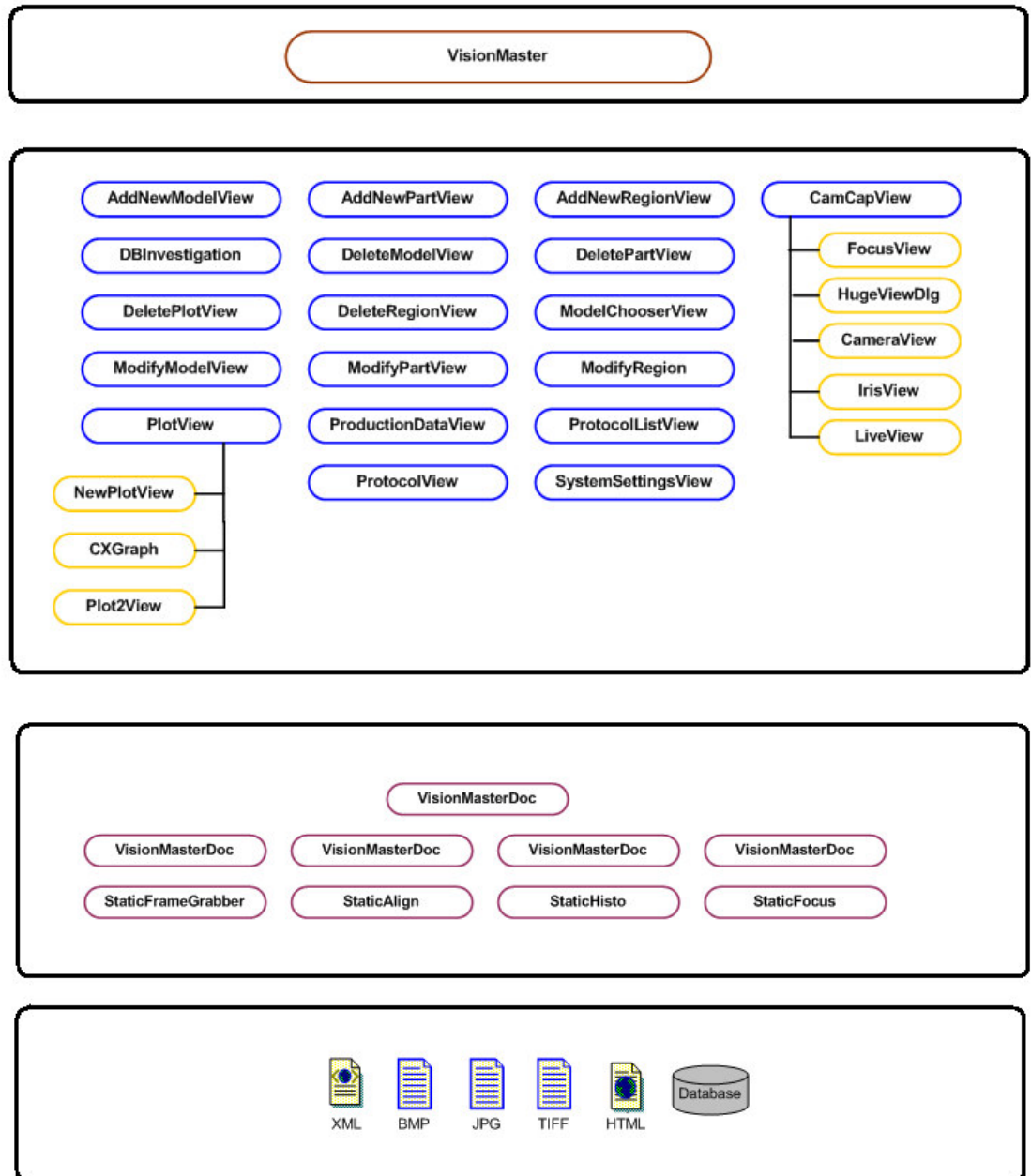


Abbildung 5.14 - Feinarchitektur des Systems VisionMaster

VisionMaster

Das Modul VisionMaster stellt das Hauptprogramm dar und bietet die Funktionalität, die von den anderen Modulen benötigt wird. Über die Menüleiste bzw. der Symbolleiste lassen sich die anderen Module aufrufen und anzeigen. Am rechten Bildschirmrand liefert das Informationszentrum Auskünfte über den aktuellen Zustand der wichtigsten Umgebungsvariablen, so z.B. die Anzahl der Modelle in der Datenbank, oder die aktuelle Auswahl der Region eines Autoteils. Somit hat der Benutzer alle relevanten Informationen auf einen Blick und hat jederzeit die Möglichkeit, evtl. vorhandene Falscheingaben zu korrigieren.

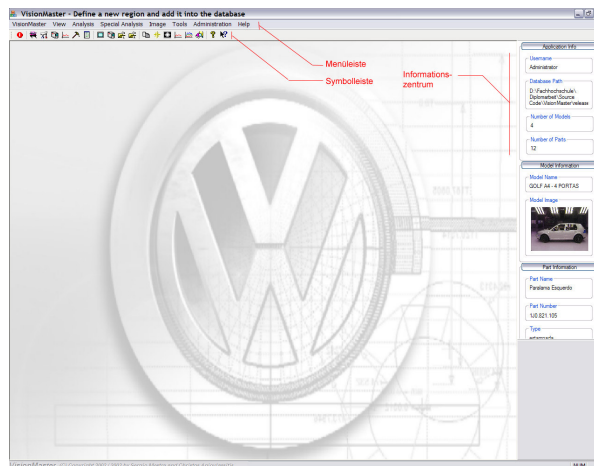


Abbildung 5.15 - Die Oberfläche des Hauptprogramms

AddNewModelView

Benutzeroberfläche für das Einfügen eines neuen Automodells in die Datenbank. Der Zugriff auf die Datenbank erfolgt über Funktionen des Dokumentes VisionMasterDoc, das wiederum über DbInterface auf die Datenbank zugreift. In diesem Modul befinden sich auch Funktionen, die die Verzeichnisstruktur für das neu eingefügte Modell erstellen.

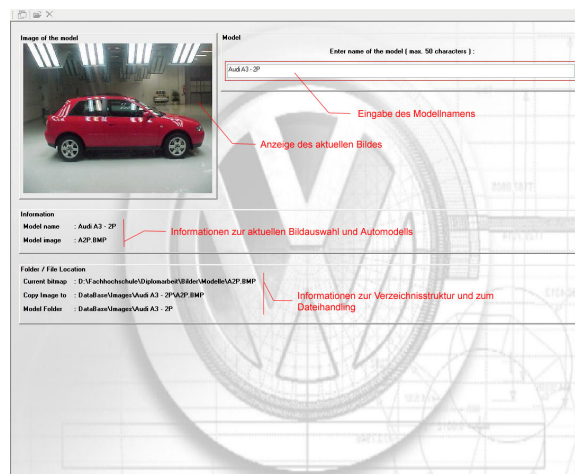


Abbildung 5.16 – Der Dialog AddNewModelView

AddNewPartView

Benutzeroberfläche für das Einfügen eines neuen Autoteils in die Datenbank. Nach Auswahl eines oder mehrerer Automodelle ist es möglich die Informationen des Autoteils einzugeben, wie z.B. die Bezeichnung und die Teilenummer.

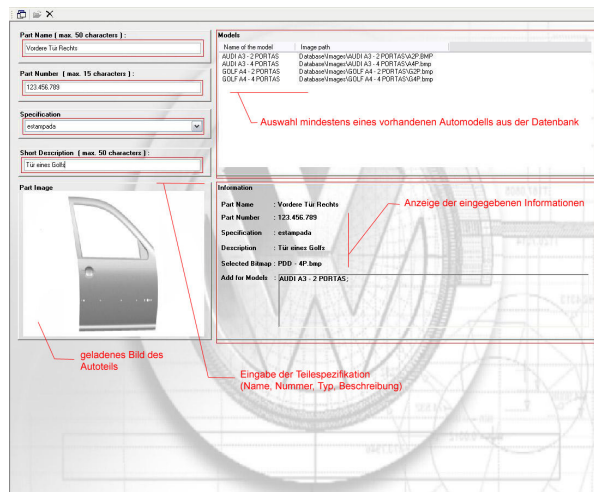


Abbildung 5.17 – Der Dialog AddNewPartView

AddNewRegionView

Dieses Modul bietet die Möglichkeit mit Hilfe eines Assistenten eine neue Region innerhalb eines Autoteils zu definieren. Es erfordert zuerst die Auswahl eines Automodells und die eines dazugehörigen Autoteils.

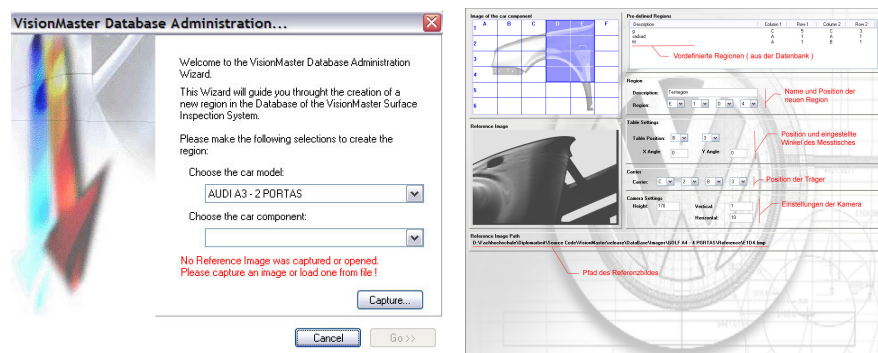


Abbildung 5.18 – Der Dialog AddNewRegionView

Nachdem diese Angaben gemacht worden sind, muss der Benutzer ein Bild dieser Region aufnehmen. Dazu drückt er mit Hilfe der Maus auf den Knopf „Capture“ und dann auf „Capture Reference Image“.

Danach gelangt er automatisch zur Benutzeroberfläche, bei der er das Einfügen einer neuen Region in die Datenbank durchführen kann.

CamCapView

Dieses Modul stellt sämtliche Benutzerschnittstellen zur Verfügung, die mit der Aufnahme eines Bildes, bzw. mit deren Manipulation zu tun hat. CamCapView bietet dem Benutzer die Möglichkeit, anhand von geeigneten Algorithmen und eines Referenzbildes das aktuelle Livebild so auszurichten, das es mit dem Referenzbild soweit wie möglich übereinstimmt.

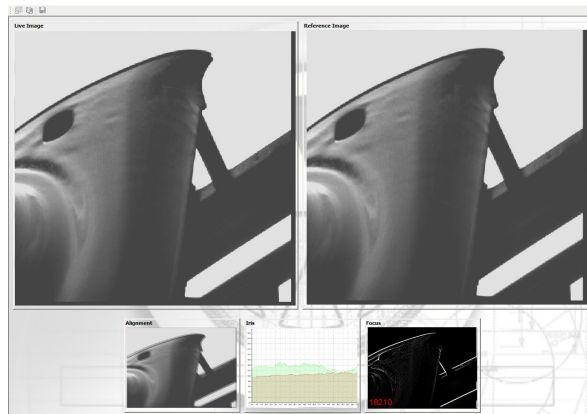


Abbildung 5.19 – Der Dialog CamCapView

Das Alignment Fenster zeigt das Livebild mit einer Deckkraft von 100% während das Referenzbild mit einer Deckkraft von 70% angezeigt wird. Somit kann man erkennen, ob das zu analysierende Objekt sich an der gleichen Position befindet, wie es bei der Aufnahme des Referenzobjektes der Fall war.

Das Iris Fenster Hilft dem Benutzer die gleichen Lichtverhältnisse herzustellen, wie sie bei der Aufnahme des Referenzbildes vorhanden waren, indem das Grauwert histogramm der beiden Bilder angezeigt und der Mittelwert berechnet wird. Liegt der Mittelwert des aktuellen Bildes unter dem des Referenzbildes, bedeutet dies, dass das Bild zu dunkel ist. Umgekehrt ist ein höherer Mittelwert ein Anzeichen dafür, dass das Bild heller als das Referenzbild ist.

Das Fokus Fenster soll dem Benutzer anzeigen, wann die Kamerafokussierung ihr Optimum erreicht hat. Durch Anwendung des Sobel Filters, sowohl in vertikaler als auch in horizontaler Ausrichtung, wird ein Bild geschaffen, das die Kanten des Objektes weiß darstellt, während das restliche Bild schwarz gefärbt wird. Nun wird die Summe über sämtliche Bildpunkte berechnet und angezeigt. Hat diese Summe ihr Maximum erreicht, so ist der Fokus der Kamera optimal eingestellt.

DBInvestigation

Um auf die gespeicherten Messdaten zugreifen zu können und diese noch einmal zu visualisieren wurde das Modul DBInvestigation erstellt. Anhand verschiedener Filter lassen sich so gezielt die Informationen extrahieren, die der Benutzer sehen möchte.

