HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Ing. Andreas Hölzl

Konzeption und Erstellung eines Informationssystems in einem Kunststofffenster-Produktionsbetrieb

Mittweida, 2013

Fakultät Elektro- und Informationstechnik

DIPLOMARBEIT

Konzeption und Erstellung eines Informationssystems in einem Kunststofffenster-Produktionsbetrieb

Autor:
Ing. Andreas Hölzl

Studiengang:
Informationstechnik

Seminargruppe:
KI09w2IA

Erstprüfer:
Prof. Dr.-Ing. Olaf Hagenbruch

Zweitprüfer:
Dipl.-Ing. Heiko Polster

Einreichung:
Mittweida, 31.12.2013

Verteidigung/Bewertung:
Mittweida, 2014

Bibliografische Angaben:

Hölzl, Andreas:
Konzeption und Erstellung eines Informationssystems in einem Kunststofffenster-Produktionsbetrieb - 2013 - VIII, 93, VII S., 39 Abbildungen, 14 Tabellen
Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,
Fakultät Elektro- und Informationstechnik, Diplomarbeit, 2013

Referat:

Die vorliegende Arbeit befasst sich mit der Konzeption eines Informationssystems zur Verbesserung des firmeninternen Informationsflusses beim Kunststofffenster-Produzenten Farkalux. Die momentan verwendeten, veralteten Insellösungen zur Abbildung der Geschäftsprozesse können die, an sie gestellten, Anforderungen teilweise nicht mehr erfüllen. Deshalb soll ein durchgängiges System und ein zentraler Datenspeicher entwickelt werden. Dazu werden die vorhandenen Lösungen dokumentiert, analysiert und konsolidiert.
Zur Überprüfung des Konzepts und für einen weiteren Erkenntnisgewinn wird ein ausgewählter Bereich des Systems umgesetzt.

Department of
Electro- and Informationtechnology

DIPLOMA THESIS

Conception and creation of an information system in a plastic window production plant

Author:
Andreas Hölzl

Course of studies:
Informationtechnology

Seminar group:
KI09w2IA

First examiner:
Prof. Dr.-Ing. Olaf Hagenbruch

Second examiner:
Dipl.-Ing. Heiko Polster

Submission:
Mittweida, 31th of December 2013

Defence /Evaluation:
Mittweida, 2014

Abstract:

The present work deals with the design of an information system to improve the company's internal information flow at the plastic window producer Farkalux.
Currently used systems are outdated and isolated. They no longer meet the requirements, to fully submit the business processes. Consequently, a continuous system and a central data store have to be developed. For this purpose, the solutions currently used are documented, analysed and consolidated.
To test the concept and for a further gain in knowledge, a selected area of the system is implemented too.

Danksagung:

Ich möchte mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Diplomarbeit unterstützt und motiviert haben.

Meinem Arbeitgeber, der Firma Farkalux, speziell Herrn Josef und Herrn Roland Farka, danke ich für die großzügige Unterstützung während meines Studiums.

Den Studienkollegen, vor allem aber Hans Singer, Christian Mladek und Christoph Ragonig für den Zusammenhalt, die gegenseitige Unterstützung und die Motivation.

Ein Dank gilt auch Herrn Prof. Dr.-Ing. Olaf Hagenbruch für die fachliche Unterstützung und die kritischen Anmerkungen, welche mich motivierten diese Arbeit zum vorliegenden Resultat zu bringen. In diesem Zuge danke ich auch Herrn Dipl.-Ing. Heiko Polster für das kompetente Querlesen der Arbeit.

Besonders danke ich meinen Eltern für die Unterstützung auf meinem bisherigen Lebensweg.

Mein größter Dank richtet sich an meine Familie für das Verständnis und die Nachsicht über die fehlende Zeit während des Studiums.
Meiner Frau Patricia danke ich besonders für die emotionale Unterstützung.
Bei meinen Töchtern Sarah und Sophia bedanke ich mich für das notwendige Maß an Ablenkung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADM	Außendienstmitarbeiter
AD	Active Directory
ADO	ActiveX Data Objects
API	Application Programming Interface
CLR	Common Language Runtime
CSDL	Conceptual Schema Definition Language
DBMS	Datenbankmanagementsystem
DSN	Data Source Name
EF	Entity-Framework
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
Http	Hyper Text Transfer Protokoll
ID	Identifikator
IDE	Integrated Development Environment
IIS	Internet Information Server
IT	Informationstechnologie
KMU	Klein- und Mittelunternehmen
KVP	kontinuierlicher Verbesserungsprozess
LINQ	Language Integrated Query
MAT	Mensch/Aufgabe/Technik
MSDN	Microsoft Developer Network
MSL	Mapping Specification Language
MSMQ	Microsoft Message Queuing
MVVM	Modell-View-View Modell
NTLM	NT LAN Manager

ODBC	Open Database Connectivity
OLE DB	Object Linking and Embedding Database
OR	Objektrelational
PCC	Pervasive Control Center
SBS	Small Business Server
SOA	Serviceorientierte Architektur
SQL	Structured Query Language
SSDL	Storage Schema Definition Language
URI	Uniform Resource Identifier
WAS	siehe WPAS
WCF	Windows Communication Foundation
WF	Workflow Foundation
WPAS	Windows (Process) Activation Services
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

1 Einleitung

Die Motivation dieser Arbeit ist die schwierige Lage, in der sich die Farkalux Fenster- & Elementbau GmbH bezüglich ihrer Informationstechnologie derzeit befindet. Zu Beginn wird deshalb diese Situation kurz dargestellt. Danach erfolgen die Zielsetzung und das Treffen notwendiger Definitionen. Zum Schluss dieser Einleitung folgt noch eine Kapitelübersicht als Wegfestlegung zur Zielerreichung.

1.1 Motivation

Farkalux ist ein typischer österreichischer Vertreter eines Klein- und Mittelunternehmens in der Baunebenbranche. Die Firma ist innerhalb von 30 Jahren vom Ein-Mann-Unternehmen zur aktuellen Größe mit ca. 55 Mitarbeitern angewachsen und steht momentan am Übergang vom Klein- zum Mittelunternehmen.

Abbildung 1: Farkalux Firmenzentrale

Ein Problem bei Farkalux stellen die verwendeten Informationstechnologien dar. Diese sind – wie auch das Unternehmen selbst – unstrukturiert und bedarfsorientiert gewachsen. Außerdem sind sie teilweise veraltet, die grundlegenden Applikationen für die Benutzer werden zum Großteil seit 10 bis 15 Jahren verwendet.

Diese heterogenen Systeme stellen aktuell ein Problem bei der Abwicklung der Geschäftsprozesse dar. Der Informationsfluss zwischen den notwendigen Personen und

Abteilungen findet nur unzureichend statt, weil viele Lösungen an ihre Grenzen stoßen. Zukünftige Anforderungen können nur mit unverhältnismäßig hohem Aufwand erfüllt werden.

Bei vielen Unternehmen in dieser Größenordnung wird in den letzten Jahren meist eine Umstellung der Insellösungen auf ein durchgängiges ERP-System (Enterprise Resource Planning) zur Bekämpfung ähnlicher Probleme eingesetzt. Deshalb wurde auch bei Farkalux eine Anschaffung und Einführung einer solchen Lösung geprüft. Dabei stellte sich heraus, dass der Aufwand dafür – im Verhältnis zur Firmengröße gesehen – sehr groß ist. Schuld daran sind unter anderem die breite Produktpalette sowie die Vielfalt an Geschäftsfeldern. Die Unternehmensleitung sah die Rentabilität einer solchen Investition gefährdet und kam zum Entschluss, dass die finanziellen und personellen Belastungen für eine Umstellung bereits in der ersten Stufe nicht tragbar für die Firma sind.

Die Idee ist nun die Erstellung eines selbst entwickelten Informationssystems, das sich auf die wichtigsten Funktionen für Farkalux konzentriert und deshalb kostengünstig in der Erstellung und Einführung ist, aber trotzdem die Vorteile einer zentralen Datenhaltung und einer durchgängigen Lösung wie bei einem ERP-System bietet. Der Hauptzweck der Anwendung ist die Verbesserung der vorhandenen Informationstechnologie durch Ersetzen der Insellösungen.

1.2 Zielsetzung

Das Hauptziel dieser Arbeit ist die Konzeption und teilweise Umsetzung eines Informationssystems bei Farkalux. Die Kernfrage, die dabei geklärt werden soll, lautet:

Wie kann ein Informationssystem bei Farkalux unter Berücksichtigung des speziellen betrieblichen Umfelds erstellt werden?

Ausgangslage ist der aktuelle Stand der IT (Informationstechnologie) im Unternehmen. Dieser ist jedoch nicht dokumentiert, deshalb muss als ein Teilziel eine vollständige Dokumentation der verwendeten Systeme erstellt werden.

Darauf aufbauend wird ein Konzept für ein übergreifendes IS (Informationssystem) benötigt. Dieses System muss so ausgelegt werden, dass es zukünftig die zentrale Anwendung zur Organisation der Geschäftsprozesse wird und einen unternehmensweiten Informationsfluss gewährleistet. Eine Umsetzung und Einführung des Systems soll stufenweise möglich sein, hier ist besonders auf die laufende Integrierbarkeit in den Bestand zu achten. Zu Beginn werden die Bereiche mit der höchsten Priorität für Farkalux abdeckt. Eine einfache Erweiterbarkeit fördert dann den laufenden Ausbau des IS.

Zum Prüfen des Konzepts soll ein Teilbereich davon umgesetzt werden. Diese Umsetzung dient dem Sammeln von Erfahrungen. Außerdem werden die Möglichkeiten des Systems für die Geschäftsführung und für die potentiellen Benutzer dargestellt. Die Ent-

wicklung soll einen selber auszuwählenden Bereich komplett abdecken. Dabei wird nicht ein Prototyp erstellt, sondern eine lauffähige Lösung, die bereit für die Abnahmetests ist. Die Inbetriebnahme der Lösung ist nicht Gegenstand der Arbeit – diese erfolgt erst nach Freigabe durch die Geschäftsführung.

Diese Arbeit hat als vorrangiges Ziel einen Erkenntnisgewinn. Dieser kommt durch Erarbeitung des Konzepts und durch dessen Erprobung zustande. Das gewonnene Wissen dient als Grundlage für die Entscheidung bezüglich einer Einführung. Dazu werden die Möglichkeiten eines solchen Systems aufgezeigt und die jeweiligen Aufwände und Nutzen einschätzbar gemacht.

1.3 Definition Informationssystem

Als Informationssysteme werden in dieser Arbeit sogenannte MAT-Systeme (Mensch / Aufgabe / Technik) [lutz 2011, S. 17] angesehen, die hauptsächlich der Informationsgewinnung und -verarbeitung dienen. Das System dient dabei der Interaktion zwischen diesen drei Komponenten.

Der Anwender ist hier der Mensch, im Speziellen die Mitarbeiter von Farkalux. Sie sollen vom Informationssystem für die Durchführung ihrer Aufgaben unterstützt werden.