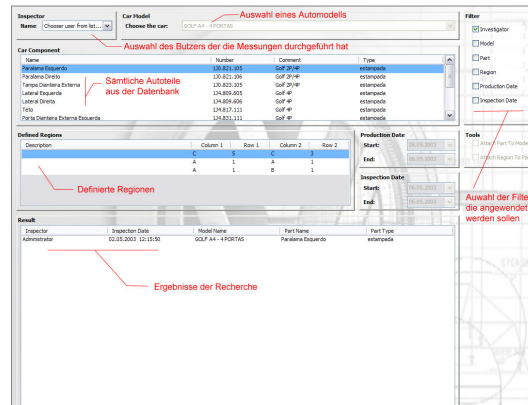


Abbildung 5.20 – Der Dialog DBInvestigation

DeleteModelView

Benutzeroberfläche für das Löschen eines vorhandenen Automodells aus der Datenbank. Nach der Auswahl des Automodells aus der Combobox wird durch Drücken des „Delete Model from Database“ Buttons das Modell vollständig gelöscht. Die Verzeichnisstruktur und sämtliche Dateien, die diesem Modell zugeordnet sind, werden ebenfalls gelöscht.

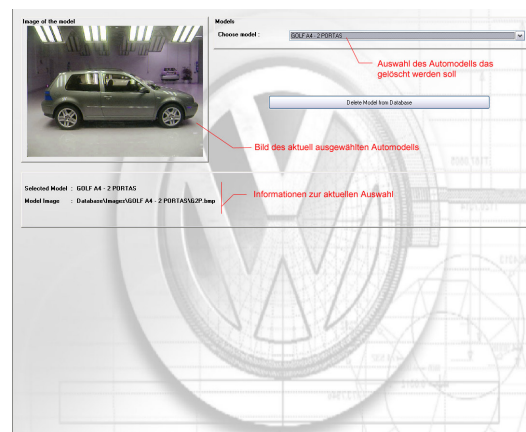


Abbildung 5.21 – Der Dialog DeleteModelView

DeletePartView

Benutzeroberfläche für das Löschen eines vorhandenen Autoteils aus der Datenbank. Nach der Auswahl des Automodells werden die dazugehörigen Autoteile in einer Liste angezeigt. Beim Löschen eines Autoteils, das zu mehreren Modellen gehört wird eine Infobox angezeigt, die darauf hinweist, dass sämtliche Verbindungen zu den Modellen gelöscht werden.

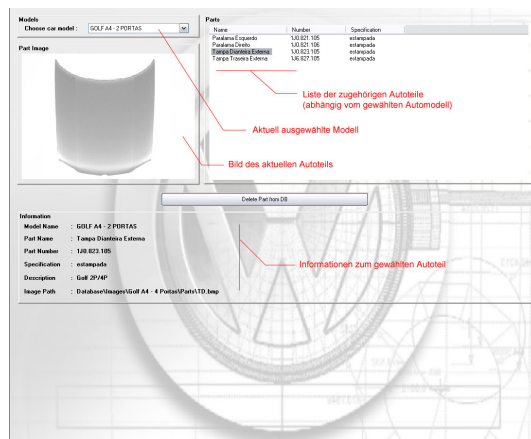


Abbildung 5.22 – Der Dialog DeletePartView

DeletePlotView

Mit Hilfe eines Assistenten werden das Automodell, das Autoteil und die entsprechende Region ausgewählt. Erst danach gelangt man zu der Oberfläche, in der die definierten Plots angezeigt werden. Hier können ein oder mehrere Plots ausgewählt und danach gelöscht werden. Bei der Auswahl eines Plots wird deren Kurve in das Referenzbild eingezeichnet und das dazugehörige Diagramm angezeigt.

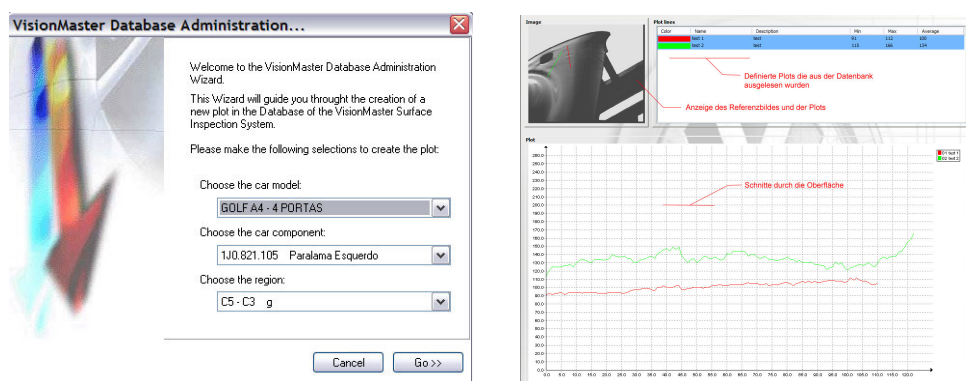


Abbildung 5.23 - Der Dialog DeletePlotView

DeleteRegionView

Benutzeroberfläche für das Löschen einer vorhandenen Region aus der Datenbank bzw. eines bestimmten Parametertupels, abhängig von einem Teil und einer Region. Wird das Kontrollkästchen „Delete Region completely“ aktiviert, so werden in der Combobox der Regionen sämtliche Regionen angezeigt, unabhängig vom Autoteil. Somit können auch Regionen gelöscht werden, die mehrfach für unterschiedliche Autoteile definiert wurden.

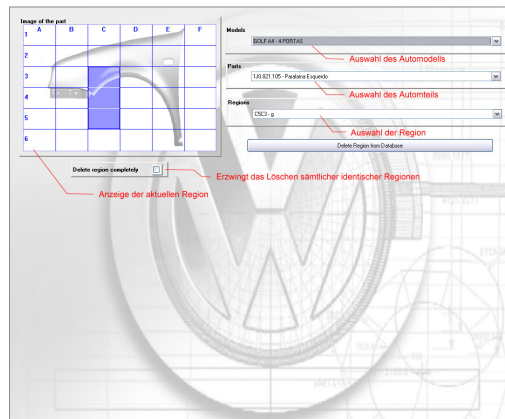


Abbildung 5.24 - Der Dialog DeleteRegionView

ModelChooserView

Hier kann der Benutzer dem System mitteilen welches Autoteil und welche Region analysiert werden soll. Dazu stehen ihm die Auswahlbox mit sämtlichen Automodellen zur Verfügung sowie die Listen der Autoteile und der definierten Regionen.

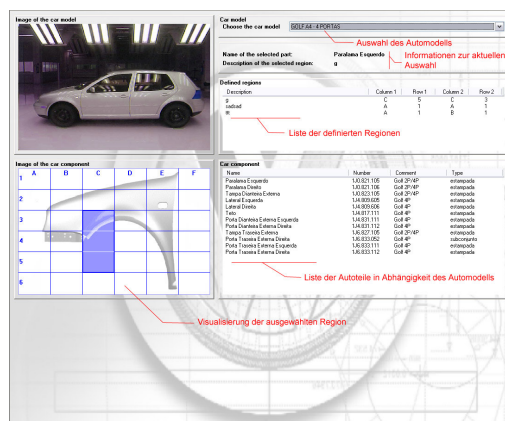


Abbildung 5.25 - Der Dialog ModelChooserView

ModifyModelView

Hier kann der Benutzer nachträglich den Namen des Automodells und das Bild ändern, bzw. das Bild des Modells löschen.

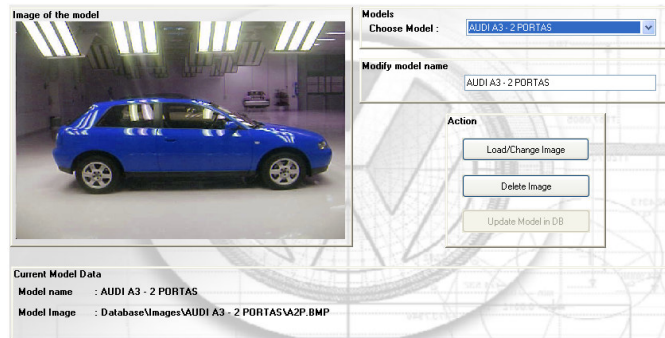


Abbildung 5.26 Der Dialog ModifyModelView

ModifyPartView

Benutzeroberfläche für das Ändern eines vorhandenen Autoteils in der Datenbank. Sämtliche Eingaben die vorher vorgenommen wurden können hier geändert werden, nicht aber der Bezug zu den Automodellen. Um dies zu Ändern muss das Autoteil aus der Datenbank gelöscht und neu angelegt werden.

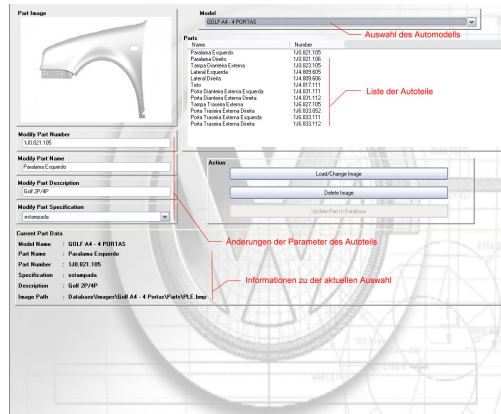


Abbildung 5.27 Der Dialog ModifyPartView

ModifyRegionView

Benutzeroberfläche für das Ändern eines vorhandenen Parametertupels, abhängig von einem Autoteil und einer Region. Regionen können nur in Abhängigkeit von einem Autoteil modifiziert werden; sie können nicht für die Allgemeinheit modifiziert werden.

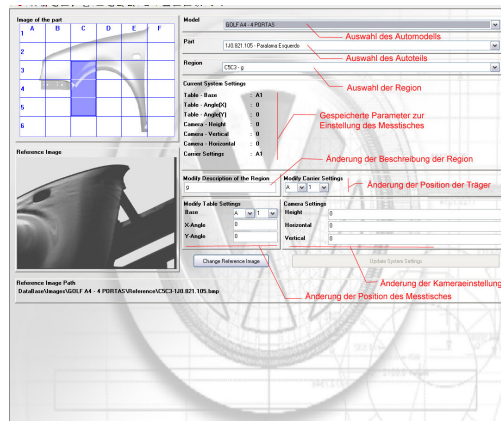


Abbildung 5.28 Der Dialog ModifyRegionView

PlotView

Mit Hilfe des Plot Tools ist es möglich Schnitte durch ein aufgenommenes Bild zu legen und diese dann im Schaubild anzuzeigen. Durch Auswahl des Menüs „Create new plot...“ hat der Benutzer die Möglichkeit einen neuen Schnitt zu erzeugen. Nach der Eingabe des Namens, einer Beschreibung und der Farbe muss der Anwender mit der linken Maustaste auf das Bild klicken und die Maus bis zum Endpunkt ziehen. Erst danach kann die Maustaste losgelassen werden.

Im Schaubild erscheint der dazugehörige Graph.

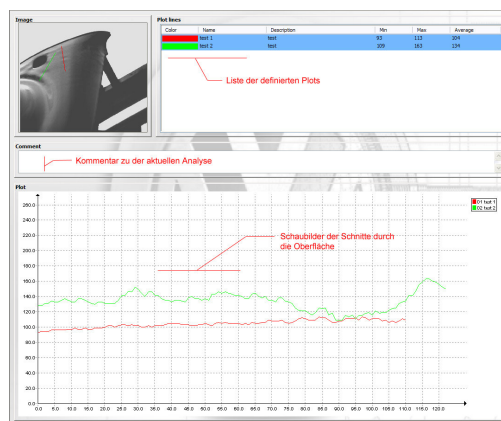


Abbildung 5.29 Der Dialog PlotView

ProductionDataView

Benutzeroberfläche für die Eingabe der Presseparameter in die Datenbank. Da es sehr viele Parameter sind, werden standardmäßig alle Felder mit Null initialisiert. Der einzig wichtige Parameter ist das Produktionsdatum des Autoteils. Aus diesem Grund muss dieser immer eingegeben werden.

Abbildung 5.30 Der Dialog ProductionDataView

ProtocolView und ProtocolListView

Diese Module bieten die Möglichkeit Protokolle darzustellen, sie im HTML Format abzuspeichern und auszudrucken, sowie die gewonnenen Messdaten in die Datenbank abzulegen, um sie später wieder aufrufen zu können. ProtocolListView bietet zusätzlich Navigationsschaltflächen um zwischen den Protokollen hin und her navigieren zu können.

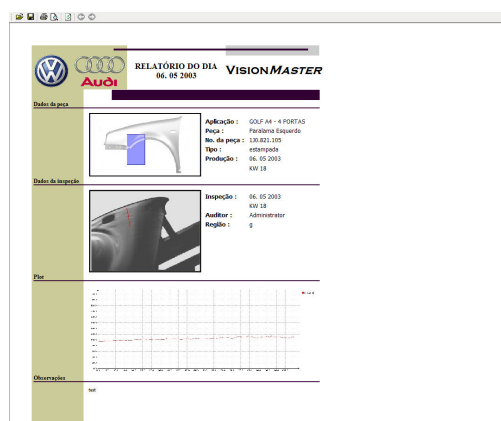


Abbildung 5.31 Die Protokollansicht

5.3.3 Das Modul DbInterface

Das Modul DbInterface ist das Bindeglied zwischen der Anwendung und der Datenbank und gliedert sich in folgende fünf Hauptkomponenten:

- Klasse VisionMasterDb
- Klasse CProtocol
- Paket ZipArchive
- Paket ADO
- Strukturen zur Verwaltung der Daten während der Programmlaufzeit

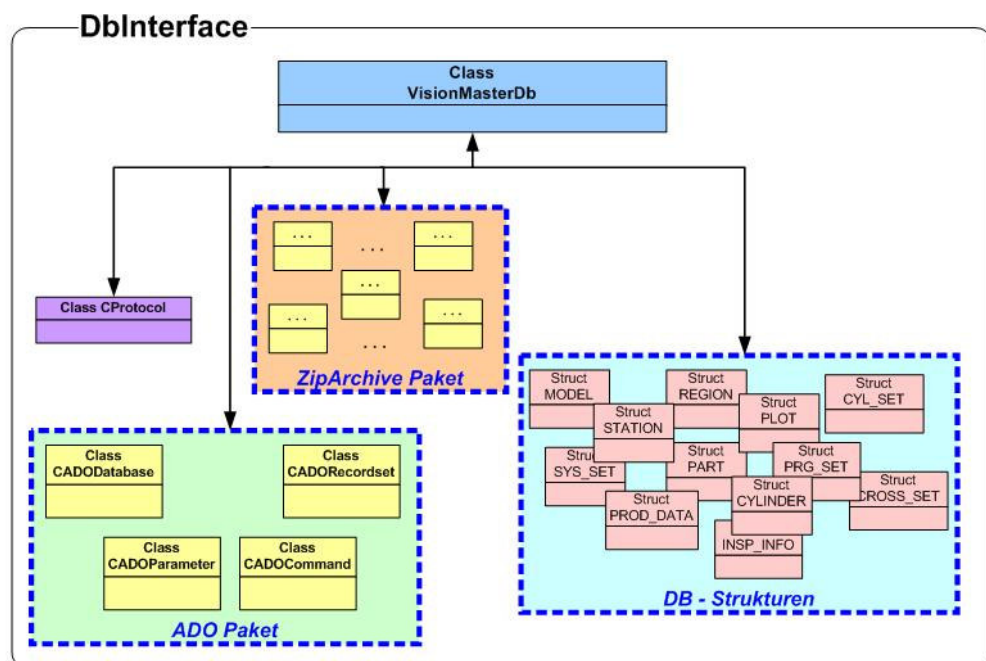


Abbildung 5.32 – Hauptkomponenten des Moduls DbInterface

Die Hauptaufgaben von Dbinterface liegen neben dem Datenbankzugriff (lesend und schreibend), in der Erstellung von Sicherungskopien der Datenbank und aller zugehörigen Dateien (DB-Backup), sowie der Aufbereitung von Protokolldaten, die zur Visualisierung an die oberste Schicht übergeben werden.

Die Klasse VisionMasterDb repräsentiert das Kontrollzentrum des Moduls. Sie steuert die Kommunikation mit allen anderen Komponenten des Moduls in beide Richtungen. Über die öffentlichen Funktionen dieser Klasse können Module höherer Ebenen Zugriff auf das Modul DbInterface erlangen und dessen Funktionalität ausnützen.

Der direkte Zugriff auf die Daten erfolgt über die CADO-Klassen. Bei den CADO-Klassen, die sich im ADO-Paket befinden, handelt es sich um ein Projekt, das die ADO Technologie in der Programmiersprache C++ kompakt und übersichtlich in objektorientierter Form umsetzt (siehe Anhang A). Dieses Paket wurde von Herrn Carlos Antollini entwickelt und freundlicherweise zur freien Nutzung zur Verfügung gestellt.

Beim ZipArchive handelt es sich ebenfalls um ein eigenständiges komplexes Projekt, welches in das Modul DbInterface integriert wurde, um eine Kompressionsfunktionalität zu erlangen. Im allgemeinen handelt es sich um Bibliotheken zur Komprimierung/Dekomprimierung von Dateien im bekannten WinZip- bzw. PKZIP-Format. Die ZipArchiv Bibliothek wurde von Tadeusz Dracz entwickelt und basiert auf den Kompressions- und Dekompressionsfunktionen von Jean-Loup Gailly.

Nähere Informationen zum ZipArchive sind auf der Homepage des Entwicklers unter: <http://www.artpol-software.com/zipdoc/index.html>

Die Klasse CProtocol kapselt die Daten und Funktionen, die zur Erzeugung des Inspektionsprotokolls notwendig sind.

Zur Abbildung der Daten aus der Datenbank werden im Programm Strukturen eingeführt, die eine Eins-zu-eins-Abbildung der entsprechenden Datenbanktabelle darstellen. Jeder Tabelle der Datenbank wird eine Struktur zugewiesen, deren Membervariablen die Attribute eines Datensatzes darstellen.

Datenbaktabelle

	Name	Image Path
▶	AUDI A3 - 2 PORTAS	base\images\AUDI A3 - 2 PORTAS\A2P.BMP
	AUDI A3 - 4 PORTAS	base\images\AUDI A3 - 4 PORTAS\A4P.bmp
	GOLF A4 - 2 PORTAS	Databas
	GOLF A4 - 4 PORTAS	Databasi
	MODEL	DataBas
*		

Programmcode

```

*  DEFINITION OF THE STRUCTURE MODEL  - *
*
struct MODEL
{
    CString name;
    CString imgPath;
};
        
```

Abbildung

Abbildung 5.33 – Mapping einer Datenbanktabelle im Programm

5.3.4 Datenbankstruktur

Ausgehend von den Anforderungen an das System wurde als erster Schritt der Datenbankmodellierung eine abstrakte Darstellung des Realweltausschnittes in ein ER-Modell umgesetzt.

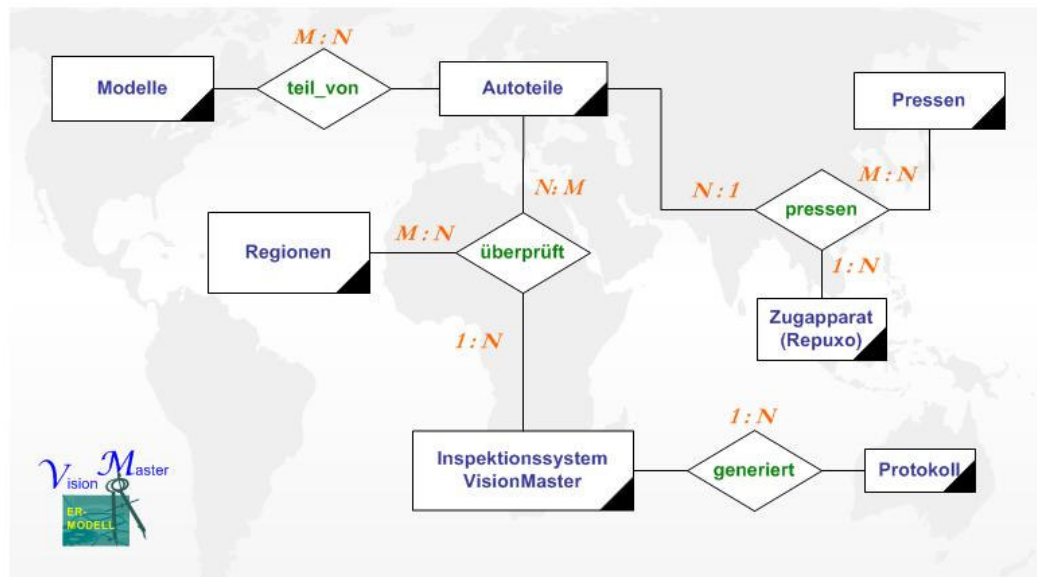


Abbildung 5.34 – ER-Modell des Systems VisionMaster

Die Transformation des ER-Modells in ein relationales Datenbankmodell erfolgt nach den Regeln der relationalen Entwurfstheorie (siehe Kapitel 4.2 bzw. Kapitel 4.4). Als Datenbanksystem wird MS Access der Firma Microsoft eingesetzt.

Durch Anwendung der Normalformen und Betrachtung der verschiedenen Beziehungstypen ergeben sich folgende Relationen, die anschließend in Tabellen umgesetzt werden:

- CrossbarSettings
- Cylinders
- CylinderSettings
- InspectionInfo
- Models
- partOf
- Parts
- Plot
- ProductionData
- ProgramSettings
- Regions
- Stations
- SystemSettings

Im Folgenden wird der prinzipielle Aufbau aller Tabellen der Datenbank beschrieben. Die Primärschlüssel einer Tabelle werden mir grüner Farbe hervorgehoben.

CROSSBARSETTINGS		
<p>Diese Tabelle enthält die Parameterwerte der sechs Pressen, die zu einem bestimmten Produktionszeitpunkt für ein bestimmtes Teil eingestellt wurden.</p>		
Attributname	Format	Beschreibung
prodDate	Datum/Uhrzeit	Produktionsdatum (Tag, Monat, Jahr und Zeit der Teilepressung) des entsprechenden Autoteils
stationName	String (50 Zeichen)	Name der Presse (Station)
reg	Double	Presseparameter
force	Double	Presseparameter
realForce	Double	Presseparameter
balance	Double	Presseparameter
extract	Double	Presseparameter
golpes	Double	Presseparameter

Tabelle 5.4 – Struktur der Tabelle „CrossbarSettings“ in der Datenbank

CYLINDERSETTINGS		
<p>Diese Tabelle enthält die einzelnen Parameterwerte aller acht Zylinder, die zu einem bestimmten Produktionszeitpunkt für ein bestimmtes Teil eingestellt wurden.</p>		
Attributname	Format	Beschreibung
cylNo	Integer	Nummer der Zylinders
prodDate	Datum/Uhrzeit	Produktionsdatum (Tag, Monat, Jahr und Zeit der Teilepressung) des entsprechenden Autoteils
weight	Double	Parameter
reg	Double	Parameter
angleP	Double	Parameter
angleS	Double	Parameter

Tabelle 5.5 - Struktur der Tabelle „CylinderSettings“ in der Datenbank

CYLINDERS		
Diese Tabelle enthält die Nummer und den jeweiligen Namen aller acht Zylinder, die sich auf dem Zugapparat (Repuxo-Apparat) befinden.		
Attributname	Format	Beschreibung
cylNo	Integer	Nummer der Zylinder
cylName	String (15 Zeichen)	Name des Zylinders

Tabelle 5.6 - Struktur der Tabelle „Cylinders“ in der Datenbank

INSPECTIONINFO		
Diese Tabelle enthält alle wichtigen Informationen einer Inspektion. Die gespeicherten Daten werden bei der Generierung eines Protokolls herangezogen.		
Attributname	Format	Beschreibung
inspDate	Datum/Uhrzeit	Tag, Monat, Jahr und Zeit der Inspektion
partNo	String (15 Zeichen)	Name des Zylinders
regionID	String (20 Zeichen)	Eindeutige ID der Region
prodDate	Datum/Uhrzeit	Produktionsdatum (Tag, Monat, Jahr und Zeit der Teilepressung) des entsprechenden Autoteils
imgLivePath	String (255 Zeichen)	Relativer Pfad des Live-Bildes
imgPlotPath	String (255 Zeichen)	Relativer Pfad des Plot-Bildes
Auditor	String (30 Zeichen)	Name des Auditors
comment	String (255 Zeichen)	Kommentar zur Inspektion (z.B. Fehlerbeschreibung usw.)

Tabelle 5.7 – Struktur der Tabelle „InspectionInfo“ in der Datenbank

MODELS		
Diese Tabelle enthält die Automodelle, deren Teile bei BUC gepresst oder inspiziert werden.		
Attributname	Format	Beschreibung
modelName	String (50 Zeichen)	Eindeutiger Name des Automodells
imgPath	String (255 Zeichen)	Relativer Pfad des zugehörigen Model-Bildes

Tabelle 5.8 - Struktur der Tabelle „Models“ in der Datenbank

PARTS		
Diese Tabelle enthält die Autoteile, die bei BUC gepresst oder inspiziert werden.		
Attributname	Format	Beschreibung
partNo	String (15 Zeichen)	Eindeutige Identifikationsnummer des Autoteils
partName	String (80 Zeichen)	Name des Autoteils
description	String (15 Zeichen)	Kurze Beschreibung des Autoteils
spec	String (25 Zeichen)	Spezifikation des Teils. Spec \in {estampada, subconjunto, pintada, especial}
imgPath	String (255 Zeichen)	Relativer Pfad des zugehörigen Autoteil-Bildes

Tabelle 5.9 - Struktur der Tabelle „Parts“ in der Datenbank

PARTOF		
Diese Tabelle repräsentiert die Beziehungen, die zwischen Automodellen und Autoteilen herrschen. Die M:N-Beziehung zwischen Modellen (Models) und Autoteilen (Parts) wird durch diese zusätzliche Tabelle aufgelöst.		
Attributname	Format	Beschreibung
modelName	String (50 Zeichen)	Eindeutiger Name des Automodells
partNo	String (15 Zeichen)	Eindeutige Identifikationsnummer des Autoteils

Tabelle 5.10 - Struktur der Tabelle „partOf“ in der Datenbank

PROGRAMSETTINGS		
Diese Tabelle enthält Parameter die von der VisionMaster-Software zu Beginn eingelesen werden (z.B. Pfad des Hintergrundbildes usw.). Diese Tabelle wurde beim ER-Modell nicht berücksichtigt, sondern zusätzlich eingeführt, um das Hauptprogramm zu entlasten.		
Attributname	Format	Beschreibung
id	Long Integer	Autowert (automatische Nummerierung)
setName	String (50 Zeichen)	Name des Programmparameters
setValue	String (50 Zeichen)	Wert des Programmparameters