Die Aufgabe ist die Problemstellung, in diesem Fall betriebliche Handlungsziele als Geschäftsprozesse, die mit Hilfe des Systems abgebildet werden sollen. Ein Geschäftsprozess ist eine Folge von logisch zusammenhängenden Aktivitäten, die für das Unternehmen einen Beitrag zur Wertschöpfung leisten. Er besitzt einen definierten Anfang und ein definiertes Ende, wird typischerweise wiederholt durchgeführt und ist in der Regel am Kunden orientiert [lauden 2010, S. 11].

Die Technik des MAT-Systems besteht aus Hard- und Software, im speziellen Fall aus vorhandener Hardware, Betriebssystemen und Datenbankmanagementsystemen in Zusammenarbeit mit den zu erstellenden Softwarekomponenten.

1.4 Kapitelübersicht

Im nun anschließenden Kapitel dreht sich alles um die Bestandsysteme. Diese sollen zuerst einheitlich erfasst und dokumentiert werden, danach erfolgt eine Analyse.

Das dritte Kapitel dient der Wissensfindung zu relevanten Punkten dieser Arbeit. Ziel ist ein Erkenntnisgewinn im Bereich Anwendungserstellung, Datensysteme und Datenzugriff.

Im darauf folgenden Kapitel werden die Anforderungen an das zu erstellende System gestellt. Hier werden der umzusetzende Bereich ausgewählt und die genauen Funktionen, die vom zukünftigen System erfüllt werden sollen, festgelegt.

Das fünfte Kapitel befasst sich mit der Modellierung der Daten. Hier wird die Datenbasis des Informationssystems erarbeitet und festgelegt, aufbauend auf den Erhebungen im 2. Kapitel.

Im sechsten Kapitel werden die Architektur und das Design der Software festgelegt. Hier wird das Konzept des Informationssystems definiert, sowie der Rahmen für die Umsetzung des gewählten Bereichs festgelegt.

Das Kapitel sieben widmet sich der Programmierung und Implementierung der Lösung. Dabei werden die Rahmenbedingungen, die Hauptarbeitsschritte und die auftretenden Probleme der Umsetzung näher beschrieben.

Im Anschlusskapitel dreht sich alles um das Testen der umgesetzten Lösung. Es werden die Tests definiert und die Ergebnisse aufgearbeitet.

Aus den Testergebnissen und bisherigen Erfahrungen werden im Kapitel neun Optimierungen ausgearbeitet und eventuell umgesetzt.

Das abschließende Kapitel widmet sich einer Zusammenfassung der gesamten Arbeit. Die Hauptergebnisse werden dargestellt, außerdem wird noch ein Ausblick auf eine eventuelle Fortführung des Themas gegeben.

2 Bestandsaufnahme und -analyse

In diesem Abschnitt liegt der Schwerpunkt auf der Erfassung und Beurteilung der Bestandssysteme. In der Erhebung soll die Grundlage für die nachfolgende Analyse geschaffen werden. Diese soll die größten Schwachstellen aufzeigen, damit im Kapitel 4 die Funktionen für das System festgelegt werden können.

2.1 Erhebung der Datenbestände

2.1.1 Vorstellung Farkalux

Die Firma Farkalux ist ein Unternehmen mit ca. 55 Beschäftigten. Das Hauptaufgabengebiet ist die Herstellung und der Vertrieb von Kunststofffenstern und -türen. Mit ca. 22000 hergestellten Fenstereinheiten pro Jahr gehört Farkalux zu den größten regionalen Herstellern. Am Firmensitz in Kematen in Tirol werden noch Wintergärten in der hauseigenen Tischlerei hergestellt. Zum sehr vielfältigen Produktsortiment von Farkalux gehören außerdem noch Sonnenschutzartikel, Aluminiumtüren und Terrassenüberdachungen.

Der Vertrieb der Produkte erfolgt in erster Linie durch eigene Außendienstmitarbeiter, außerdem werden Großaufträge durch die Teilnahme an Ausschreibungen lukriert. Vor allem die Kunststofffenster werden zusätzlich mittels eines Händlernetzes in Tirol vertrieben.

2.1.2 Organisationsstruktur von Farkalux

Zur besseren Einordnung der Daten wird zuerst die organisatorische Struktur von Farkalux dargestellt. Die ermittelten Datensysteme sollen dann den entsprechenden Abteilungen bzw. Produktgruppen zugeordnet werden.

Farkalux ist in folgende Bereiche bzw. Abteilungen aufgegliedert:

- Geschäftsführung
- Vertrieb
- Verwaltung
- Technik
- Montage
- Produktion
- Lager

Die Produkte von Farkalux können in folgende Hauptgruppen unterteilt werden:

- Fenster
- Sonnenschutz
- Wintergärten
- Terrassenüberdachungen

2.1.3 Ziel der Erhebung

Da Farkalux dem Tischlergewerbe entstammt und dieser Zweig dem handwerklichen Gewerbe zuzuordnen ist, sind für diese Branche typischerweise wenige automationsgestützte Arbeitsweisen in Verwendung. In den meisten Abteilungen sind von den Mitarbeitern selbst erstellte Lösungen zum Speichern und Verarbeiten der notwendigsten Daten vorhanden. Hier existieren hauptsächlich Dateien, die mit den Programmen „Excel“ bzw. „Word“ aus der Office-Suite von Microsoft – im folgendem nur mehr als Excel bzw. Word bezeichnet – erstellt wurden. Da die verwendeten Systeme allerdings nicht dokumentiert sind, soll diese Erhebung der Auflistung aller vorhandenen Datensysteme und Daten dienen. Das Hauptziel ist die vollständige und ganzheitliche Erfassung.

In diesem Fall sind nur die Daten bzw. Datensysteme von Belang. Auf Prozessabläufe wird hier nur so weit eingegangen, wie dies für das Verständnis der jeweiligen Lösung notwendig ist.

2.1.4 Durchführung der Erhebung

Sie stellt aufgrund der Firmengröße eine mittelmäßige Herausforderung dar. Es gilt hier die effizienteste Methode zu finden und keine versteckten Datenhaltungen zu übersehen. Zur Durchführung einer Ist-Analyse der Informationstechnik einer Organisation stehen prinzipiell folgende Erhebungstechniken zur Verfügung [orghb 2012, S. 79]:

- Dokumentenanalyse
- Interview
- Fragebogen
- Selbstaufschreibung
- Laufzettelverfahren
- Multimomentaufnahme
- Zeitaufnahme
- Analytischen Schätzen
- Workshop

Da für dieses Projekt die Erhebung des Datenbestandes von Belang ist, wird die Erfassung mittels der Dokumentenanalyse und Interviews durchgeführt. Die Beurteilung der generellen Organisationsstruktur ist hier nicht von Bedeutung.

Die Ergebnisse der Erhebung pro Abteilung sind im Anhang - Teil 1 ersichtlich. Mehrfachaufzählungen wurden dabei bewusst durchgeführt, um die abteilungsübergreifende Nutzung der einzelnen Lösungen aufzuzeigen.

Eine Zusammenfassung aller Systeme bei Farkalux ist im Abschnitt der Analyse in Tabelle 4 ersichtlich.

2.1.5 Zusammenfassung

Die zentrale Datenhaltung bei Farkalux wird mit der Fensterbau-Software Adulo durchgeführt. Hier gibt es eine Schnittstelle zur Finanzbuchhaltungssoftware, deshalb sind alle Aufträge sämtlicher Produktgruppen ab dem Prozessschritt Rechnungslegung vertreten. Bei früheren Prozessschritten sind weniger Produktgruppen enthalten, bei Angebotslegung gibt es nur mehr die Produktgruppe der Fenster.

Die Bereiche, die nicht in der Fensterbau-Software implementiert sind, werden größtenteils mit Excel realisiert. Hier gibt es eine große Anzahl an teilweise sehr unterschiedlichen Lösungen.

Im Bereich des Angebotswesens gibt es zusätzlich viele Word-Dokumente. Hier wird die Erstellung mit vorgefertigten Textblöcken, den sogenannten Autotexten, beschleunigt.

2.2 Analyse der Datensysteme

2.2.1 Ziel der Analyse

Die Erhebung hat die unterschiedlichsten Systeme bei Farkalux aufgezeigt. Die Analyse baut auf diese Daten auf und soll die Vor- und Nachteile der vorhandenen Systeme aufzeigen. Hauptziel ist das Aufdecken von Schwachstellen in der IT von Farkalux. Es geht ebenfalls darum, Optimierungspotenziale der bestehenden IT-Prozesse zu finden.

Die Erhebung der Datensysteme hat gezeigt, dass hauptsächlich Daten in Excel-Sheets und der Fensterbau-Software „Adulo“ vorliegen. Deshalb sollen diese beiden Hauptlösungen zuerst näher untersucht werden.

2.2.2 Analyse der Fensterbau-Software

Die Branchenlösung Adulo in der Version 11.9h ist eine verteilte Datenbankanwendung. Als DBMS (Datenbankmanagementsystem) dient ein Pervasive SQL-Server Version 10.1 SP1, dieser läuft auf einem Windows Server 2008 R2 Standard mit SP1. Laut Herstellerangaben ist die momentan aktuellste Version 11.10c, die verwendete Version ist ungefähr 3 Monate alt. Eine Aktualisierung auf die neueste Version findet bei Farkalux jährlich statt.

Auf den Clients existieren komplette Programminstallationen, serverseitig gibt es das DBMS, eine Clientinstallation und einen Dienst zum Ausführen der Stapeldruckjobs. Das Programm ist auf beinahe allen Desktops installiert, nur an 3 Arbeitsplätzen gibt es keine Installation.