Tabelle 5.11 - Struktur der Tabelle „ProgramSettings“ in der Datenbank

PLOT		
<p>Diese Tabelle enthält die definierten Schnitte (Geraden) durch die einzelnen Grauwertbilder. Anhand des Verlaufs einer Geraden werden die Pixel des Bildes über die diese Gerade verläuft ausgewählt und in ein Grauwerthistogramm übertragen.</p>		
Attributname	Format	Beschreibung
plotName	String (50 Zeichen)	Eindeutiger Name der Plotgeraden
regionID	String (15 Zeichen)	Eindeutige ID der Region
color	String (20 Zeichen)	Farbe mit der die Plotkurve gezeichnet werde soll
description	String (50 Zeichen)	Kurze Beschreibung der Kurve
lineSx	Integer	X- Startpunkt der Geraden
lineSy	Integer	Y- Startpunkt der Geraden
lineEx	Integer	X- Endpunkt der Geraden
lineEy	Integer	X- Endpunkt der Geraden
min	Integer	Minimaler Grauwert der zugehörigen Grauwertkurve des Bild durch das die Gerade verläuft
max	Integer	Maximaler Grauwert der zugehörigen Grauwertkurve des Bild durch das die Gerade verläuft
average	Integer	Mittlerer Grauwert der zugehörigen Grauwertkurve des Bild durch das die Gerade verläuft

Tabelle 5.12 - Struktur der Tabelle „Plot“ in der Datenbank

REGIONS		
<p>Diese Tabelle enthält alle definierten Regionen aller Autoteile. Eine Region wird als eine rechteckige Fläche über die ausgewählten Zeilen und Spalten definiert.</p>		
Attributname	Format	Beschreibung
regionID	String (20 Zeichen)	Eindeutige ID der Region
column1	String (1 Zeichen)	Spalte 1
row1	String (1 Zeichen)	Zeile 1
column2	String (1 Zeichen)	Spalte 2
row2	String (1 Zeichen)	Reihe 2

Tabelle 5.13 - Struktur der Tabelle „ProgramSettings“ in der Datenbank

PRODUCTIONDATA		
<p>Diese Tabelle enthält die wichtigen Parameter des Materials, das die für die Pressung eines bestimmten Teils zu einem bestimmten Zeitpunkt verwendet wurde.</p>		
Attributname	Format	Beschreibung
prodDate	Datum/Uhrzeit	Produktionsdatum (Tag, Monat, Jahr und Zeit der Teilepressung) des entsprechenden Autoteils.
partNo	String (15 Zeichen)	Eindeutige Identifikationsnummer des Autoteils
rm	String (50 Zeichen)	Materialparameter
rp	String (50 Zeichen)	Materialparameter
a80	String (50 Zeichen)	Materialparameter
r90	String (50 Zeichen)	Materialparameter
n90	String (50 Zeichen)	Materialparameter
faceRa	String (50 Zeichen)	Materialparameter
facePc	String (50 Zeichen)	Materialparameter

Tabelle 5.14 - Struktur der Tabelle „ProductionData“ in der Datenbank

STATIONS		
<p>Diese Tabelle enthält die Namen und Standardwerte aller sechs Stationen der Gesamtpresse.</p>		
Attributname	Format	Beschreibung
stationName	String (50 Zeichen)	Eindeutiger Name einer Station
minForce	Integer	Minimale Kraft, die bei dieser Station eingestellt werden kann
maxForce	Integer	Maximale Kraft, die bei dieser Station eingestellt werden kann
minGAjst	Integer	Minimale Kraft, die bei dieser Station beim Testen eingestellt wird
maxGAjst	Integer	Maximale Kraft, die bei dieser Station beim Testen eingestellt wird
minGAuto	Integer	Minimale Kraft, die bei dieser Station beim „Automatico“ eingestellt wird
maxGAuto	Integer	Maximale Kraft, die bei dieser Station beim „Automatico“ eingestellt wird

Tabelle 5.15 - Struktur der Tabelle „Stations“ in der Datenbank

SYSTEMSETTINGS		
Diese Tabelle enthält die Messtisch- und Kameraeinstellungen, die bei der Inspektion immer vom Autoteil und der zu inspizierenden Region abhängen.		
Attributname	Format	Beschreibung
partNo	String (15 Zeichen)	Eindeutige Identifikationsnummer des Autoteils
regionID	String (20 Zeichen)	Eindeutige ID der Region
comment	String (50 Zeichen)	Kommentar zu der Region des entsprechenden Autoteils. Hier wird empfohlen den Namen der Region zu verwenden, der vom Teil abhängt.
refImgPath	String (255 Zeichen)	Pfad des zugehörigen Referenzbildes
tblBase	String (2 Zeichen)	Bodenposition des Tisches
tblAngleX	Integer	X-Neigungswinkel des Tisches
tblAngleY	Integer	Y-Neigungswinkel des Tisches
camHorizontal	Integer	Horizontale Ausrichtung der Kamera
camVertical	Integer	Vertikale Ausrichtung der Kamera
camHeight	Integer	Höhe der Kamera
mount	String (4 Zeichen)	Position der Teilehalterung

Tabelle 5.16 - Struktur der Tabelle „SystemSettings“ in der Datenbank

Die Tabellen enthalten die Daten bzw. die logische Verknüpfung der Daten zueinander. Eine Tabelle ist untergliedert in Zeilen und Spalten. Die Spalten beinhalten die Attribute, also die beschreibenden Merkmale eines Datensatzes, die Zeilen den Datensatz selbst.

Die Beziehungen zwischen den einzelnen Tabellen werden abhängig vom Beziehungstyp (1:-N, N:1, M:N usw.) entweder durch direkte¹⁾ oder indirekte Verknüpfung²⁾ realisiert.

1)) Bei einer 1:N- bzw. N:1-Beziehung wird die Beziehung durch Einführung eines Fremdschlüssels erzeugt, der direkt auf die zugehörige Tabelle verweist, bei der dieser Fremdschlüssel einen Primärschlüssel darstellt.

2) M:N-Beziehungen können nicht direkt umgesetzt werden. Es muss eine zusätzliche Tabelle eingeführt werden.

Abbildung 5.35 verdeutlicht die Beziehungen der einzelnen Tabellen zueinander:

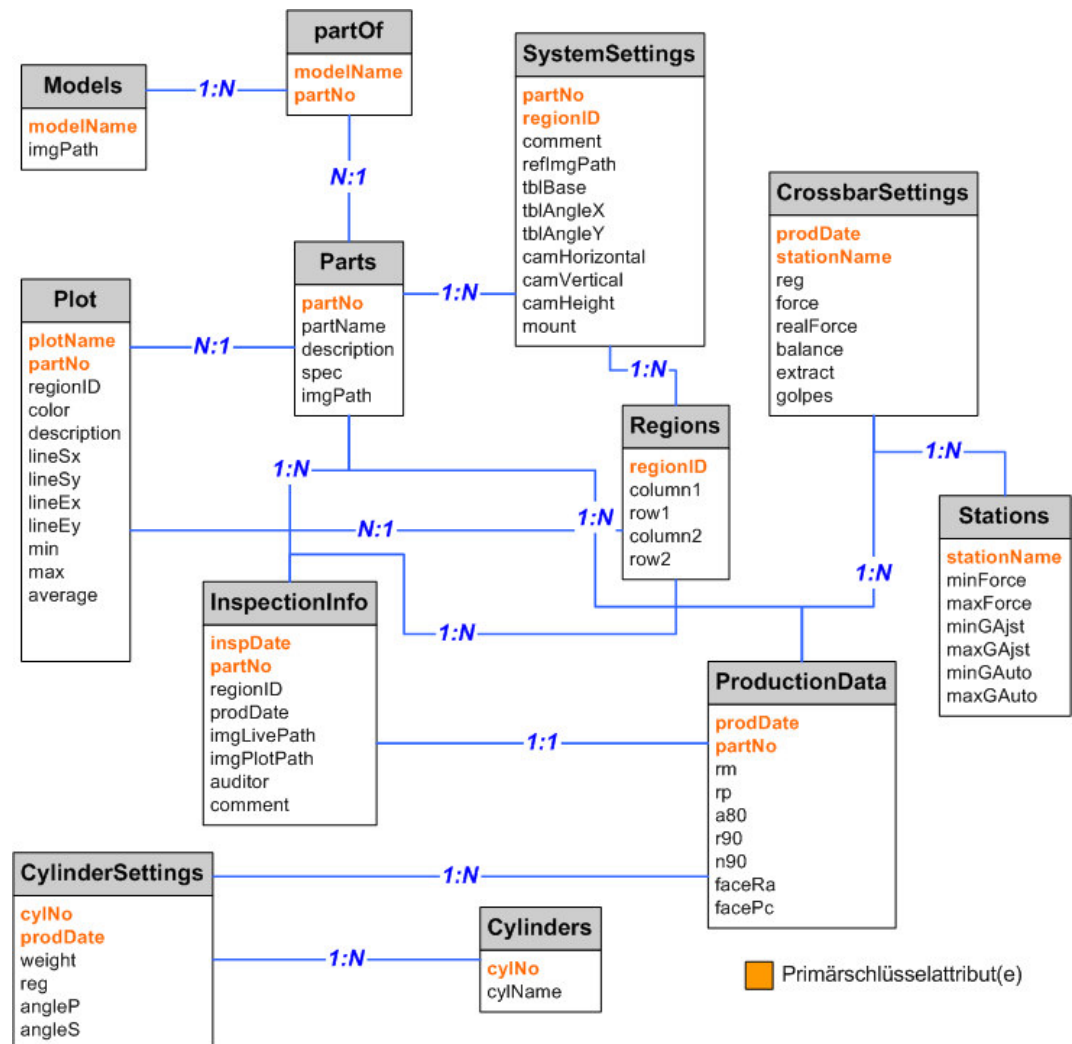


Abbildung 5.35 – VisionMaster Beziehungsdiagramm

Um die Größe der Datenbank möglichst klein zu halten, und den Zugriff auf diese nicht unnötig zu verlangsamen werden Bilder nicht direkt in der Datenbank gespeichert, sondern in entsprechend dafür eingerichtete Verzeichnisse auf der Festplatte. In der Datenbank befindet sich lediglich ein Verweis auf den Pfad des zugehörigen Bildes.

Damit ein korrekter Programmablauf erfolgen werden kann, muss eine vorgegebene Verzeichnisstruktur eingehalten werden. Diese Pfade sind im Programm fest kodiert. Eine Abweichung davon führt zwangsläufig zu mehr oder weniger schwerwiegenden Fehlern im Programm.

Damit die richtige Verzeichnisstruktur eingehalten wird, werden die entsprechenden Verzeichnisse bei der Installation der Software eingerichtet.

Es sollte strengstens vermieden werden Manipulationen an der Verzeichnisstruktur bzw. dem Inhalt der einzelnen Verzeichnisse vorzunehmen.

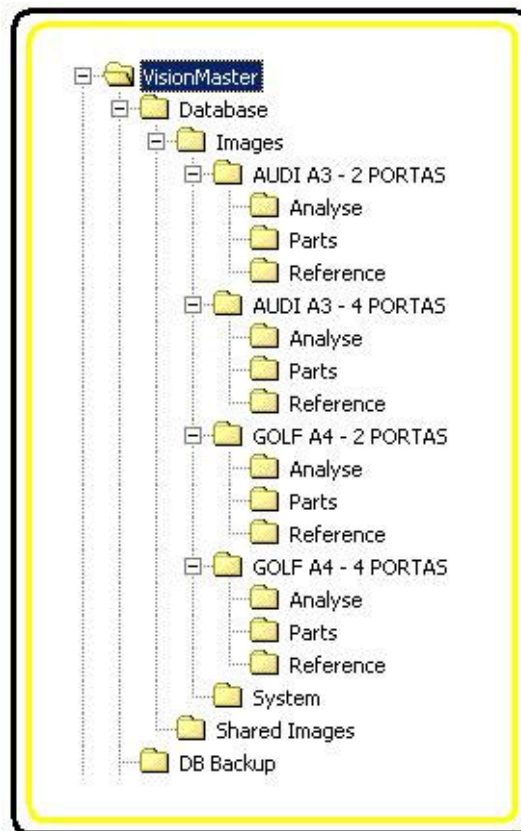


Abbildung 5.36 – Verzeichnisstruktur von VisionMaster

Im Stammverzeichnis von VisionMaster werden bei der Installation zwei Unterverzeichnisse angelegt:

- Database
- DB Backup

Die VisionMaster-Software unterstützt die Erstellung von Sicherheitskopien aller relevanter Daten. Dabei werden im DB Backup-Verzeichnis diese Sicherheitskopien, die die Datenbank und die dazugehörigen Verzeichnisse enthalten, in Form von Zip-Dateien aufbewahrt.

Im Database-Verzeichnis befindet sich die MS Access Datenbank als verschlüsselte Datei (VisionMaster.vmd). Diese wird bei Programmstart entschlüsselt und geöffnet. Außerdem werden dort sämtliche Bilder aufbewahrt, die vom System verwendet bzw. erzeugt werden. Dabei handelt es sich um Bilder der Automodelle, Autoteile, Inspektionsbilder und Plotgraphiken. Dafür werden dort zwei Unterverzeichnisse (Images und Shared Images) angelegt.