Alle von Farkalux lizenzierten Module sind in Abbildung 2 ersichtlich. Folgende Module decken teilweise Funktionen ab, die aktuell anderweitig realisiert werden:

- Provisionsabrechnung
- Serienfertigung
- Fertigmeldung
- FIT (Fertigungsinformationstechnologie)

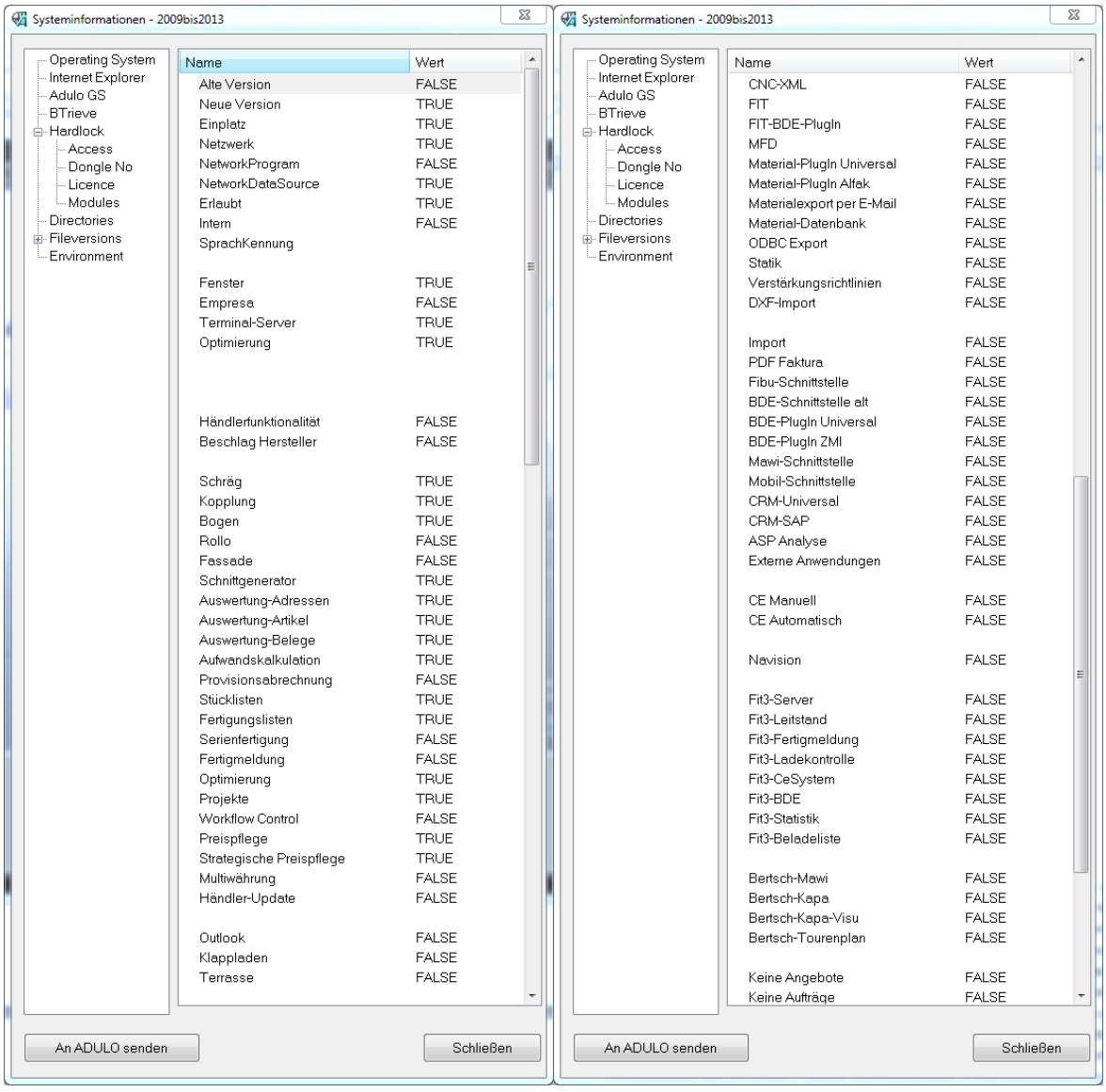


Abbildung 2: Übersicht der Module von Adulo

Die Module Serienfertigung, Fertigmeldung und FIT betreffen die Fensterproduktion bzw. die Steuerung derselben. Hier hat eine genauere Betrachtung im Jahr 2011 den Einsatz

der Module bei Farkalux als unwirtschaftlich qualifiziert. Die Auftragsübersichtsliste aller Fertigungsaufträge deckt diese Bereiche größtenteils ab.

Mit der Softwarelösung Adulo werden folgende Daten zentral gespeichert:

- die Kundenadressdaten zu den Aufträgen aller Produktgruppen
- die Adressdaten aller Angebotskunden für Fenster
- alle Rechnungen
- Lieferscheine und Aufträge im Bereich Fenster, Sonnenschutz und Terrassenüberdachung
- Angebote im Bereich Fenster
- die Artikeldaten sämtlicher für die Fensterproduktion notwendigen Artikel
- Preislisten für Fenster und Fensterzubehör
- die Maschinensteuerungsdaten der Produktionsmaschinen
- Vorlagen für Belegformulare, Stücklisten und Auswertungen

2.2.3 Analyse der Excel-Lösungen

Einer der größten Vorteile eines Excel-Sheets ist sicherlich die Vielseitigkeit und die relativ schnelle Erstellungszeit einer Lösung. Am Beispiel der Daten-Erhebung wird klar deutlich, dass sehr unterschiedliche Zwecke mit diesem Programm erfüllt werden. Von reinen Formularen zum Ausfüllen und Ausdrucken, über klassische Tabellen, Diagramme, Kalkulationen, Zeitaufzeichnungen und sogar Terminplanungen wie in einem Kalender reicht das Spektrum der abgedeckten Problemstellungen. Diese Vielseitigkeit beim Erstellen bringt aber leider den Nachteil der unterschiedlichsten Implementationen mit sich. Solch heterogene Umsetzungen erschweren das Zusammenführen auf eine gemeinsame Datenbasis erheblich.

Excel-Lösungen sind eine gute Wahl, wenn es um Daten geht, die hauptsächlich für einen Benutzer von großem Interesse sind. Hier sind die Zeiterfassungen der Techniker ein gutes Beispiel. Sie werden immer von derselben Person bedient, die Auswertung der Zeitaufteilung wird einmal im Jahr vorgenommen – diese Daten werden manuell in die Berechnung des Prämiensystems übertragen.

Die Erfahrung bei Farkalux hat gezeigt, dass Excel-Listen, die ständig von mehreren Bearbeitern mit Daten gefüllt werden, nur unzulänglich funktionieren. Da Excel in Tabellen keine Eingabe-Verifizierung durchführt, bleiben teilweise notwendige Felder leer oder es können ungültige Werte eingegeben werden.

Ein weiterer großer Nachteil von Excel-Lösungen ist die Abgeschlossenheit eines jeden Excel-Sheets. Verknüpfungen zu Daten aus Tabellen von anderen Dateien können zwar erstellt werden, allerdings sind diese Verknüpfungsinformationen statisch abgespeichert. Dieser Umstand fördert einen Datenwildwuchs – die verteilten Datenhaltungen werden nicht gegenseitig abgeglichen. Dies stellt für den Betrieb ein Risiko dar.

Eine Möglichkeit, die Nachteile der verteilten Daten zu mindern und trotzdem die flexiblen Lösungen zu nutzen, ist der Zugriff auf externe Daten in Excel-Sheets. Hier können Daten von einer zentralen Stelle wie zum Beispiel dem SQL-Server (Structured Query Language) von Microsoft mit gewissen Einschränkungen eingebunden werden.

2.2.4 Prioritätenbewertung

Als ersten Schritt der vergleichenden Analyse muss eine Möglichkeit gefunden werden, die Relevanz der einzelnen Systeme für das gesamte Unternehmen zu beurteilen. Es ist eine Bewertung hinsichtlich der Prioritäten notwendig. Diese Reihung soll als zusätzliche Entscheidungsgrundlage für das Erarbeiten neuer Lösungen dienen.

Um diese Analyse durchzuführen, werden Prioritätskriterien eingeführt. Für jedes Kriterium werden eine maximale Punkteanzahl und allenfalls notwendige Zwischenstufen festgelegt. Das Verhältnis der maximalen Punktezahl zur maximalen Gesamtpunkteanzahl ergibt die Gewichtung des jeweiligen Kriteriums. Die Summe der erreichten Punkte pro Datensystem ist ein Maß der Priorität.

Die Punkte für die Umsatzwertung ergeben sich durch Summierung der Einzelpunkte für jede – von der jeweiligen Lösung betroffene – Produktgruppe. Die Einzelpunkte werden anhand des folgenden Umsatzschlüssels festgelegt:

Fenster und Türen	75 %
Sonnenschutz	13 %
Wintergarten	9 %
Terrassenüberdachungen	3 %

Tabelle 1: Umsatzanteile der Produktgruppen 2012

Die komplette Auflistung aller Lösungen und der erreichten Punkte ist in Tabelle 2 und Tabelle 3 ersichtlich.

Etwas überraschend ist das Erreichen der meisten Prioritätspunkte für die Mail- und Kalenderverwaltung. Bei genauerer Betrachtung wird dies allerdings klarer. Die Mail- und Kalenderfunktionen werden von allen Mitarbeitern mit PC-Arbeitsplätzen für sämtliche Produktgruppen benutzt, der Zugriff darauf erfolgt sehr häufig und ein Verlust der Daten wäre unternehmenskritisch, weil viele Geschäftsfälle per Mail abgewickelt werden.

Prioritätenanalyse

Gewichtung in % Max. Punkte		19,2% 50	36,5% 95		23,1% 60		15,4% 40		5,8% 15		Summen 100,0% 260
Bezeichnung	**Daten kritisch**		**Produktgruppe**	**Anteil**	**Anzahl der Benutzer**		**Häufigkeit der Zugriffe**		**Aktualitäts-notwendigkeit**		**Summe**
Vorgabe für Teilbewertungen	für Unternehmen teilw. / für Produktgr. eher unkritisch	50 25 0	Fenster Sonnenschutz Wintergarten Überdachung	50 20 15 10	über 20 11 bis 20 5 bis 10 3 bis 4 2 1	60 48 36 24 12 0	stündlich täglich wöchentlich monatlich jährlich	40 30 20 10 0	sofort stündlich täglich wöchentlich monatlich jährlich	15 12 9 6 3 0	
Mail- und Kalenderverwaltung	Exchange und Outlook	50	95		60		40		12		257
adulo	Anwendung	50	95		48		40		15		248
Zeitconsens	Anwendung	25	95		60		40		15		235
Finanzbuchhaltung Orlando	Anwendung	50	95		0		40		15		200
ADM Verkaufsauswertung	Excel	0	95		48		40		9		192
Glasbestellungen	Excel	25	60		48		40		12		185
Angebote	Word	25	95		24		30		9		183
CPU Lohn	Anwendung	50	95		0		30		6		181
Produktionsübersicht Fenster	Excel	25	50		48		40		15		178
Montageplan	Excel	25	50		48		40		12		175
Übersichtsliste Fensteraufträge	Excel	25	60		36		40		12		173
Elba	Anwendung	25	95		0		30		9		159
Elda	Anwendung	25	95		0		30		9		159
Projektordner	Dateifreigabe	25	50		24		40		12		151
Ausschreibungsliste	Excel	25	70		12		30		9		146
KVP	Excel	0	95		12		20		6		133
Sonnenschutzmontage	Exchange Öffentl. Kal.	25	20		36		40		12		133
Deckungsbeitragsauswertung	Excel	25	95		0		10		3		133
Montagestundenerfassung	Excel und VBA	25	50		12		30		9		126
Innenfensterbankliste	Excel	0	50		36		30		9		125
Außenfensterbankliste	Excel	0	50		36		30		9		125
Bestellübersicht Aluüren	Excel und VBA	0	50		36		30		9		125
Angebotssummen	Excel	0	95		0		20		10		125
Telefonliste	Excel	0	95		0		10		10		115
Halbfertigen Liste	Excel	25	50		12		20		6		113