Unter Shared Images werden alle Bilder abgelegt, die von mehr als einem Teil gleichzeitig verwendet werden. Dies erspart die Mehrfachspeicherung von gleichen Bildern auf der Festplatte und Erleichtert die Verwaltung der Dateien (z.B. beim Löschen eines Bildes, das von mehreren Modellen benutzt wird).

Im Images Verzeichnis wird für jedes Modell ein eigener Ordner angelegt. Dieser Ordner wird vom Programm beim Einfügen eines neuen Modells in der Datenbank automatisch kreiert. Dort befindet sich das Bild des entsprechenden Automodells und die drei zusätzlichen Ordner Analyse, Parts und Reference:

Im Analyse Ordner werden sämtliche Bilder einer Inspektion gespeichert (z.B. Livebild, Plotgraphik usw.).

Der Ordner Parts enthält die Bilder der Autoteile, die zu dem entsprechenden Modell gehören. Im Verzeichnis Reference sind die Referenzbilder der einzelnen Autoteile untergebracht.

Das Verzeichnis System, das ebenfalls im Images Verzeichnis untergebracht ist, wurde für Bilder und Graphiken vorgesehen, die für die Oberfläche des Programms benötigt werden. Im Moment befindet sich dort nur das Hintergrundbild (graues VW-Logo) des Programms, das bei Programmstart eingelesen wird.

5.4 Erzeugen von Datenbank-Backups

Unter den Funktionen, die das System VisionMaster bereitstellt befindet sich auch das Erzeugen von Sicherungskopien bzw. das Einspielen vorhandenen Sicherheitskopien.

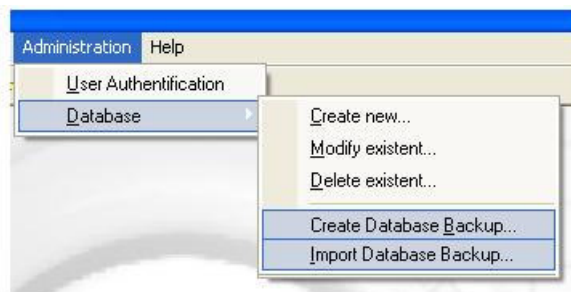


Abbildung 5.37 – Erstellung und Importierung eines Datenbank-Backups

Das Erstellen von Sicherheitskopien hat folgenden Hintergrund:

- Bei Beschädigung der vorhandenen Konfiguration (z.B. der Verzeichnisstruktur) kann ein vorheriger (abgespeicherter) Zustand des Systems zu jedem beliebigen Zeitpunkt wiederhergestellt werden.
- Die Datenbank kann auf mehrere Dateien aufgeteilt werden. Je nach Bedarf kann z.B. nach einem bestimmten Zeitpunkt die aktuelle Datenbank mit den zugehörigen Verzeichnissen abgespeichert werden und eine neue (evtl. leere) Datei angelegt werden. Dies kann z.B. dazu verwendet werden, eine übersichtliche Verwaltung der Inspektionen zu gewährleisten. Für jede Periode (z.B. Quartal) kann je ein Archiv angelegt werden. Außerdem wird dadurch vermieden, dass die Datenbankgröße so weit ansteigt, dass ein Arbeiten nicht mehr möglich ist.

Alle wichtige Dateien (Datenbank, Bilder usw.) werden in einem Zip-Archiv abgelegt und in dem Unterverzeichnis DB Backup archiviert. Für die Erzeugung der Zip-Datei wurde die Bibliothek **ZArchive** in das Modul DbInterface integriert.

Um ein Backup der vorhandenen Datenbankkonfiguration vorzunehmen, muss man unter dem Menüpunkt *Administration* lediglich auf den Unterpunkt *Create Database Backup* klicken (siehe Abbildung 5.37).

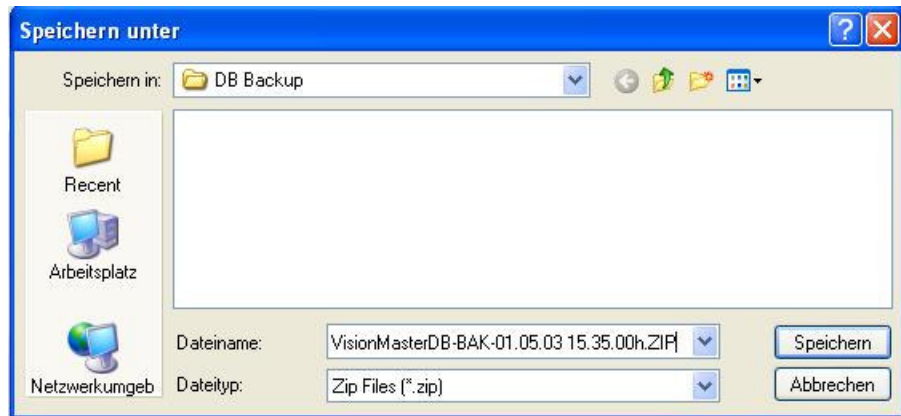


Abbildung 5.38 - Der Speichern unter Dialog der DB-Backup Funktion

Standardmäßig wird für den Namen der Backup-Datei das aktuelle Datum und die aktuelle Uhrzeit vorgeschlagen. Der Namen der Datei kann natürlich umbenannt werden.



Abbildung 5.39 – Der Öffnen Dialog der DB-Backup Funktion

Will man die aktuelle Datenbank durch eine ältere Version ersetzen, so wählt man unter *Administration* den Unterpunkt *Import Database Backup* (siehe Abbildung 5.37).

Die ausgewählte Zip-Datei wird im aktuellen Programmverzeichnis entpackt. Alle alten Dateien werden dadurch überschrieben.

6 Optimierungsvorschläge

Kein System ist vollkommen, geschweige denn „überlebensfähig“, wenn es nicht gepflegt und weiterentwickelt wird.

Das System VisionMaster darf im momentanen Zustand nicht als Komplettsystem für die Qualitätssicherung von Pressteilen im Automobilbereich angesehen werden. Es soll die Qualitätssicherung weitgehend unterstützen und erleichtern.

Um jedoch eine effiziente und sinnvolle Nutzung bzw. Erweiterung zu ermöglichen, sollte sich das System ständig dem kontinuierlichen Verbesserungsprozess unterziehen und weiterentwickelt werden.

Im Weiteren sind einige Vorschläge aufgeführt, die dazu beitragen könnten die Leistung und Effizienz des System zu verbessern:

■ **Protokollierung der Systemschwächen**

Da es sich bei der vorliegenden Version von VisionMaster noch um die erste Ausgabe handelt, und eine vollständige Testphase nach der Implementierungsphase von den Entwicklern des Systems aus Zeitgründen nicht vollständig abgeschlossen werden konnte, sollte der Einsatz und Umgang mit dem System in den ersten Monaten als erweiterte Testphase genutzt werden.

Vermeidlich auftretende Schwächen und Fehler des Systems sollten erfasst und protokolliert werden, sodass eine rasche Beseitigung der Fehlerursache erfolgen kann.

Für die Weiterentwicklung des Systems ist es zwingend erforderlich Fachkräfte einzusetzen, die im Bereich der C++- und Visual C++ Programmierung ausreichend gute Kenntnisse vorweisen können.

Voraussetzung dabei sind:

- Kenntnisse in ADO-Programmierung
- Sehr gute Kenntnisse in MFC/DLL- Programmierung)

■ **Implementierung zusätzlicher Auswertungsalgorithmen**

Durch die Implementierung zusätzlicher Auswertungsalgorithmen kann die Qualität des Systems erheblich gesteigert werden. Bekannte nützliche Algorithmen wären dabei z.B. Surface, Feature oder Metal, die beim DSight-System von LMI eingesetzt werden (vgl. LMI Automotive - DSight AS-2 User´s Manual V.2.04).

Vor allem aber sollten die Fortschritte der letzten Jahre im Bereich der Bildverarbeitung ausgenutzt werden und neue, vermeidlich bessere Algorithmen gesucht und eingesetzt werden.

■ Automatisierung des Messtisches

Der wichtigste Punkt im Bezug auf die Optimierung/Erweiterung, der im Moment auch den größten Schwachpunkt darstellt, betrifft die Automatisierung der Messeinrichtung.

Im Moment wird die Einstellung des Messsystems weitgehend manuell vorgenommen. Dies verhindert eine exakte Reproduzierbarkeit einer Messung, da sich kleinste Abweichungen bei der Höheneinstellung und Linseneinstellung der Kamera oder der Positionierung des Messtisches aufsummieren und das Ergebnis einer Messung stark beeinträchtigen können.

Die Automatisierung der Messeinrichtung erfordert somit eine rechnergesteuerte Ausrichtung der oben genannten Systemkomponenten.

Dies kann zum Beispiel durch Anbringen von Schrittmotoren realisiert werden.

Bei der Automatisierung des Messtisches gab es schon während der Entwicklungsphase des Systems mehrere Verhandlung mit verschiedenen Unternehmen, über die Entwicklung und Lieferung eines motor- gesteuerten Messtisches. Leider sind alle Verhandlungen aus finanziellen Gründen gescheitert.

Die nachfolgende Tabelle zeigt die Anforderungen, die auch bei einer automatischen Positionierung des Messtisches eingehalten werden müssen:

1. Winkeleinstellung je 45° in X- und Y- Richtung - ausgehend von Grundposition	2. Tischverschiebung je 2 m in X- und Y- Richtung - ausgehend von Grundposition
3. Genauigkeit 0,1 mm in X-Y-Richtung 8 Minuten Winkelgenauigkeit	4. Tragkraft 20-25 kg

Tabelle 6.1 – Anforderungen an den motor- gesteuerten Messtisch

■ Gleichmäßiges Auftragen der Glanzflüssigkeit

Auch in Bezug auf die Glanzflüssigkeit sollte eine Lösung gefunden werden, die ein automatisches und gleichmäßiges Auftragen der Flüssigkeit auf der Oberfläche eines Teiles ermöglicht. Dabei könnten z.B. spezielle Sprüheinrichtungen mit feinen Düsen eingesetzt werden. Ein herkömmlicher Roboterarm, wie er in der Industrie seinen täglichem Einsatz findet (z.B. Lackiererei) könnte für die Positionierung solcher Düsen in Betracht gezogen werden.

■ **Zusätzliche Masken für Datenbankmodifikation**

Die VisionMaster-Software stellt einige Masken zur Manipulation der Datenbank (Oberflächen) zur Verfügung, die es dem Benutzer erlauben einfach Daten der Datenbank hinzuzufügen bzw. zu ändern.

Die aktuelle Version unterstützt dabei folgende Aktionen:

- Hinzufügen von neuen Automodellen, Autoteilen, Regionen und Plotkurven
- Löschen vorhandener Automodelle, Autoteile, Regionen und Plotkurven
- Änderung vorhandener Automodelle, Autoteile und Regionen
- Hinzufügen von Presseparameter

Aus Zeitgründen wurden nur die wichtigsten Fälle berücksichtigt. Für die Modifikation anderer Tabellen muss direkt auf die Access-Datenbank zugegriffen werden, was erstens umständlich ist, da diese erst von der vorliegenden verschlüsselten Datei zu einer MS-Access Datei exportiert werden müsste, und zweitens aus Sicherheitsgründen vermieden werden sollte. Es empfiehlt sich deshalb auch für die Modifikation der fehlenden Tabellen zugeschnittene Masken einzusetzen, die die Datenmanipulation regeln und Fehlerfälle weitgehend verhindern.

Abbildung 6.1 zeigt die Eingabemaske für die Presseparameter. Für die Änderung der eingegebenen Parameter steht noch keine Oberfläche bereit und sollte nachträglich in das Programm integriert werden.

Abbildung 6.1 – Eingabemaske für die Presseparameter

■ Klassifikation der Fehler

Das größte Ziel bei der Oberflächeninspektion ist die Klassifikation von Fehlern.

Im allgemeinen hat die Klassifikation das Ziel aufgrund von Merkmalen eine Zuordnung treffen zu können und unbekannte Objekte aufgrund von Beobachtungen bestimmten Klassen zuzuordnen. Dabei wird ein Objekt durch verschiedene Merkmale beschrieben. Die Anzahl der Merkmale, die gleichzeitig berücksichtigt werden müssen, um eine eindeutige Zuordnung treffen zu können gibt auch die Komplexität der Klassifikation an. Bei der Wahl der Merkmale sollte darauf geachtet werden, dass diese weitgehend unabhängig voneinander sind.

Eine Überlappung von Klassen sollte auch vermieden werden.

Abbildung 6.2 zeigt das prinzipielle Schema einer Mustererkennung/Klassifikation:

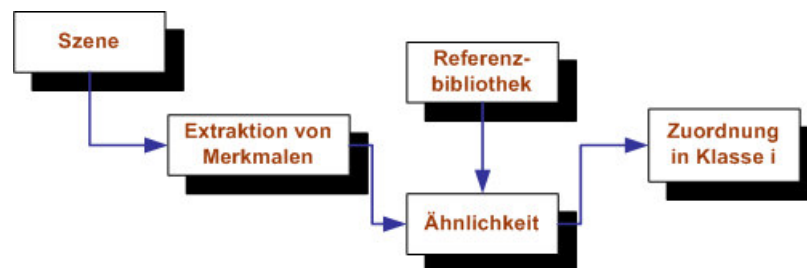


Abbildung 6.2 – Schema einer Mustererkennung und Klassifikation

In unserem Fall soll das System nach der Auswertung der Bilder, anhand der erkannten Defekte die inspizierten Teile in verschiedene Qualitätsklassen einteilen.

Die Bildung möglicher Klassen könnte beispielsweise (einfacher Fall) folgendermaßen aussehen:

Klasse A	:	keine Defekte oder Defekte vernachlässigbar
Klasse B	:	einige Defekte im akzeptablen Toleranzbereich
Klasse C	:	Defekte können nicht toleriert werden – Teil ist unbrauchbar

Tabelle 6.2 – Mögliche Klassenbildung für Pressteile

In der Literatur werden verschiedene unterschiedliche mathematische Verfahren aufgeführt, wie man eine Klassenzuordnung bestmöglich durchführt (z.B. lineare Klassifizierung, nicht-lineare Klassifizierung, Bayes- Klassifizierung usw.). Die Wahl des entsprechenden Verfahrens hängt unter anderem von der Art und Abhängigkeit der Merkmale ab.

Die Grundlage der Klassifikation bilden jedoch immer noch Messungen (Beobachtungen), die sorgfältig und kontinuierlich durchgeführt werden müssen.

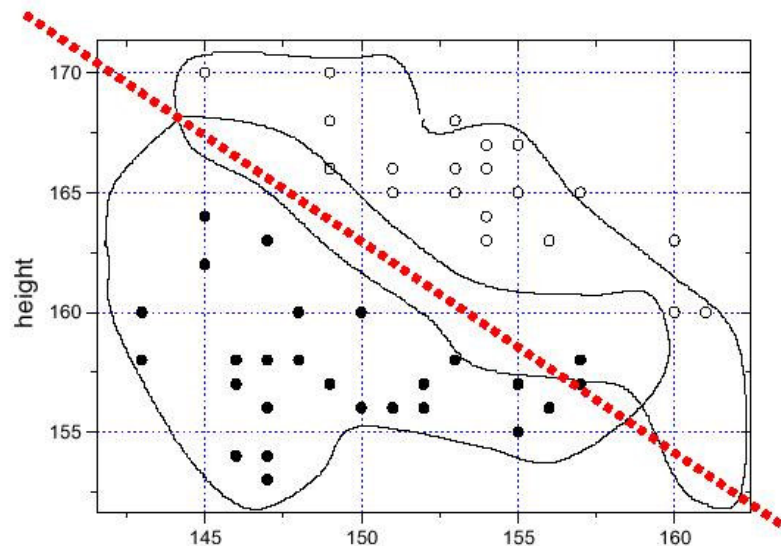


Abbildung 6.3 – lineare Klassifikation mit überlappenden Klassen

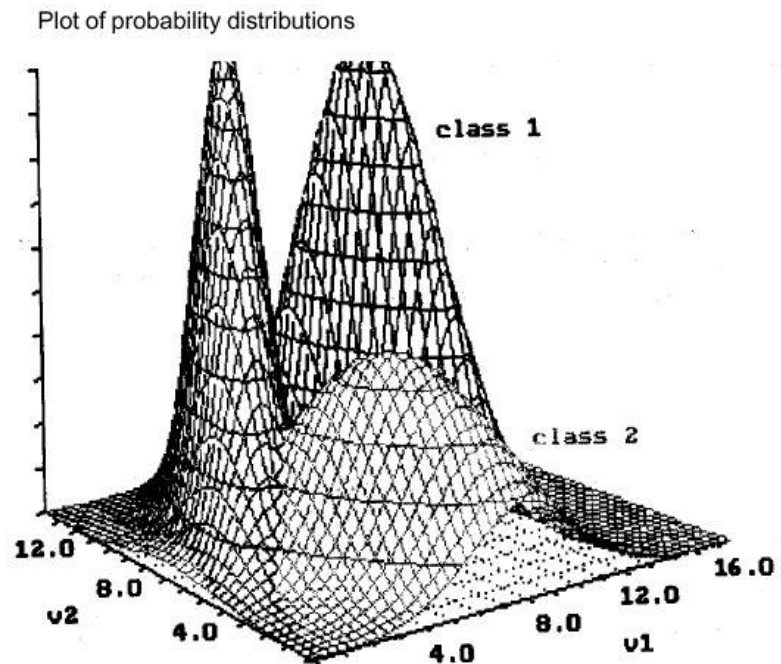


Abbildung 6.4 – Klassifikation mit dem Bayes-Klassifizierer

Schlusswort

Als Abschluss dieser Arbeit möchte ich meinen Dank an alle Beteiligten aussprechen, die bei der Erstellung dieser Diplomarbeit beigetragen und mich während diesem Zeitraum unterstützt haben.

Besonderer Dank gilt meinen Betreuern Heiko Borho und Jürgen Massierer für die Betreuung und Unterstützung vor Ort, meiner Kollegin Silmara Guebur Alessi für die Mitarbeit am Projekt und meinem Professor Dr. Wolfgang Arndt, der die Idee zu dieser Diplomarbeit entwickelt und mich dabei unterstützt und betreut hat.

Die Arbeit an dem Projekt VisionMaster hat nicht nur unheimlich viel Spaß gemacht, sondern war auch eine Herausforderung aus der eine Menge neuer, positiver Erkenntnisse resultierten. Auch wenn das Projekt VisionMaster den zeitlichen Rahmen und Aufwand einer Diplomarbeit überschritten hat, so trieb die Motivation das Oberflächen-Inspektions-System optimal an die Anforderung anzupassen die Arbeit an diesem Projekt ständig voran. Leider konnten wir aus Zeitgründen nicht alle unsere Ideen in dieses Projekt einbringen und verwirklichen.

Mit dieser Diplomarbeit ist der erste Schritt für eine rechnergestützte Qualitätssicherung von Pressteilen getan. Ich hoffe, dass auch in Zukunft eine Weiterentwicklung und Optimierung des Systems unterstützt und gefördert wird...

Christos Agiovlassitis
Konstanz, 31. Mai 2003

Anhang – Die CADO Klassen

The CADODatabase Class

The `CADODatabase` class has a set of functions that corresponds to the `_ConnectionPtr`

`CADODatabase::CADODatabase`

Creates a `CADODatabase` object.

```
CADODatabase();
```

`CADODatabase::Open`

The `Open` function Opens a connection to a Database.

```
BOOL Open(LPCTSTR lpstrConnection = _T(""));
```

Parameters:

`LPCTSTR lpstrConnection` = The connection string

Note: The class has the function `SetConnectionString`. You can insert the connection string through this function. In this case you can do the following.

```
//Sample with Connection String for Access database  
  
CADODatabase pAdoDb;  
CString strConnection = _T("");  
  
strConnection = _T("Provider=Microsoft.Jet.OLEDB.4.0;"  
    "Data Source=C:\\VCProjects\\ADO\\  
    Test\\dbTest.mdb");  
pAdoDb.SetConnectionString(strConnection);  
  
if (pAdoDb.Open())  
    DoSomething();
```

CADODatabase::Execute

The `Execute` function executes a sql statement in the open database.

```
BOOL Execute (LPCTSTR lpstrExec)
```

Parameters

`LPCTSTR lpstrExec`: A string pointer containing the sql statement to execute.

```
CADODatabase pAdoDb;  
CString strConnection = _T("");  
  
strConnection = _T("Provider=MSDASQL;" "PersistSecurityInfo=False;"  
                  Trusted_Connection=Yes" "Data Source=  
                  Access Sql Server;catalog=sampled");  
  
if (pAdoDb.Open(strConnection))  
    pAdoDb.Execute("Delete From tblClients Where Cheker = 3");
```

Return Value

The function returns **TRUE** if was successfully.

CADODatabase::GetRecordsAffected

The `GetRecordsAffected` function returns the number of records affected to the last sql statement executed.

```
int GetRecordsAffected();
```

CADODatabase::GetActiveConnection

The `GetActiveConnection` returns the active connection.

```
_ConnectionPtr GetActiveConnection();
```

CADODatabase::GetRecordCount

`GetRecordCount` returns the number of records affected in a query.

```
DWORD GetRecordCount(_RecordsetPtr m_pRs);
```

Parameters

`_RecordsetPtr m_pRs` = The recordset.

CADODatabase::BeginTransaction

Call this function to initiate a transaction. After you call `BeginTransaction`, updates you make to your data take effect when you commit the transaction.

```
long BeginTransaction();
```

CADODatabase::CommitTransaction

Call `CommitTransaction` function to commit a transaction for example save a group of edits and updates to one or more databases.

```
long CommitTransaction();
```

CADODatabase::RollbackTransaction

Call `RollbackTransaction` function to end the current transaction and restore all databases to their condition before the transaction was begun.

```
long RollbackTransaction();
```

CADODatabase::IsOpen

The `IsOpen` function returns the status of the connection with the database.

```
BOOL IsOpen();
```

Return Value

The function returns **TRUE** if the connection to database is open.

CADODatabase::Close

The `Close` function close the connection to the database.

```
void Close();
```

CADODatabase::SetConnectionString

With the `SetConnectionString` You can insert the connection string.

```
void SetConnectionString(LPCTSTR lpstrConnection);
```

Parameters

`LPCTSTR lpstrConnection`: A connection string used for opening the database.

```
CString GetConnectionString();
```

CADODatabase::GetLastError

The `GetLastError` function returns the last error code.

```
DWORD GetLastError();
```

CADODatabase::GetLastErrorString

The `GetLastErrorString` function returns the last error string.

```
CString GetLastErrorString();
```

The CADORecordset Class

The `CADORecordset` class has a set of functions that corresponds to the `_RecordsetPtr`.

CADORecordset::CADORecordset

Creates a `CADORecordset` object.

```
CADODatabase();  
CADORecordset(CADODatabase* pAdoDatabase);
```

Parameters

`CADODatabase* pAdoDatabase` = A `CADODatabase` object pointer.

`CADORecordset::Open` = The `Open` function opens a recordset

```
BOOL Open(_ConnectionPtr mpdb, LPCTSTR lpstrExec = _T(""),  
          int nOption = CADORecordset::openUnknown);  
BOOL Open(LPCTSTR lpstrExec = _T(""),  
          int nOption = CADORecordset::openUnknown);
```

Parameters

`_ConnectionPtr mpdb` = A connection pointer.

`LPCWSTR lpstrExec`: = A string pointer containing an SQL select statement.

`int nOption` = An integer that defines the access mode. The values are as follows:

```
CADORecordset::openUnknown
CADORecordset::openQuery
CADORecordset::openTable
CADORecordset::openStoredProc
```

Return Value

Returns **TRUE** if was successfully.

CADORecordset::Execute

The `Execute` function opens a recordset

```
BOOL Execute(CADOCommand* pCommand);
```

Parameters

`CADOCommand* pCommand`: A `CADOCommand` pointer.

Return Value

Returns **TRUE** if was successfully.

CADORecordset::GetQuery

`GetQuery` returns the string containing the SQL Select statement.

```
CString GetQuery();
```

CADORecordset::SetQuery

```
void SetQuery(LPCWSTR strQuery);
```

Parameters

`LPCWSTR strQuery` = A string pointer containinig an SQL Select statement.

CADORecordset::RecordBinding

```
BOOL RecordBinding(CADORecordBinding pAdoRecordBinding);
```

Parameters

`CADORecordBinding pAdoRecordBinding`

`CADORecordset::GetRecordCount`

`GetRecordCount` = returns the number of records accessed in the recordset

```
DWORD GetRecordCount ();
```

CADORecordset::IsOpen

`IsOpen` determines if the recordset is open.

```
BOOL IsOpen ();
```

Return Value

This member function returns **TRUE** if the recordset has not been closed.

CADORecordset::Close

The `Close` function closes the recordset.

```
void Close ();
```

CADORecordset::GetFieldCount

The `GetFieldCount` function returns the number of fields in the recordset.

```
long GetFieldCount ();
```

CADORecordset::GetFieldValue

The `GetFieldValue` function returns a value that contains the value of a field.

```
BOOL GetFieldValue(LPCTSTR lpFieldName, int nValue);
BOOL GetFieldValue(int nIndex, int nValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, long lValue);
BOOL GetFieldValue(int nIndex, long lValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, unsigned long ulValue);
BOOL GetFieldValue(int nIndex, unsigned long ulValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, double dbValue);
BOOL GetFieldValue(int nIndex, double dbValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, CString strValue,
    CString strDateFormat = _T(""));
BOOL GetFieldValue(int nIndex, CString strValue,
    CString strDateFormat = _T(""));
BOOL GetFieldValue(LPCTSTR lpFieldName, COleDateTime time);
BOOL GetFieldValue(int nIndex, COleDateTime time);
BOOL GetFieldValue(LPCTSTR lpFieldName, bool bValue);
BOOL GetFieldValue(int nIndex, bool bValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, COleCurrency cyValue);
BOOL GetFieldValue(int nIndex, COleCurrency cyValue);
```

Parameters

<code>LPCTSTR lpFieldName</code>	: Pointer to a string that contains the name of a field.
<code>int nIndex</code>	: A zero-based index of the field in the recordset's Fields collection, for lookup by index.
<code>double dbValue</code>	: Reference to an object that will store the value of a field.
<code>long lValue</code>	: Reference to an object that will store the value of a field.
<code>unsigned long ulValue</code>	: Reference to an object that will store the value of a field.
<code>int nValue</code>	: Reference to an object that will store the value of a field.
<code>CString strValue</code>	: Reference to an object that will store the value of a field.
<code>CString strDateForma</code>	: Formatting time string similar to <code>strftime</code>

The more common formatting strings are:

<code>%a</code>	Abbreviated weekday name
<code>%A</code>	Full weekday name
<code>%b</code>	Abbreviated month name
<code>%B</code>	Full month name
<code>%c</code>	Date and time representation appropriate for locale
<code>%d</code>	Day of month as decimal number (01 - 31)
<code>%H</code>	Hour in 24-hour format (00 - 23)
<code>%I</code>	Hour in 12-hour format (01 - 12)
<code>%j</code>	Day of year as decimal number (001 - 366)
<code>%m</code>	Month as decimal number (01 - 12)
<code>%M</code>	Minute as decimal number (00 - 59)
<code>%p</code>	Current locale's A.M./P.M. indicator for 12-hour clock
<code>%S</code>	Second as decimal number (00 - 59)
<code>%U</code>	Week of year as decimal number, with Sunday as first day of week (00 - 53)
<code>%w</code>	Weekday as decimal number (0 - 6; Sunday is 0)
<code>%W</code>	Week of year as decimal number, with Monday as first day of week (00 - 53)
<code>%x</code>	Date representation for current locale
<code>%X</code>	Time representation for current locale
<code>%y</code>	Year without century, as decimal number (00 - 99)
<code>%Y</code>	Year with century, as decimal number

<code>ColeDateTime time</code>	: Reference to an object that will store the value of a field.
<code>bool bValue</code>	: Reference to an object that will store the value of a field.
<code>COleCurrency cyValue</code>	: Reference to an object that will store the value of a field.