Tabelle 2: Analyse der Prioritäten Teil 1

Gewichtung in % Max. Punkte			19,2% 50		36,5% 95		23,1% 60		15,4% 40		5,8% 15		100,0% 260
Bezeichnung	**Daten kritisch**			**Produktgruppe**	**Anteil**		**Anzahl der Benutzer**		**Häufigkeit der Zugriffe**		**Aktualitäts-notwendigkeit**		**Summe**
Vorgabe für Teilbewertungen	für Unternehmen	50		Fenster	50		über 20	60	stündlich	40	sofort	15	
	teilw. / für Produktgr.	25		Sonnenschutz	20		11 bis 20	48	täglich	30	stündlich	12	
	eher unkritisch	0		Wintergarten	15		5 bis 10	36	wöchentlich	20	täglich	9	
				Überdachung	10		3 bis 4	24	monatlich	10	wöchentlich	6	
							2	12	jährlich	0	monatlich	3	
							1	0			jährlich	0	
Sonstige Verkaufsauswertung	Excel	0			50		36		20		6		112
Provisionabrechnung	Excel	25			70		0		10		3		108
Gealan Profile Gesamtübersicht	Excel	25			50		0		20		6		101
Prämiensystem Produktion	Excel	25			50		0		20		6		101
Prämiensystem Montage	Excel	25			50		12		10		3		100
Glaspreislisten	Excel	0			50		36		10		3		99
Zeiterfassung Techniker	Excel	0			50		12		30		6		98
Adresssammlungen Serienbriefe	Access	0			85		12		0		0		97
Montage unbezahlte Stunden	Excel	0			50		12		30		3		95
Wintergärten Projektordner	Dateifreigabe	25			15		12		30		9		91
Montageartikelbeschaffung	Excel	0			60		0		20		9		89
Wintergartenwochenplanung	Excel	25			15		12		30		6		88
Montage Soll Ist Auswertung	Excel	0			50		12		20		3		85
Gealan Bestellung	Excel	0			50		0		20		6		76
Überdachungen Projektordner	Dateifreigabe	25			10		0		30		9		74
Mindestbestand Lagerprofile	Excel	0			50		0		20		3		73
Inventurliste Lager	Excel	0			60		12		0		0		72
diverse Bestelllisten Produktion	Excel	0			50		0		10		6		66
Visus	Anwendung	0			25		12		20		9		66
Inventurliste Produktion	Excel	0			50		12		0		0		62
Überdachungen-Kalkulation	Excel	0			10		12		20		15		57
Wintergartenkalkulation	Excel	0			15		0		20		15		50
Übersichtsliste Überdachungen	Excel	0			10		0		30		9		49

Tabelle 3: Analyse der Prioritäten Teil 2

2.2.5 Analyse nach Kriterien

Die Analyse der Datensysteme wird mittels der Nutzwertanalyse durchgeführt. Die Grundlage dieser ist der subjektive Wertbegriff. Mittels dieser Methode können auch sehr unterschiedliche Systeme nach Vergleichskriterien bewertet werden. Als Ergebnis erhält man eine vergleichende Bewertung, wobei nur die Reihenfolge und nicht die erreichte Punktezahl ein Maß darstellen.

Im ersten Schritt werden die Zielkriterien festgelegt, die für den Vergleich herangezogen werden. Danach erfolgt die Gewichtung dieser Kriterien. Zum Schluss wird die Bewertung der einzelnen Systeme nach den Kriterien durchgeführt [schulte 2011, S. 235].

Die größte Herausforderung liegt in der Findung geeigneter Analysekriterien, die bei allen Systemen gleichermaßen angewendet werden können. Generell gilt für jede Software, dass gewisse Anforderungen erfüllt werden müssen, die nicht in direktem Zusammenhang mit der eigentlichen Aufgabe stehen. Diese nicht funktionalen Anforderungen, oft auch Randbedingungen genannt, stellen Analysekriterien dar. Dies sind folgende Kriterien [geirhos 2011, S.39]:

- Performance
- Skalierbarkeit
- Sicherheit
- Benutzerfreundlichkeit
- Erweiterbarkeit
- Kompatibilität
- Robustheit / Ausfallssicherheit oder -wahrscheinlichkeit
- Supportbarkeit
- Dokumentation

Benutzer wurden in den Interviews für die Erhebung bereits zu Problemen bzw. Unzulänglichkeiten befragt. Die meistgenannten Schwierigkeiten können mit folgenden Kriterien zusammengefasst werden:

- Durchgängigkeit des Systems
- Datenhaltung zentral / lokal / verteilt
- Art der Speicherung
- Gleichzeitiger gemeinsamer Zugriff möglich
- Datenübergabe automatisch / halbmanuell / manuell

Folgende Kriterien werden ebenfalls in die Analyse aufgenommen:

- Aktualität der Daten sofort / stündlich / täglich / wöchentlich / monatlich / jährlich
- Datenredundanz (sind dieselben Daten mehrfach vorhanden)
- Zugriffsberechtigungen steuerbar

- Vollständigkeit der Daten
- Prozessbezogene Durchgängigkeit der Daten vorhanden
- Lösung vom jeweiligen Benutzer adaptierbar und erweiterbar
- Informationen der Lösung ausreichend
- Einschulungsaufwand
- Erweiterbarkeit gering / mittel / hoch
- Komplexität der Lösung gering / mittel / hoch
- Bedienkomfort der Lösung gering / mittel / hoch

Mit diesen Kriterien kann jede Datenhaltung bei Farkalux abgeglichen werden. Um eine Wertung zu erhalten, müssen die jeweiligen Kriterien gewichtet werden. Die Gewichtung der einzelnen Sparten erfolgt über die maximal mögliche Punktezahl pro Kriterium. Wird diese in Verhältnis zu der Gesamtpunktezahl aller Kriterien gesetzt, erhält man einen prozentualen Anteil, der die Gewichtung darstellt.

Aus Gründen der Übersichtlichkeit wird die Analyse in funktionale und nicht funktionale Anforderungen aufgeteilt. Die erhaltenen Punkte pro Wertung werden in einer Zusammenfassungsliste (siehe Tabelle 4) addiert. Die Einzelauswertungen sind im Anhang A ersichtlich. Das Verhältnis von funktionalen zu nichtfunktionalen Anforderungen ist aufgrund der Punktefestlegung mit 53 % zu 47 % festgelegt.

2.2.6 Analyse ausgewählter Lösungen

Herausragend ist die Tatsache, dass die Lösung mit der höchsten Priorität die zweitmeisten Punkte erreicht. Die Mail- und Kalenderverwaltung mit Microsoft Exchange und Microsoft Outlook als Clientprogramm stellt somit eine sehr gute Wahl für Farkalux dar.

Wie erwartet schneiden Anwendungen bei der Analyse besser ab als die selber erstellten Excel Lösungen. Auffallend sind die hohe Anzahl an Excel-Sheets, welche die Funktionen Auftragsabwicklung und Bestellung bzw. Beschaffung abdecken. Diese Prozesse sind für das Unternehmen sehr wichtig, können allerdings mit den bestehenden Technologien nicht gut genug organisiert werden.

Besonders problematisch ist die Produktionsübersicht der Fensteraufträge. Sie hat eine hohe Ausfallswahrscheinlichkeit – laut Produktionsleiter traten bereits häufig Speicherprobleme auf. Hier besteht akuter Handlungsbedarf, stellt dieser Excel-Sheet doch die Grundlage der Produktionsplanung dar. Aufgrund der Komplexität der durchgeführten Berechnungen und der nicht vorhanden Dokumentation wäre hier eine Aufteilung der Liste in einen Produktionssteuerungsbereich und in einen Auswertungsbereich sinnvoll. Der, für den laufenden Betrieb wichtige, erste Teil würde deutlich an Komplexität verlieren und wäre dadurch stabiler. Der zweite fehleranfälligere Teil wäre nicht mehr so kritisch für die Produktionsabteilung.

Bezeichnung	Art der Lösung	Prioritäts-punkte	Platz Priorität	funkt. Punkte	nichtfunkt. Punkte	Summe
Elba	Anwendung	159	12	156	130	286
Mail- und Kalenderverwaltung	Exchange / Outlook	257	1	141	139	280
CPU Lohn	Anwendung	181	8	153	120	273
Elda	Anwendung	159	13	156	115	271
Finanzbuchhaltung Orlando	Anwendung	200	4	128	115	243
Zeitconsens	Anwendung	235	3	153	88	241
adulo	Anwendung	248	2	144	96	240
Zeiterfassung Techniker	Excel	98	32	103	87	190
Sonnenschutzmontage	Exchange Kalender	133	17	87	91	178
Visus	Anwendung	66	44	83	88	171
KVP	Excel	133	16	123	48	171
Überdachungen Projektordner	Dateifreigabe	74	40	103	66	169
Projektordner	Dateifreigabe	151	14	111	56	167
Wintergärten Projektordner	Dateifreigabe	91	35	98	66	164
Ausschreibungsliste	Excel	146	15	91	61	152
ADM Verkaufsauswertung	Excel	192	5	68	79	147
Montage Soll Ist Auswertung	Excel	85	38	64	79	143
Montageartikelbeschaffung	Excel	89	36	68	74	142
Inventurliste Lager	Excel	72	42	74	66	140
Übersichtsliste Überdachungen	Excel	49	48	70	66	136
Montagestundenerfassung	Excel und VBA	126	19	78	57	135
Prämiensystem Montage	Excel	100	30	74	59	133
Inventurliste Produktion	Excel	62	45	66	66	132
Deckungsbeitragsauswertung	Excel	133	18	79	53	132
Montageplan	Excel	175	10	72	53	125
Wintergartenwochenplanung	Excel	88	37	68	56	124
Bestellübersicht Aluüren	Excel und VBA	125	22	58	64	122
Mindestbestand Lagerprofile	Excel	73	41	54	66	120
Telefonliste	Excel	115	24	57	61	118
Sonstige Verkaufsauswertung	Excel	112	26	53	64	117
Prämiensystem Produktion	Excel	101	29	61	56	117
Provisionsabrechnung	Excel	108	27	44	71	115
Montage unbezahlte Stunden	Excel	95	34	55	59	114
Übersichtsliste Fensteraufträge	Excel	173	11	58	56	114
Adresssammlung Serienbriefe	Access	97	33	37	69	106
div. Bestelllisten Produktion	Excel	66	43	46	56	102
Innenfensterbankliste	Excel	125	20	46	51	97
Außenfensterbankliste	Excel	125	21	46	51	97
Glasbestellungen	Excel	185	6	56	41	97
Gealan Profile Gesamtübersicht	Excel	101	28	54	36	90
Gealan Bestellung	Excel	76	39	58	28	86
Produktionsübersicht Fenster	Excel	178	9	42	43	85
Überdachungen-Kalkulation	Excel	57	46	43	41	84
Wintergartenkalkulation	Excel	50	47	43	41	84
Halbfertigen Liste	Excel	113	25	44	36	80
Angebote	Word	183	7	49	31	80
Glaspreislisten	Excel	99	31	27	43	70
Angebotssummen	Excel	125	23	17	28	45

Tabelle 4: Analyse der Datensysteme nach Kriterien

Viele Excel-Lösungen werden auch als Reports – vor allem für den Vertrieb – eingesetzt. Auf die ADM-Verkaufsauswertung (Außendienstmitarbeiter), die fünfthöchste Priorität besitzt, muss hier etwas näher eingegangen werden. Bei dieser Auswertung handelt es sich um eine Erfassung aller Aufträge und den dazugehörigen Auswertungen in Form von Tabellen und Diagrammen. Die Erfassung wird dabei manuell durchgeführt. Auftragskennzahlen, die im Fensterbauprogramm bereits enthalten sind, müssen hier noch einmal angeführt werden.

Ein weiterer Problembereich ist das Angebotswesen der Produktgruppen Sonnenschutz, Wintergärten und Terrassenüberdachungen. Diese werden mit Word verfasst und als Einzeldokumente abgelegt. Die so erzeugten Daten können momentan nicht in weiterführenden Prozessschritten verwendet werden.

Ein weiterer Schwachpunkt ist der sogenannte Montageplan. Bei diesem besteht das Hauptproblem in der fehlenden Datenübergabe. Hier wäre zumindest eine halbautomatisierte Datenübergabe von unschätzbarem Vorteil – der Montageleiter würde somit alle notwendigen Informationen viel früher erhalten.

Die Angebotssummen, die am wenigsten Punkte erreicht haben, sind eine Auswertung von Angebotsdaten. Wenn diese Daten an einer zentralen Stelle zusammengefasst wären, könnte diese Auswertung von einer Software automatisch durchgeführt werden.

3 Grundlagen zur Anwendungserstellung

Um den bisher beschriebenen Problemen bei Farkalux zu begegnen, soll ein individuelles Informationssystem erstellt werden. Ziel dieses Kapitels ist deshalb der Erkenntnisgewinn in den Bereichen Anwendungserstellung (vor allem im Enterprise-Umfeld) und Datenzugriff mittels Anwendungen. Deshalb werden in diesem Kapitel die Grundlagen einer solchen Anwendungsentwicklung aufgezeigt. Mögliche Technologien werden angeführt, um eine Konkretisierung der Aufgabenstellung und anschließend eine gezielte Auswahl in den Kapiteln 5 und 6 zu ermöglichen.

Als Grundlage zur Programmierung im Windows-Umfeld wird der aktuellste Stand des .NET-Frameworks aufgezeigt und mit dem ADO.NET Entity-Framework eine aktuelle Form des Datenzugriffs vorgestellt. Für die Datennutzung des Bestandsystems Pervasive werden ebenfalls alle verfügbaren Arten ausgearbeitet.

Mit der Windows Workflow Foundation wird eine sehr junge Technik zur Modellierung und Implementierung von (langläufigen) Geschäftsprozessen gezeigt. Am Ende des Kapitels erfolgt noch eine Einführung in die Enterprise-Library, die häufig benötigte Funktionen und Möglichkeiten bereitstellt.

3.1 .NET-Framework

.NET bezeichnet die von Microsoft entwickelte Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. .NET ist auf verschiedenen Plattformen verfügbar und unterstützt die Verwendung einer Vielzahl von Programmiersprachen. .NET-Programme werden zunächst in eine Zwischensprache (Common Intermediate Language) übersetzt, bevor sie von der Laufzeitumgebung ausgeführt werden. Diese Übersetzung geschieht in der Regel mithilfe eines Just-In-Time-Compilers [wiki 2013].

Das .NET-Framework implementiert die Common Language Infrastructure. Wesentliche Bestandteile des .NET-Frameworks sind die verschiedenen Sprachen, die für die Software-Entwicklung verwendet werden können, die Framework Class Library (FCL), eine gemeinsame Klassenbibliothek, die zum Teil als API (Application Programming Interface) vorhandene Dienste des jeweiligen Betriebssystems adaptiert, und die gemeinsame virtuelle Laufzeitumgebung CLR (Common Language Runtime), in der die Programme ausgeführt werden.

Die Version 4.0 des .NET-Frameworks stellt eine eigenständige Entwicklung dar, die nicht auf andere Versionen (wie z. B. 3.5 auf 3.0 +SP1) aufbaut, sondern für sich steht. Große

Teile wurden sogar komplett neu implementiert – allerdings ohne zu riskieren, dass bestehender .NET-Code nicht mehr läuft [louis 2013, S.38].

Die seit Ende 2012 verfügbare Version 4.5 ist hingegen wieder eine reine Erweiterung der Version 4.0 (so wie z. B. schon Version 3.0 und 3.5 Erweiterungen des .NET-Frameworks 2.0 waren). Die wichtigste Neuerung ist sicherlich die Unterstützung für die Erstellung von Windows Store-Apps (siehe auch Abbildung 3). Diese Unterstützung drückt sich allerdings nicht in APIs aus, sondern vielmehr in Form von Profilen, Projektionsschichten und Schnittstellen. Sie gestatten dem Code, der in einer .NET-Sprache geschrieben wurde, auf die neue WinRT (Windows Runtime) zuzugreifen [louis 2013, S. 38].

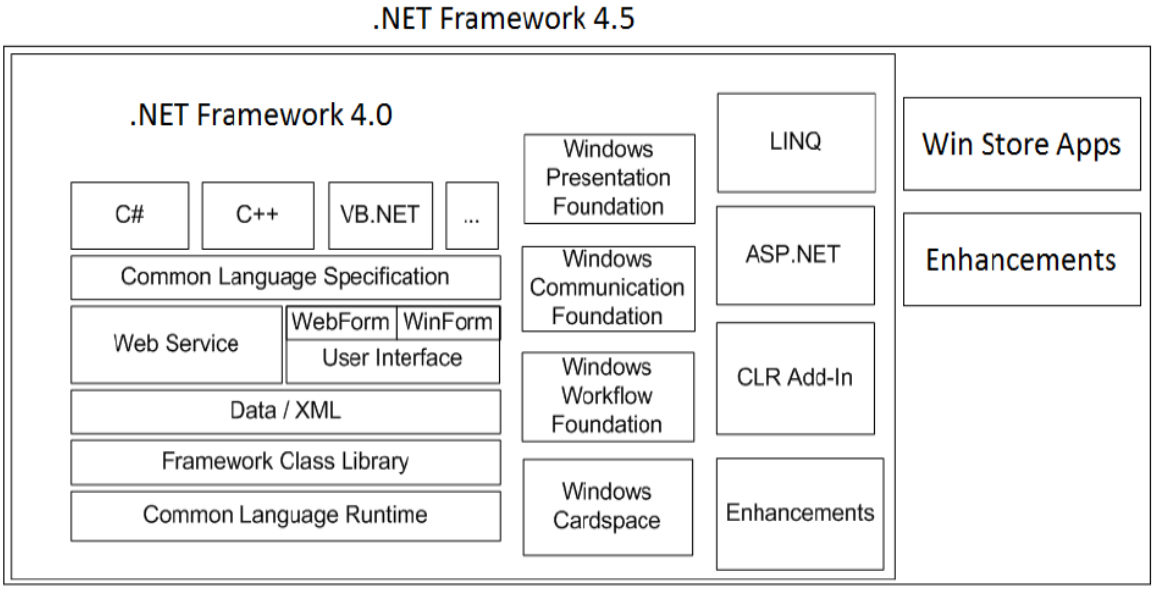


Abbildung 3: Aufbau .NET-Framework 4.5 (Quelle: MSDN)

3.2 ADO.NET Entity-Framework

Das ADO.NET Entity-Framework ist neben LINQ to SQL einer der beiden möglichen O/R-Mapper im .NET-Framework. Ein objektrelationaler Mapper ist eine Verbindung zwischen Anwendung und Datenbank, er stellt also gewissermaßen eine Zwischenschicht dar. Der Vorteil gegenüber LINQ to SQL liegt darin, dass im EF (Entity-Framework) die größte Unabhängigkeit zwischen Daten- und Objektmodell besteht. Eine Tabelle kann auf mehrere Klassen aufgeteilt sein, aber es können auch mehrere Tabellen in einer Klasse zusammengefasst werden [vergl. panther 2012, S. 289].

Das EF besteht aus dem konzeptionellen Modell, dem Speichermodell und der Zuordnungsschicht (siehe Abbildung 4). Das konzeptionelle Modell nutzt die Schema-Definitionssprache CSDL (Conceptual Schema Definition Language) zur Beschreibung der Objektstruktur. Das Speichermodell (auch logisches Modell) verwendet die Datenspeicherschema-Definitionssprache SSDL (Storage Schema Definition Language) und stellt die Daten am Datenbankserver dar. Die Zuordnungsschicht stellt die Zuordnung

zwischen Tabellen und Objekten dar, dies geschieht mithilfe der MSL (Mapping Specification Language) [vergl. panther 2012, S. 289].

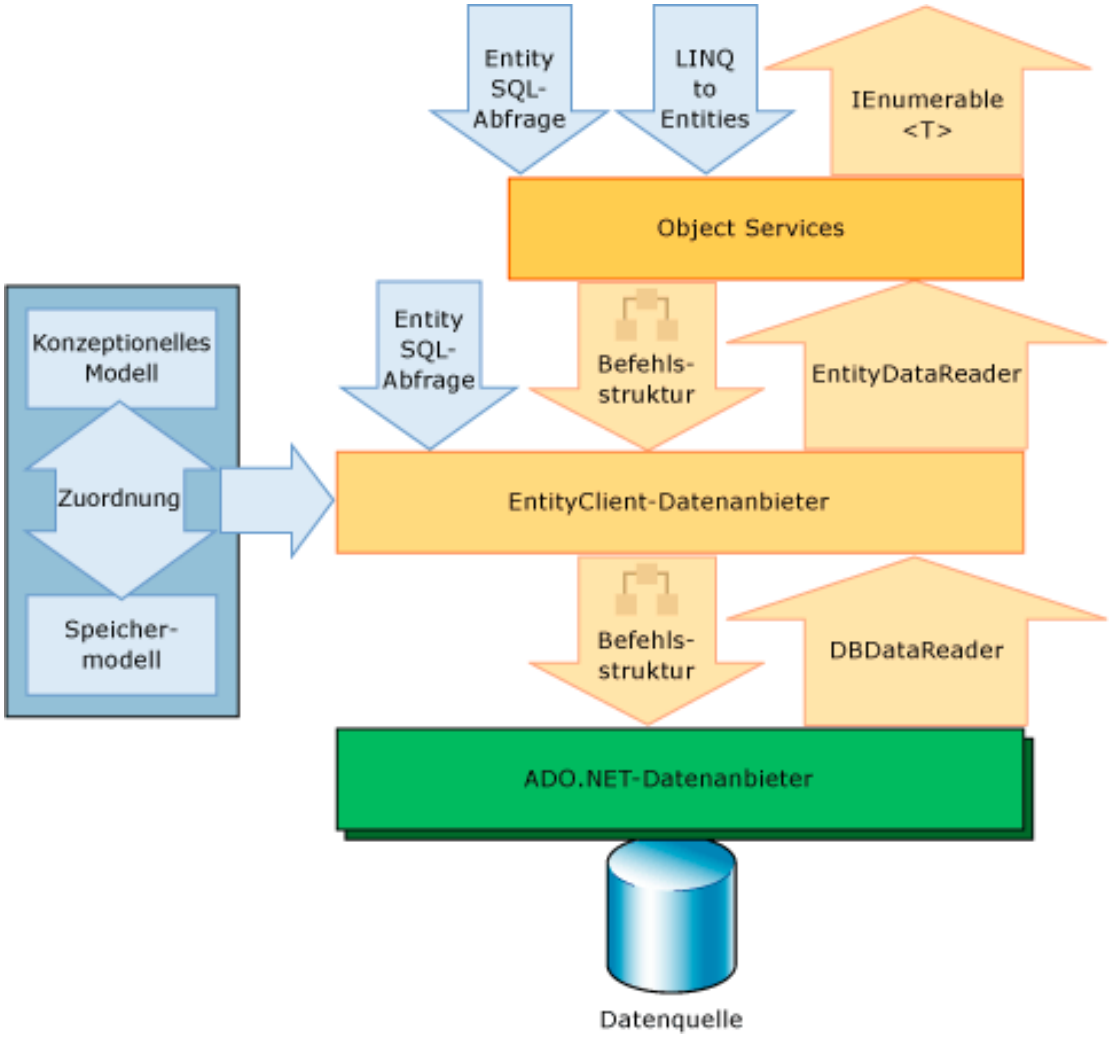


Abbildung 4: Architektur des ADO.NET Entity-Frameworks (Quelle: MSDN)

Beim Erstellen eines Entitäten-Modells gibt es aktuell 3 verschiedene Ansätze [kansy 2013, S.459 und S. 536].

3.2.1 Database First

Bei diesem Modell startet der Entwurf auf klassische Weise mit der Erstellung der Datenstrukturen in einem DBMS. Danach werden diese Datenstrukturen ins Modell importiert und daraus werden die Klassen erzeugt. Dieser Ansatz eignet sich besonders, bei der Verwendung von bestehenden Datenbanken. Optimierungen von Hand an der Datenbank sowie eine von der Anwendung unabhängige Erweiterung der Datenbank sind bei diesem Ansatz möglich.

3.2.2 Modell First

Hier startet der Entwurf mit dem Entitäten-Modell, daraus werden dann die entsprechenden Tabellen und auch Klassen erzeugt. Dieser Entwurf eignet sich besonders für neue Projekte, die unabhängig von bestehenden Datenstrukturen sind. Der Vorteil liegt in der grafischen Gestaltungsmöglichkeit und in der Loslösung des Entwickelns von physischen Datenstrukturen. Bei komplexen Datenstrukturen ist dieser Ansatz ebenfalls gut geeignet, hier könnten die erzeugten Klassen vom Database-First-Ansatz nicht brauchbar sein.

3.2.3 Code First

Bei Code First (auch „Code only“ genannt) werden gewöhnliche Klassen erzeugt und mit Attributen ausgezeichnet. Das EF erzeugt mithilfe dieser Informationen die Modelle und Zuordnungen. Diese Variante hat Vorteile, wenn die Datenbank nur Mittel zum Zweck ist, die volle Kontrolle über die Klassen benötigt wird oder aber bestehende Klassen nachträglich datenbanktauglich gemacht werden sollen [geirhos 2013, S. 682].

3.3 Datenanbindung Pervasive-SQL

Die aktuellste Version des Pervasive PSQL DBMS ist die Version Vx mit Service Pack 3 [pervasive1]. Laut den aktuellen Systemanforderungen von Adulo [adulo1] wird der Pervasive PSQL-Server in der Version 11.x SP 3 als aktuellste Variante unterstützt. In der Version 11.x stehen prinzipiell die Zugriffsmethoden laut Abbildung 5 zur Verfügung [pervasive2, S. 2].

Für eine .NET-Anwendung können dabei die Methoden Btrieve API, ADO (ActiveX Data Objects) und ODBC (Open Database Connectivity) verwendet werden.

3.3.1 Btrieve API

Beim Zugriff auf die Daten per Btrieve API werden die Funktionen der API-Bibliotheken, die als Dll-Dateien vorliegen, direkt aufgerufen. Dies kann von nahezu jeder Programmiersprache aus erfolgen. Dieser Zugriff ist sehr schnell (kürzester Weg zwischen Daten und Anwendung) und es steht der komplette Funktionsumfang des DBMS zur Verfügung. Diese Art der Lösung benötigt für den Zugriff auf die Daten am wenigsten Code, allerdings muss die Anwendung gewisse Managementfunktionen übernehmen, was den zu erzeugenden Code wieder vergrößern kann. Der Zugriff über Internet oder Intranet ist nicht direkt möglich. Die Einarbeitungszeit und der Programmieraufwand sind relativ hoch, ebenso wird die Komplexität der Anwendung erhöht.

Access Method	Description	Used For
Btrieve (transactional interface)	Original Btrieve API	Creating Btrieve database applications
ADO (Microsoft IDEs)	High-level visual or code-based programming	Visual programming of transactional or relational (SQL) applications. This is the recommended programming interface for Microsoft development environments.
PDAC (Embarcadero IDEs)	Pervasive Direct Access Components for Delphi and C++ Builder	Replaces functionality of Embarcadero data-aware components and eliminates need for Embarcadero Database Engine.
ODBC (relational)	Microsoft's Open Database Connectivity	Creating SQL-based applications
Java Class Library	Java Class Library for transactional interface data access.	Creating Java-based applications that connect to the transactional interface
JDBC	Implementation of Sun's Java Database Connectivity	Creating JDBC-based SQL applications using an industry-standard API.
Distributed Tuning Interface (DTI)	Pervasive's API for monitoring and administration	Performing administrative and utility functions from applications, creating and maintaining Data Dictionary Files from applications
Distributed Tuning Objects (DTO)	Pervasive's object-oriented programming interface for monitoring and administration	Performing administrative and utility functions from applications, creating and maintaining Data Dictionary Files from applications

Abbildung 5: Pervasive Zugriffsmethoden (Quelle: pervasive.com)

3.3.2 ADO

Beim Zugriff mittels ADO kann zwischen den Varianten ADO/OLE DB (Object Linking and Embedding Database) und der neueren Variante ADO.NET unterschieden werden.

3.3.2.1 ADO/OLE DB

OLE DB ist eine von Microsoft entwickelte Programmierschnittstelle für den Zugriff auf unterschiedliche Datenquellen. Die Unterstützung dafür ist in der IDE (Integrated Development Environment) Visual Studio gut integriert. Der Zugriff kann auch über Internet/Intranet erfolgen. Die Daten werden in Form von sogenannten Rowsets geliefert, die Felder werden abstrahiert. Die Performance und der Programmieraufwand sind allerdings aufgrund der älteren Technologie etwas schlechter.

3.3.2.2 ADO.NET

Der Zugriff per ADO.NET bietet einen guten Kompromiss zwischen Performance und Aufwand, sowie die beste Unterstützung für eine sehr effizient skalierbare Architektur. Die Verbindung kann auch über Intranet/Internet hergestellt werden. XML (Extensible Markup Language) wird ebenfalls unterstützt. Mit dem Update Release Service Pack 3 vom Mai 2013 wird der Datenprovider für ADO.NET in der Version 4.0 unterstützt. Dieser beinhaltet folgende Funktionen [pervasive3]:

- Unterstützung für das .NET-Framework 4.5
- Unterstützung für das Entity-Framework 4.0
- Unterstützung des OR-Mappers (Objektrelationaler Mapper) im Visual Studio 2012

3.3.3 ODBC

Beim Zugriff auf die Daten per ODBC erfolgt der Zugriff mittels SQL, dadurch wird der Zugriff auf die Daten für die Anwendung komplett abstrahiert. Ein Austausch des Datenproviders ist somit problemloser möglich. Da diese Art der Schnittstelle eine sehr allgemeine Schnittstelle darstellt, kann nicht der volle Funktionsumfang des DBMS ausgeschöpft werden. Der Zugriff kann aber durch viele Anwendungen und Programmiersprachen erfolgen. Die Performance ist schlechter als der Zugriff per Btrieve API. Dieser Zugriff eignet sich besonders für Anwendungen, die möglichst unabhängig vom Datenprovider sein sollen.

3.4 Workflow Foundation

Für langlebige Objekte, die Geschäftslogik abbilden sollen, wurde von Microsoft die Workflow Foundation geschaffen. Bei dieser handelt es sich um ein erweiterbares Framework zur Entwicklung von programmgesteuerten, komplexen Arbeitsabläufen bzw. Prozessen in der Visual-Studio-Designumgebung. Sie existiert seit der Version 3.0 in der .NET-

Klassenbibliothek, wurde in Version 4.0 zu großen Teilen neu überarbeitet und ist in der aktuellen Version 4.5 des .NET-Frameworks um viele Features ergänzt worden [geirhos 2013, S. 731].

Die WF (Workflow Foundation) stellt eine eigene API und eine grafische Oberfläche zum Editieren der Workflows für den Programmierer zur Verfügung.

„Bei einem Workflow handelt es sich um die grundlegenden Aufgaben, Verfahren, Personen und Organisationen, den Ein- und Ausgang von Systeminformationen, Richtlinien und Regeln sowie Werkzeuge, die für die einzelnen Schritte des Geschäftsprozesses benötigt werden“ [scribner 2007, S.3]. Ein Workflow ist also gewissermaßen das Modell eines Prozesses, der aus einer Abfolge von Aktivitäten besteht. Diese Aktivitäten können Verzweigungen, Entscheidungen, einzelne Aktionen oder aber auch ganze Workflows sein und sind somit beliebig schachtelbar. Sie stellen einen einzelnen Schritt im Ablauf dar. Der gesamte Workflow beschreibt die Regeln, Aktionen und möglichen Zustände des Prozesses. Das Hauptaugenmerk liegt mit diesem Framework in der Darstellung der Prozesse und nicht mehr in deren direkten Programmierung, es stellt also eine Abstraktionsebene dar.

Ein großer Vorteil ist die Möglichkeit, laufende Workflows zu unterbrechen und in einer Datenbank zu sichern (zu persistieren), damit diese zu einem späteren Zeitpunkt wieder (automatisch) ausgeführt werden können. Vor allem bei lang laufenden Workflows werden dadurch in der Wartephase keine Ressourcen bezüglich Speicher und Prozessor verbraucht. Die asynchrone Kommunikation, wie sie in Geschäftsprozessen oft üblich ist, wird durch die WF ebenso besonders stark unterstützt.

Vorteile der Workflow Foundation sind die schnellere Anpassung an geänderte Prozesse, die bessere Übersicht aufgrund der Workflowdiagramme und schnellere Implementierung von Geschäftssoftware. Nachteile sind mögliche Performanceeinbußen aufgrund zusätzlicher Verwaltungsaufgaben und Mehraufwand in der Umsetzung aufgrund der noch nicht so großen Anzahl an eingebauten Aktivitäten.

Die wichtigsten Neuerungen in der Version 4.5 sind die Versionsverwaltung, neue Aktivitäten, Vertrag-zuerst-Entwurf und Zustandsautomat-Workflows (vergleiche [msdnwf1]).

3.5 Enterprise-Library

Die Enterprise-Library stellt einen Satz an wiederverwendbaren Softwarekomponenten dar, die dort Application Blocks genannt werden. Entwickelt wurden sie ursprünglich von der Microsoft Patterns & Practises Group. Sie können in eigenen Projekten unentgeltlich genutzt werden [geirhos 2013, S. 224].