CADORecordset::IsFieldNull

The `IsFieldNull` function determines if the field data is null.

```
BOOL IsFieldNull(LPCTSTR lpFieldName);  
BOOL IsFieldNull(int nIndex);
```

Parameters

`LPCTSTR lpFieldName` = Pointer to a string that contains the name of a field.
`int nIndex` = A zero-based index of the field in the recordset's Fields collection, for lookup by index.

Return Value

This function returns **TRUE** if the field data is Null.

CADORecordset::IsFieldEmpty

The `IsFieldEmpty` function determines if the field data is Empty.

```
BOOL IsFieldEmpty(LPCTSTR lpFieldName);  
BOOL IsFieldEmpty(int nIndex);
```

Parameters

`LPCTSTR lpFieldName` = Pointer to a string that contains the name of a field.
`int nIndex` = A zero-based index of the field in the recordset's Fields collection, for lookup by index.

Return Value

This function returns **TRUE** if the field data is Empty.

CADORecordset::IsEof

```
BOOL IsEof();
```

Return Value

This function returns **TRUE** if the current position contains no records.

CADORecordset::IsBof

```
BOOL IsBof();
```

Return Value

This function returns **TRUE** if the current position is the bottom of the recordset.

CADORecordset::MoveFirst
CADORecordset::MoveNext
CADORecordset::MovePrevious
CADORecordset::MoveLast

This functions make the First/Next/Previous/or Last record of the recordset the current record.

```
void MoveFirst();  
void MoveNext();  
void MovePrevious();  
void MoveLast();
```

CADORecordset::GetAbsolutePage
CADORecordset::SetAbsolutePage

Indicates on which page the current record resides.

```
long GetAbsolutePage();  
void SetAbsolutePage(int nPage);
```

Parameters

`int nPage` = The number of the page starting from 1.

CADORecordset::GetPageCount

`GetPageCount` returns the number of pages in the recordset.

```
long GetPageCount();
```

CADORecordset::GetPageSize
CADORecordset::SetPageSize

Indicates the number of records per page.

```
Long GetPageSize();  
void SetPageSize(int nSize);
```

Parameters

`int nSize` Set the number of records per page.

For example

```

CADORecordset pRs (&pDb);

if (pRs.Open("MyBigTable", CADORecordset::openTable))
{
    pRs.SetPageSize(5);
    for (register int nPageIndex = 1; nPageIndex <=
        pRs.GetPageCount(); nPageIndex++)
    {
        pRs.SetAbsolutePage(nPageIndex);
        for (register int nRecNumber = 0; nRecNumber <
            pRs.GetPageSize(); nRecNumber++)
        {
            long lVal;
            pRs.GetFieldValue("ID", lVal);
            pRs.MoveNext();

            if (pRs.IsEof())
                break;
        }
    }
    pRs.Close();
}

```

CADORecordset::GetAbsolutePosition**CADORecordset::SetAbsolutePosition**

Indicates the position of the record in the recordset.

```

long GetAbsolutePosition();
void SetAbsolutePosition(int nPosition);

```

Parameters

int **nPosition** = Move to the position in the recordset.

GetAbsolutePosition() can returns the position of the record or one of the following values:

- CADORecordset::positionUnknown
- CADORecordset::positionBOF
- CADORecordset::positionEOF

CADORecordset::GetFieldInfo

GetFieldInfo returns the attributes of a field.

```

BOOL GetFieldInfo(LPCTSTR lpFieldName, CAdoFieldInfo* fldInfo);
BOOL GetFieldInfo(int nIndex, CAdoFieldInfo* fldInfo);

```

Parameters

LPCTSTR **lpFieldName** = A pointer to a string that contains the name of a field.

int **nIndex** = A zero-based index of the field in the recordset's Fields collection, for lookup by index.

`CADOFieldInfo* fldInfo` = A struct that returns the field attributes.

```
struct CADOFieldInfo
{
    char m_strName[30];
    short m_nType;
    long m_lSize;
    long m_lDefinedSize;
    long m_lAttributes;
    short m_nOrdinalPosition;
    BOOL m_bRequired;
    BOOL m_bAllowZeroLength;
    long m_lCollatingOrder;
};
```

The element `m_nType` of the class `CADOFieldInfo` can be one of the following values:

- `CADORecordset::typeEmpty`
- `CADORecordset::typeTinyInt`
- `CADORecordset::typeSmallInt`
- `CADORecordset::typeInteger`
- `CADORecordset::typeBigInt`
- `CADORecordset::typeUnsignedTinyInt`
- `CADORecordset::typeUnsignedSmallInt`
- `CADORecordset::typeUnsignedInt`
- `CADORecordset::typeUnsignedBigInt`
- `CADORecordset::typeSingle`
- `CADORecordset::typeDouble`
- `CADORecordset::typeCurrency`
- `CADORecordset::typeDecimal`
- `CADORecordset::typeNumeric`
- `CADORecordset::typeBoolean`
- `CADORecordset::typeError`
- `CADORecordset::typeUserDefined`
- `CADORecordset::typeVariant`
- `CADORecordset::typeIDispatch`
- `CADORecordset::typeIUnknown`
- `CADORecordset::typeGUID`
- `CADORecordset::typeDate`
- `CADORecordset::typeDBDate`
- `CADORecordset::typeDBTime`
- `CADORecordset::typeDBTimeStamp`
- `CADORecordset::typeBSTR`
- `CADORecordset::typeChar`
- `CADORecordset::typeVarChar`
- `CADORecordset::typeLongVarChar`
- `CADORecordset::typeWChar`
- `CADORecordset::typeVarWChar`
- `CADORecordset::typeLongVarWChar`
- `CADORecordset::typeBinary`
- `CADORecordset::typeVarBinary`
- `CADORecordset::typeLongVarBinary`
- `CADORecordset::typeChapter`
- `CADORecordset::typeFileTime`
- `CADORecordset::typePropVariant`
- `CADORecordset::typeVarNumeric`
- `CADORecordset::typeArray`

For example

```
CADORecordset prs (&m_pDb);
if(prs.Open("Clients", CADORecordset::openTable))
{
    CADOFieldInfo pInfo;

    prs.GetFieldInfo("Description", &pInfo);

    if(pInfo.m_nType == CADORecordset::typeVarChar)
        AfxMessageBox("The type Description Field Is VarChar");
}

if(prs.Open("TestTable", CADORecordset::openTable))
{
    CADOFieldInfo* fInfo = new CADOFieldInfo;

    prs.GetFieldInfo(0, fInfo);
    CString strFieldName = fInfo->m_strName;
    prs.Close();
}
```

Return Value

Returns **TRUE** if was successfully.

CADORecordset::GetChunk

This function returns all, or a portion, of the contents of a large text or binary data Field object.

```
BOOL GetChunk(LPCTSTR lpFieldName, CString& strValue);
BOOL GetChunk(int nIndex, CString& strValue);
BOOL GetChunk(LPCTSTR lpFieldName, LPVOID pData);
BOOL GetChunk(int nIndex, LPVOID pData);
```

Parameters

LPCTSTR lpFieldName = A pointer to a string that contains the name of a field.

int nIndex = A zero-based index of the field in the recordset's Fields collection, for lookup by index.

CString& strValue = A string pointer that contains the data that returns from the object.

LPVOID pData = A pointer that contains the data that returns from the object.

Return Value

Returns **TRUE** if was successfully.

CADORecordset::AppendChunk

This function appends data to a large text or binary data Field.

```
BOOL AppendChunk(LPCTSTR lpFieldName, LPVOID lpData, UINT nBytes);  
BOOL AppendChunk(int nIndex, LPVOID lpData, UINT nBytes);
```

Parameters

LPCTSTR lpFieldName = A pointer to a string that contains the name of a field.
int nIndex = A zero-based index of the field in the recordset's Fields collection, for lookup by index.

LPVOID lpData = A pointer that contains the data to append to the object.

UINT nBytes = A UINT that indicates the size of the data to be inserted.

Return Value

Returns **TRUE** if was successfully.

For example

```
//Sample of AppendChunk  
prs.AddNew();  
prs.SetFieldValue("ID", 5);  
prs.SetFieldValue("Description", "Client 05");  
prs.SetFieldValue("Checker", 1);  
prs.AppendChunk("Document", "This Document is the story of Bob and his  
Friends...", 37);  
prs.Update();  
  
//Sample of GetChunk  
char data[1024];  
prs.GetChunk("Document", (LPVOID)&data);
```

CADORecordset::GetString

This function returns a recordset as a string.

```
CString GetString(LPCTSTR lpCols, LPCTSTR lpRows, LPCTSTR lpNull,  
                long numRows = 0);
```

Parameters

LPCTSTR lpCols A columns delimiter.

LPCTSTR lpRows A rows delimiter.

LPCTSTR lpNull A expression that represents a null value.

long numRows The number of rows affected.

CADORecordset::GetLastError

The `GetLastError` function returns the last error code.

```
DWORD GetLastError();
```

CADORecordset::GetLastErrorString

The `GetLastErrorString` function returns the last error string.

```
CString GetLastErrorString();
```

CADORecordset::AddNew

The `AddNew` function adds a record in the open recordset.

```
BOOL AddNew();
```

Return Value

Returns `TRUE` if was successfully.

CADORecordset::Edit

The `Edit` function allow changes to the current record in the open recordset.

```
void Edit();
```

CADORecordset::Delete

The `Delete` function deletes the current record in the open recordset.

```
BOOL Delete();
```

Return Value

Returns `TRUE` if was successfully.

CADORecordset::Update

The `Update` function updates the pending updates in the current record.

```
BOOL Update();
```

Return Value

Returns `TRUE` if was successfully.

CADORecordset::CancelUpdate

The `CancelUpdate` function cancels any pending update in the open recordset.

```
void CancelUpdate();
```

CADORecordset::SetFieldValue

The `SetFieldValue` function sets the value of a field.

```
BOOL SetFieldValue(int nIndex, int nValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, int nValue);
BOOL SetFieldValue(int nIndex, long lValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, long lValue);
BOOL SetFieldValue(int nIndex, unsigned long ulValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, unsigned long ulValue);
BOOL SetFieldValue(int nIndex, double dblValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, double dblValue);
BOOL SetFieldValue(int nIndex, CString strValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, CString strValue);
BOOL SetFieldValue(int nIndex, COleDateTime time);
BOOL SetFieldValue(LPCTSTR lpFieldName, COleDateTime time);
BOOL SetFieldValue(int nIndex, bool bValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, bool bValue);
BOOL SetFieldValue(int nIndex, COleCurrency cyValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, COleCurrency cyValue);
```

Parameters

`LPCTSTR lpFieldName` = A pointer to a string that contains the name of a field.

`int nIndex` = A zero-based index of the field in the recordset's `Fields` collection, for lookup by index.

`int nValue` = A pointer to a object containing the value of the field.

`long lValue` = A pointer to a object containing the value of the field.

`unsigned long ulValue` = A pointer to a object containing the value of the field.

`double dblValue` = A pointer to a object containing the value of the field.

`CString strValue` = A pointer to a object containing the value of the field.

`COleDateTime time` = A pointer to a object containing the value of the field.

`bool bValue` = A pointer to a object containing the value of the field.

`COleCurrency cyValue` = A pointer to a object containing the value of the field.

Return Value

Returns **TRUE** if was successfully.

For example

```
CADORecordset prs(&m_pAdoDb);
prs.Open("Test", openTable);

prs.AddNew();
prs.SetFieldValue(0, "dataU");
prs.SetFieldValue(1, "data01");
prs.SetFieldValue(2, (long)51000);
COleDateTime time = COleDateTime(2001,6,15, 10, 8, 30);
prs.SetFieldValue(3, time);
prs.Update();
```

CADORecordset::Find

The `Find` function locates a string from the current position in the open recordset using a operator of comparison.

```
BOOL Find(LPCTSTR lpFind,  
          int nSearchDirection = CADORecordset::searchForward);
```

Parameters

`LPCTSTR lpFind` = A string expression used to locate the record.