In Abbildung 6 sind die enthaltenen Application-Blocks ersichtlich. Sie decken einige grundlegende Anforderungen für immer wiederkehrende Aufgaben im Bereich der Entwicklung von Enterprise-Software ab.

Application Block	Beschreibung
Caching	Damit können lokale Caches in Anwendungen implementiert werden. Der Cache-Inhalt kann dabei auch persistiert werden.
Cryptography	Mit diesem Application Block lassen sich Daten ver- und entschlüsseln, besonders praktisch ist das beispielsweise für das Speichern von Zugangspasswörtern.
Data Access	Dieser Application Block baut auf ADO.NET auf und bietet eine vereinfachte Schnittstelle zum Abrufen und Speichern von Daten.
Exception Handling	Erlaubt den flexiblen Umgang mit Exceptions, zum Beispiel deren Logging, der Benachrichtigung von Anwendern und dem Ersetzen von Exceptions.
Logging	Bietet Standardfunktionalität zum Loggen an.
Policy Injection	Damit können Policys automatisch den Objektinstanzen zugewiesen werden, die sich dann um Querschnittsaufgaben kümmern, zum Beispiel der Validierung übergebener Parameter.
Security	Bietet Standardfunktionalität zum Implementieren eigener Sicherheitsrichtlinien in Anwendungen.
Unity	Mithilfe dieses Application Blocks steht ein IoC (Inversion of Control)-Container bereit, der Constructor-, Property- und Method-Injection beherrscht.
Validation	Damit können Sie Validierungsregeln definieren, gegen die Geschäftsobjekte geprüft werden.

Abbildung 6: Application Blocks der Enterprise-Library [geirhos 2013]

Der große Vorteil der Enterprise-Library ist die einfache Anwendung und die weitgehende Konfigurierbarkeit. Diese wird zusätzlich durch eine eigene Konfigurationsoberfläche unterstützt.

Die Enterprise-Library kann als eigenständiger Download oder mittels der NuGet-Paket-Verwaltung in Visual-Studio-Projekte eingebunden werden.

4 Anforderungsanalyse

Aufbauend auf den Erhebungen in Kapitel 2 wird in diesem Abschnitt die Aufgabenstellung präzisiert. Dazu werden die Ziele des gesamten Systems definiert, für das in den nachfolgenden Kapiteln ein Konzept entwickelt werden soll. Außerdem werden, nach Auswahl eines Bereichs für die Umsetzung, die genauen Anforderungen diesbezüglich ausgearbeitet.

4.1 Ziele des gesamten Systems

Wie bereits in der Einführung erwähnt, lautet die Kernfrage dieser Arbeit:

Wie kann ein Informationssystem bei Farkalux unter Berücksichtigung des speziellen betrieblichen Umfelds erstellt werden?

Um diese Frage beantworten zu können, soll in den folgenden Kapiteln ein Konzept für ein solches Informationssystem ausgearbeitet werden.

Die Ziele dieses Systems lauten dabei:

- Das Informationssystem soll eine einheitliche, durchgängige Anwendung sein.
- Es muss die momentan vorhandenen Bereiche Angebotswesen, Auftragsabwicklung und Bestellverwaltung abdecken.
- Es soll in allen Abteilungen verwendet werden können.
- Die vorhandenen Systeme sollen konsolidiert oder ersetzt werden.
- Das IS muss an die steigenden Anforderungen des Unternehmens anpassbar und einfach erweiterbar sein.
- Es muss die bestehende Fensterbau-Software Adulo integrieren.
- Es muss eine zentrale, relationale Datenbank verwenden.
- Es muss verschiedene Berechtigungsstufen und die entsprechende Authentifizierung vorsehen.
- Die Entwicklung und Einführung muss stufenweise möglich sein.
- Die Entwicklung muss für die vorhandene IT-Abteilung machbar sein.

Das Konzept soll hauptsächlich beschreiben, wie die vorhandenen Systeme konsolidiert werden können. Das heißt in erster Linie sollen die vorhandenen Funktionen vom Informationssystem abgedeckt werden.

Das Konzept soll als Grundlage für eine Anwendungsentwicklung dienen. Dafür muss es folgende Punkte beinhalten:

- Welche Datenbank wird verwendet?
- Welche Daten sind in welchen Systemen?
- Wie erfolgt der Zugriff auf die Daten?
- Wie hat die Architektur der Anwendung auszusehen?
- Welche Systeme sollen bestehen bleiben und wie werden sie eingebunden?
- Welche Bereiche werden abgedeckt?

4.2 Auswahl des Teilbereichs zur Umsetzung

Parallel zum Ausarbeiten des Konzepts für das IS soll ein Teil davon umgesetzt werden. Dies dient zum Sammeln von Erfahrungen, sowie zum Darstellen der Vorteile einer Anwendung gegenüber den vorhandenen Excel-Applikationen. Laut den Zielvorgaben des vorhergehenden Abschnitts stehen dabei die Bereiche Angebotswesen, Auftragsabwicklung und Bestellverwaltung zur Verfügung.

In den Interviews und in der Analyse sind die meisten Probleme in Verbindung mit der Abwicklung der Kunststofffensteraufträge festzustellen. Da dieser Bereich auch das Kerngeschäft von Farkalux darstellt, bietet sich hier eine genaue Ausarbeitung im Zuge einer Umsetzung besonders an.

Das Hauptziel des Systems zur Auftragsabwicklung soll die Reduktion auf nur mehr eine zusätzliche, durchgängige Anwendung zur Auftragsbegleitung sein. In der Fensterbau-Software sollen die technischen und auftragsspezifischen Daten vorhanden sein, im IS Statusinformationen des aktuellen Prozesses. Die momentan verwendeten Teilsysteme sollen zur Gänze ersetzt werden. Um dieses Ziel zu erreichen, wird zuerst der Prozess der Auftragsabwicklung definiert. Dieser soll in Zukunft laut Abbildung 7 durchgeführt werden.

4.3 Anforderungsermittlung Auftragsabwicklung

4.3.1 Problemstellung

Momentan werden bei Farkalux zur Auftragsabwicklung unterschiedliche Systeme in den Bereichen Verkauf, Technik, Produktion, Montage und Verwaltung zur Unterstützung der Prozessabwicklung bei Fensteraufträgen verwendet. Diese Systeme arbeiten nur teilweise miteinander, somit ist der Informationsfluss mangelhaft. Dies soll mit einem einheitlichen Informationssystem geändert werden.

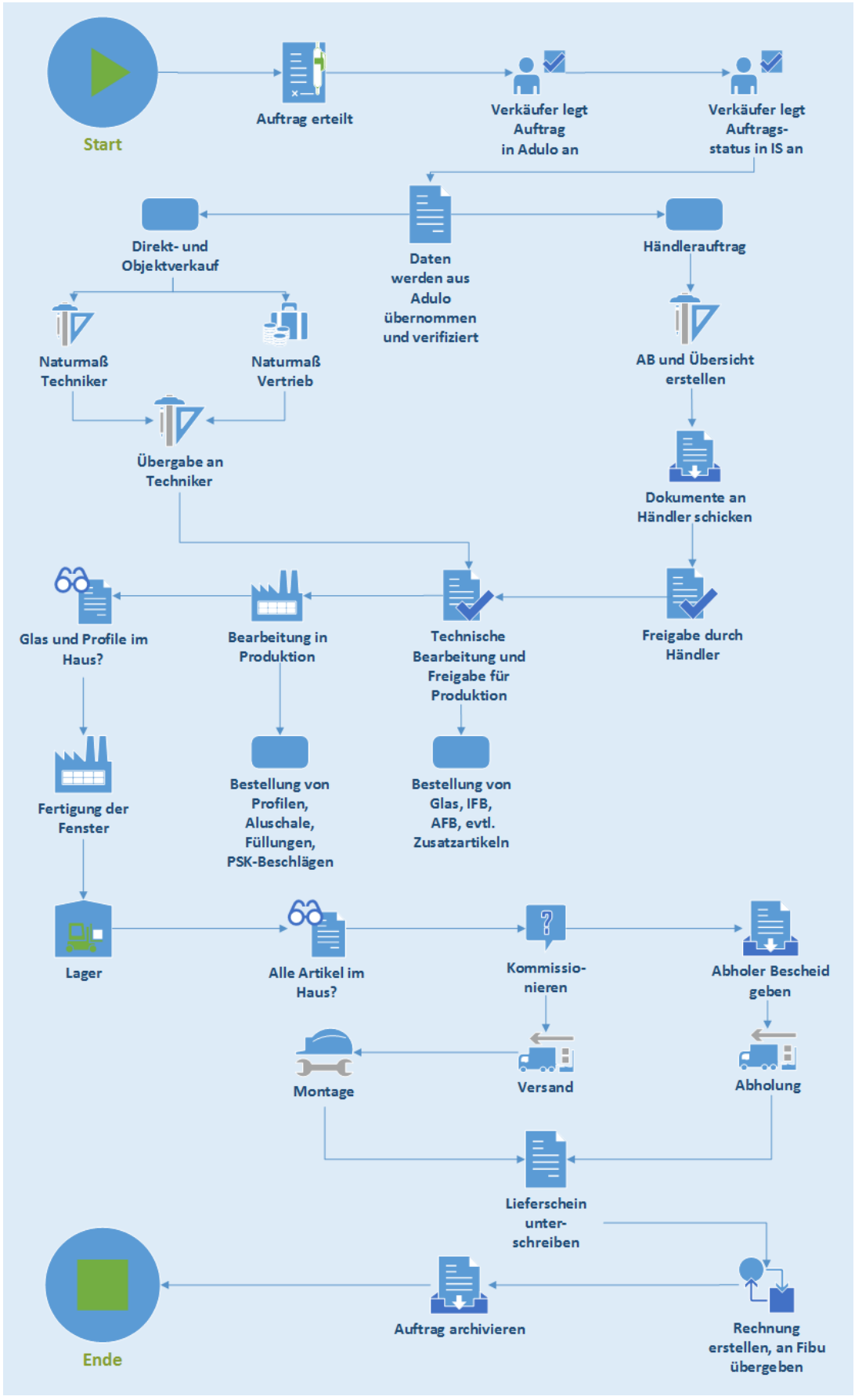


Abbildung 7: Prozess Auftragsabwicklung

4.3.2 Ziele der Anwendung

Mit dem IS soll der Prozess der Auftragsabwicklung von der Anlage des Auftrags im System bis zur Abrechnung unterstützt werden. Das System muss die Statusinformationen für jeden Auftrag in einer Übersicht bereithalten und die jeweils nächsten Schritte pro Auftrag vorgeben.

Zur Unterstützung in den oben erwähnten Bereichen ist es notwendig, dass die Übersicht erhöht wird und Informationen vom System zentral bereitgestellt werden. Das Hauptziel ist die schnelle Informationsbereitstellung für die jeweiligen Benutzer!

Die Kenndaten der Aufträge werden vom Produktionssystem Adulo übernommen. Die Produktionsdaten dürfen jedoch nicht verändert werden, damit nicht die Integrität des Fensterbausystems gefährdet wird.

4.3.3 Funktionale Anforderungen

Bei Auftragserteilung durch den Kunden wird der Auftragsbeleg vom Verkäufer in Adulo angelegt. Danach wird der neue Auftragsstatus mit dem Datensatz in der Fensterbau-Software verbunden. Dazu soll im IS eine Auswahl aus allen noch nicht verbundenen Aufträgen erfolgen.

Der Status eines Auftrages kann vom jeweiligen Benutzer geändert werden, dabei sollen vom System nur die möglichen nächsten Schritte vorgegeben werden.

Für den Produktionsleiter muss eine Funktion existieren, die alle neuen Aufträge in einer Auflistung anzeigt, die noch nicht in das Produktionssystem übernommen worden sind. Diese Liste muss im Format .xls oder .csv exportierbar sein, damit sie in die Produktionssteuerung übernommen werden kann. Der Aufbau der zu erzeugenden Datei soll analog der vorhandenen Projektliste erfolgen. Nach dem erfolgreichen Export wird das Datum „In Produktion übernommen“ gesetzt.

Der Montageleiter benötigt ebenfalls eine Anzeige aller neu zu übernehmenden Aufträge. Als Informationen sind alle Daten analog der Informationen in der Montageliste anzuzeigen. Die Übernahme der neuen Aufträge muss ebenfalls bestätigt werden!

Das IS muss den Produktions-, den Montage- und den Endtermin des Auftrags verwalten können. Termine bzw. Änderungen müssen für die Produktion bzw. Montage ersichtlich sein.

Die Anzeige der gesamten Auftragsliste sollte ähnlich aufgebaut sein wie in der Software Adulo. Eine Suche nach Aufträgen muss als Volltextsuche möglich sein (in den Feldern: Auftragsnummer, Kundenname, Kommission, Lieferort).

Die Verwaltung kann den Auftrag abschließen, indem er als fertig gestellt markiert wird.

Alle Anwendungsfälle der Auftragsverwaltung können laut Use-Case-Diagramm in Abbildung 8 zusammengefasst werden.

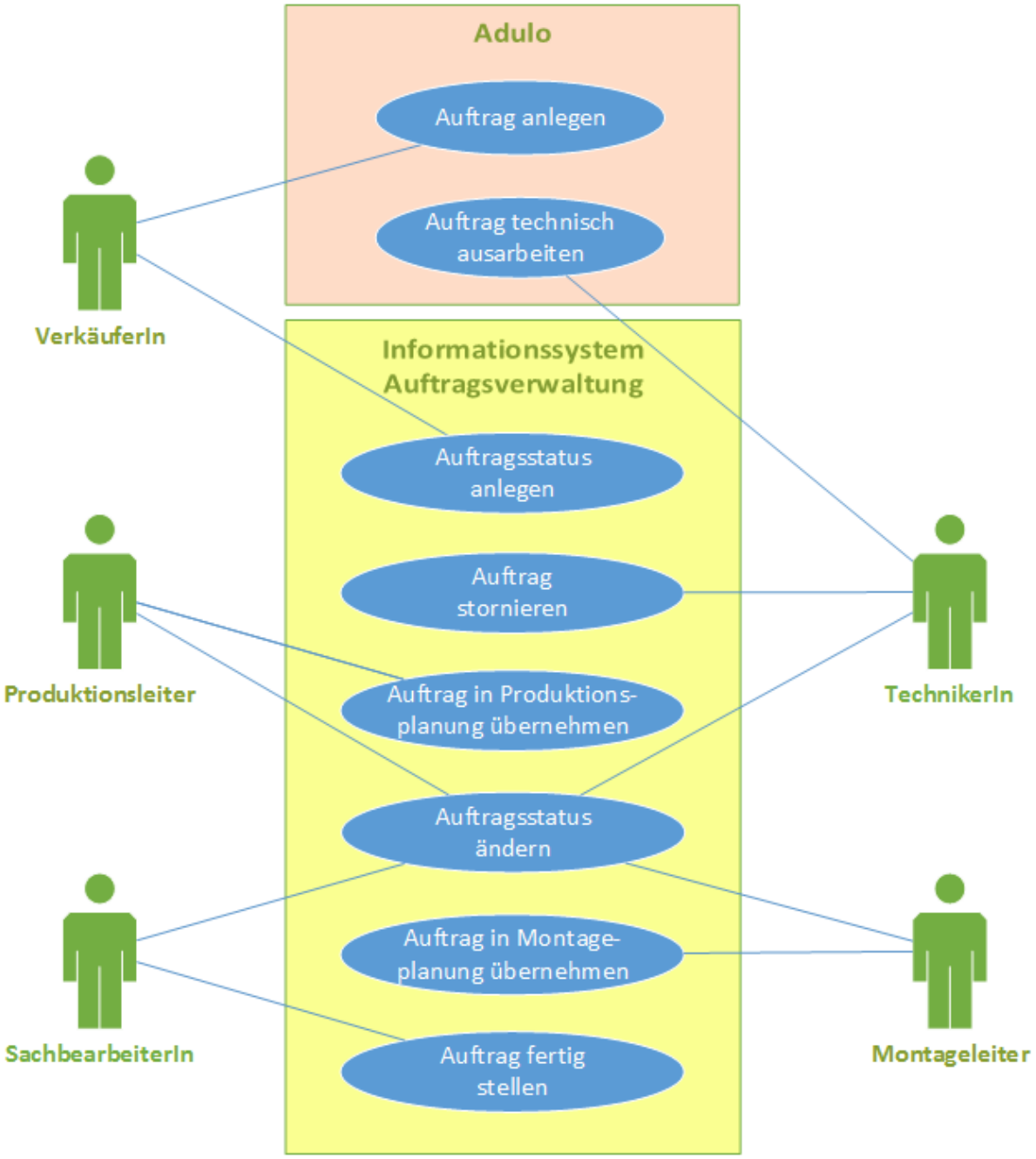


Abbildung 8: Use-Case-Diagramm der Auftragsabwicklung

4.3.4 Nichtfunktionale Anforderungen

Die bestehende Fensterbau-Software und die zugehörigen Daten dürfen nicht verändert werden!

Die grundlegende Datenhaltung muss einfach erweiterbar sein und für eine schrittweise Umstellung auch einen zentralen Zugriffspunkt für Office-Anwendungen ermöglichen.

Die Software wird als Windows-Anwendung (speziell als .NET-Anwendung) ausgelegt.

Eine einfache Anpassung des Systems an geänderte Arbeitsabläufe soll ebenso möglich sein wie das Hinzufügen von zusätzlichen Workflows (wie zum Beispiel Aufträgen mit Zukaufartikeln).

Die Autorisierung der Benutzer hat rollenbasiert zu erfolgen, der Zugriff auf die Montage- und Produktionsfunktionen soll nur mit entsprechenden Berechtigungen möglich sein.

Der Verlauf eines Auftrags muss mitgeloggt werden und jederzeit (auch nach Abschluss) einsehbar sein. Generell soll der Grad an Logging-Informationen konfigurierbar sein (Kritisch, Fehler, Warnung, Information und Debugging).

4.3.5 Zielumgebung

Die Zielumgebung besteht aus einem Windows SBS-Netzwerk (Small Business Server 2011 Standard SP1), für die Anwendung stehen zwei Applikations-Server (Windows 2008 R2 Standard SP1) zur Verfügung.

Die Produktionsdatenbank von Adulo befindet sich auf dem Server ‚far-app1'. Das DBMS ist ein Pervasive PSQL v10.3 Server.

Auf dem Applikationsserver ‚far-app2' ist bereits ein SQL-Server 2012 Express Edition im Einsatz, dieser sollte nach Möglichkeit verwendet werden.

5 Modellierung der Datenstrukturen

Dieses Kapitel beschäftigt sich mit den Daten des Informationssystems, diese sollen zentral bereitgestellt werden.

Zuerst wird festgelegt, in welchen Systemen die Daten gespeichert werden. Danach erfolgen die Festlegung des DBMS und des zentralen Zugriffs darauf. Mit der Bestimmung der zusätzlichen Daten und der Erstellung des Entitäten-Relationen-Modells für das System zur Auftragsabwicklung endet dieses Kapitel.

5.1 Verwenden von zentralen Datenspeichern

Ein Hauptziel des IS ist das Schaffen eines zentralen Datenspeichers, der von den geplanten Anwendungen verwendet werden kann. Um dieses Ziel zu erreichen, müssen die Informationen in den Zielbereichen Angebotswesen, Auftragsabwicklung und Bestellverwaltung entsprechend strukturiert gespeichert werden. Als Speichersysteme eignen sich laut Vorgabe entweder die Fensterbau-Software Adulo oder eine relationale Datenbank.

5.1.1 Änderungen im Angebotswesen

In der Angebotslegung befinden sich die Daten im Bereich der Fenster bereits in Adulo. Die Bereiche Sonnenschutz, Wintergarten und Terrassenüberdachung verwenden Word-Dokumente in einer Dateifreigabe als Speicherlösung.

Diese Bereiche sollen in Zukunft ebenfalls Adulo als Anwendung zum Speichern von Angeboten nutzen. Dadurch werden eine gemeinsame Datenbasis und auch eine einheitliche Vorgehensweise beim Erstellen sichergestellt.

Ein weiterer Vorteil wäre der Start der Prozesskette in der Fensterbau-Software. Da Sonnenschutzaufträge ebenfalls in Adulo erfasst werden, könnte diese Neuerfassung entfallen. Es müsste nur das jeweilige Angebot übernommen und allenfalls überarbeitet werden. Gleiches gilt für Wintergärten und Terrassenüberdachungen.

Adulo besitzt die Möglichkeit, Textbausteine zu hinterlegen. Diese könnten ähnlich wie die Autotexte in Word verwendet werden. Da es sich bei den Sonnenschutzartikeln größtenteils um einzelne Artikel handelt, können diese zu den vorhandenen Stammdaten hinzugefügt werden. Eine Prüfung der Möglichkeiten in Adulo hat ergeben, dass Artikel mit den dazugehörigen Texten und Preislisten angelegt werden können. So kann die Software den Preis der jeweiligen Anlage selber kalkulieren. Diese Möglichkeit wäre eine enorme Verbesserung der derzeitigen Arbeitsweise. Das manuelle Preisbestimmen durch den

ADM müsste nicht mehr durchgeführt werden – die Angabe der Kalkulationsparameter wäre ausreichend. Eine Erstellung des Angebotes im Programm durch den ADM selber ist am zweckmäßigsten – die Weitergabe an eine Sachbearbeiterin könnte entfallen.

5.1.2 Änderungen in der Auftrags- und Bestellverwaltung

In diesem Bereich werden momentan hauptsächlich Excel-Sheets verwendet. Vor allem in der Auftragsverwaltung ist ein Großteil der Auftragskenndaten, die in den externen Dateien gespeichert sind, in der Fensterbau-Software bereits vorhanden. Das heißt, mit der Schaffung einer Schnittstelle zu den Excel-Lösungen könnte die doppelte Datenhaltung vermieden werden. Hier könnte zum Beispiel ODBC (Open Database Connectivity) zur Verbindung mit der Pervasive-Datenbank verwendet werden. Mit dieser Datenbindung könnten die vorhandenen