`int nSearchDirection` = A value that indicate the type of operation. The possible values are:

- `CADORecordset::searchForward` Find the next location
- `CADORecordset::searchBackward` Find the previous location

Return Value

Returns `TRUE` if was successfully.

CADORecordset::FindFirst

The `FindFirst` function locates a string from the begin in the open recordset using a operator of comparison.

```
BOOL FindFirst(LPCTSTR lpFind);
```

Parameters

`LPCTSTR lpFind` = A string expression used to locate the record.

Return Value

Returns `TRUE` if was successfully.

CADORecordset::FindNext

The `FindNext` function locates a string from the last position in the open recordset using the operator of comparison used in `FindFirst` or `Find` functions.

```
BOOL FindNext();
```

Return Value

Returns `TRUE` if was successfully.

CADORecordset::GetBookmark

The `GetBookmark` function saves the position of the current record.

```
BOOL GetBookmark ();
```

Return Value

Returns **TRUE** if was successfully.

CADORecordset::SetBookmark

The `SetBookmark` function returns to the position saved at any time.

```
void SetBookmark ();
```

Return Value

Returns **TRUE** if was successfully.

CADORecordset::SetFilter

The `SetFilter` Indicates a filter for data in a open `Recordset`.

```
BOOL SetFilter(LPCTSTR strFilter);
```

Parameters

`LPCTSTR strFilter` = a string composed by one or more individual clauses concatenated with AND or OR operators.

Return Value

Returns **TRUE** if was successfully.

CADORecordset::SetSort

The `SetSort` function sets the sort order for records in a `CADORecordset` object.

```
BOOL SetSort(LPCTSTR lpstrCriteria);
```

Parameters

`LPCTSTR lpstrCriteria` = A String that contains the ORDER BY clause of an SQL statement

Return Value

Returns **TRUE** if was successfully.

CADORecordset::GetRecordset

The `GetRecordset` function returns a pointer to an open recordset.

```
_RecordsetPtr GetRecordset ();
```

CADORecordset::GetActiveConnection

The `GetActiveConnection` returns the active connection.

```
_ConnectionPtr GetActiveConnection ();
```

CADORecordset::Clone

The `Clone` function Creates a duplicate `CADORecordset` object from an existing `CADORecordset` object.

```
BOOL Clone(CADORecordset pAdoRecordset);
```

Parameters

`CADORecordset pAdoRecordset` is an existing `CADORecordset` Object.

Return Value

Returns **TRUE** if was successfully.

CADORecordset::SaveAsXML

The `SaveAsXML` function Save the open recordset in a file with XML Format.

```
BOOL SaveAsXML(LPCTSTR lpstrXMLFile);
```

Parameters

`LPCTSTR strXMLFile` a string that indicate the complete path name of the file where the Recordset to be saved.

Return Value

Returns **TRUE** if was successfully.

CADORecordset::OpenXML

The `OpenXML` function Open a XML File Format in a recordset.

```
BOOL OpenXML(LPCTSTR lpstrXMLFile);
```

Parameters

`LPCTSTR strXMLFile` a string that indicate the complete path name of the XML file to be opened.

Return Value

Returns **TRUE** if was successfully.

The CADOPParameter Class

The `CADOPParameter` class has a set of functions that corresponds to the `_ParameterPtr`.

CADOPParameter::CADOPParameter

Creates a `CADOPParameter` object.

```
CADOPParameter(int nType, long lSize = 0, int nDirection = paramInput,
               CString strName = _T(""));
```

Parameters

`int nType` = A int value that specifies the data type of the `CADOPParameter` object. Can be one of the values specified in [CADORecordset::GetFieldInfo](#).

If are using `CADORecordset::typeNumeric` or `CADORecordset::typeDecimal`, must define the precision and scale values:

`long lSize = 0` A optional long value that specifies the maximum length for the parameter value in Bytes or characters.

`int nDirection = paramInput` = A optional int value that specifies the direction of the `CADOPParameter` object. Can be one of following values:

- | | | |
|---|---|---|
| <code>CADOPParameter::paramUnknown</code> | : | Indicates that the parameter direction is unknown. |
| <code>CADOPParameter::paramInput</code> | : | Default. Indicates that the parameter represents an input parameter. |
| <code>CADOPParameter::paramOutput</code> | : | Indicates that the parameter represents an output parameter. |
| <code>CADOPParameter::paramInputOutput</code> | : | Indicates that the parameter represents both an input and output parameter. |
| <code>CADOPParameter::paramReturnValue</code> | : | Indicates that the parameter represents a return value. |
| <code>CString strName = _T("")</code> | : | A optional string that specifies the name of the <code>CADOPParameter</code> object. |

CADOPParameter::SetValue

The `SetValue` function sets the value for the `CADOPParameter` object.

```
BOOL SetValue(int nValue);
BOOL SetValue(long lValue);
BOOL SetValue(double dbValue);
BOOL SetValue(CString strValue);
BOOL SetValue(COLEDateTime time);
BOOL SetValue(_variant_t vtValue);
```

Parameters

`int nValue` = A int value containing the parameter value.
`long lValue` = A long value containing the parameter value.
`double dbValue` = A double value containing the parameter value.
`CString strValue` = A string value containing the parameter value.
`COLEDateTime time` = A time value containing the parameter value.
`_variant_t vtValue` = A variant value containing the parameter value.

Return Value

Returns **TRUE** if was successfully.

CADOPParameter::SetPrecision

The `SetPrecision` function sets the precision for the `CADOPParameter` object.

```
void SetPrecision(int nPrecision);
```

CADOPParameter::SetScale

The `SetScale` function sets the scale for the `CADOPParameter` object.

```
void SetScale(int nScale);
```

CADOPParameter::GetValue

The `GetValue` function returns the value of the `CADOPParameter` object.

```
BOOL GetValue(int& nValue);
BOOL GetValue(long& lValue);
BOOL GetValue(double& dbValue);
BOOL GetValue(CString& strValue, CString strDateFormat = _T(""));
BOOL GetValue(COLEDateTime& time);
BOOL GetValue(_variant_t& vtValue);
```

Parameters

`int& nValue` = A reference to a `int` that will store the value of the parameter.

`long& lValue` = A reference to a `long` that will store the value of the parameter.

`double& dbValue` = A reference to a `double` that will store the value of the parameter.

`CString& strValue` = A reference to a strings that will store the value of the parameter.

`CString strDateFormat = _T("")` = A formatting time string similar to the `strftime` formatting string.

`COleDateTime& time` = A reference to a time object that will store the value of the parameter.

`_variant_t& vtValue` = A reference to a variant object that will store the value of the parameter.

Return Value

Returns **TRUE** if was successfully.

CADOPParameter::SetName

The `SetName` function sets the name of the `CADOPParameter` object

```
CString SetName(CString strName);
```

Parameters

`CString strName` A string specifying the parameter name.

CADOPParameter::GetName

The `GetName` function returns the `CADOPParameter` object.

```
CString GetName();
```

CADOPParameter::GetType

The `GetType` function returns the type of the `CADOPParameter` object.

```
int GetType();
```

CADOPParameter::GetParameter

The `GetParameter` function returns a pointer to a `_Parameter` object

```
_ParameterPtr GetParameter();
```

The CADOCommand Class

The `CADOCommand` class has a set of functions that corresponds to the `_CommandPtr`.

CADOCommand::CADOCommand

Creates a `CADOCommand` object.

```
CADOCommand(CADODatabase* pAdoDatabase, CString strCommandText = _T(""),
            int nCommandType = typeCmdStoredProc);
```

Parameters

`CADODatabase* pAdoDatabase` = A `CADODatabase` object pointer.

`CString strCommandText = _T("")` = A optional string that indicates the text of the `CADOCommand` object.

`int nCommandType = typeCmdStoredProc` = A optional int value that indicates the type of the `CADOCommand` object. Can be one of the following values:

`CADOCommand::typeCmdText` Evaluates `CommandText` as a textual definition of a command or stored procedure call.

`CADOCommand::typeCmdTable` Evaluates `CommandText` as a table name whose columns are all returned by an internally generated SQL query.

`CADOCommand::typeCmdTableDirect` Evaluates `CommandText` as a table name whose columns are all returned.

`CADOCommand::typeCmdStoredProc` Default. Evaluates `CommandText` as a stored procedure name.

`CADOCommand::typeCmdUnknown` Indicates that the type of command in the `CommandText` property is not known.

`CADOCommand::typeCmdFile` Evaluates `CommandText` as the file name of a persistently stored Recordset. Used with `Recordset.Open` or `Requery` only.

CADOCommand::AddParameter

The `AddParameter` function

```
BOOL AddParameter(CADOParameter* pAdoParameter);
BOOL AddParameter(CString strName, int nType, int nDirection,
                 long lSize, int nValue);
BOOL AddParameter(CString strName, int nType, int nDirection,
                 long lSize, long lValue);
BOOL AddParameter(CString strName, int nType, int nDirection,
                 long lSize, double dblValue, int nPrecision = 0,
                 int nScale = 0);
BOOL AddParameter(CString strName, int nType, int nDirection,
                 long lSize, CString strValue);
BOOL AddParameter(CString strName, int nType, int nDirection,
                 long lSize, COleDateTime time);
BOOL AddParameter(CString strName, int nType, int nDirection,
                 long lSize, _variant_t vtValue, int nPrecision = 0,
                 int nScale = 0);
```


Parameters

`CADOParameter* pAdoParameter` A pointer to a `CADOParameter` object

`CString strName` a string that specifies the name of the parameter.

`int nType` A `int` value that specifies the data type of the `CADOParameter` object. Can be one of the values specified in [CADORecordset::GetFieldInfo](#). If are using `CADORecordset::typeNumeric` Or `CADORecordset::typeDecimal`, must to define the precision and scale values.

`int nDirection` A `int` value that specifies the direction of the `CADOParameter` object. Can be one of the values specified in [CADOParameter::CADOParameter](#)

`long lSize` A `long` value that specifies the maximum length for the parameter value in Bytes or characters.

`int nValue` A `int` value containing the parameter value.

`long lValue` A `long` value containing the parameter value.

`double dblValue` A `double` value containing the parameter value.

`int nPrecision` A `int` value containing the precision of the parameter value.

`int nScale` A `int` value containing the scale of the parameter value.

`CString strValue` A string value containing the parameter value.

`COleDateTime time` A time value containing the parameter value.

`_variant_t vtValue` A variant value containing the parameter value.

Return Value

Returns `TRUE` if was successfully.

CADOCommand::SetText

The `SetText` function sets the command text of the `CADOCommand` object.

```
void SetText(CString strCommandText);
```

Parameters

`CString strCommandText` A string that indicates the command text.

CADOCommand::GetText

The `GetText` function returns the command text of the `CADOCommand` object.

```
CString GetText();
```

CADOCommand::SetType

The `SetType` function sets the type of the `CADOCommand` object.

```
void SetType(int nCommandType);
```

Parameters

`int nCommandType` A `int` value that indicates the type of command.

CADOCCommand::GetType

The `GetType` function returns the type of the `CADOCCommand` object.

```
int GetType();
```

CADOCCommand::GetCommand

The `GetCommand` function returns a `Command` pointer

```
_CommandPtr GetCommand();
```

CADOCCommand::Execute

The `Execute` function executes a command text.

```
BOOL Execute();
```

Return Value

Returns **TRUE** if was successfully.







CADOCCommand::GetRecordsAffected

The `GetRecordsAffected` function returns the number of records affected to the last command executed.

```
int GetRecordsAffected();
```

By Carlos Antollini

Literaturverzeichnis

-  **Oestereich, B.:** Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language, R. Oldenbourg Verlag, München, Wien, 1998
-  **Robison, K./White Lyn, D.:** Sams Teach Yourself Database Programming with Visual C++ in 21 Tagen
-  **Heuer, A./Saake, G./Sattler, K.:** Datenbanken kompakt. Mitp Verlag, Bonn, 2001
-  **Kemper, A./Eickler, A.:** Datenbanksysteme – Eine Einführung. R. Oldenbourg Verlag, München, Wien, 1996
-  **D Sight AS-2 User's Manual V.2.04,** LMI Automotive
-  **C E M Paint Defect Detecting Machine – User's Manual**

-
-  <http://www.volkswagen.com.br>
 -  <http://www.codeproject.com/database/index.asp>
 -  <http://www.itwm.fhg.de/map/projects/ABIS>
 -  <http://www.artpol-software.com/zipdoc/index.html>
 -  <http://www.microsoft.com/data/oledb/>
 -  <http://www.microsoft.com/data/ado/>

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Christos Agiovlassitis, geboren am 05.09.1975 in Petroupolis/Athen, Griechenland, ehrenwörtlich,

- (1) dass ich meine Diplomarbeit mit dem Titel:

„Erstellung eines Software-Systems für die Qualitätssicherung von Oberflächen im Automobilbereich: Datenbankaufbau und Erstellung einer Datenbankschnittstelle für die C++ Anwendung unter Verwendung von ADO“

im Betrieb Volkswagen/Audi do Brasil unter Anleitung von Professor Dr. Wolfgang Arndt selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angeführten Hilfen benutzt habe;

- (2) dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 31. Mai 2003
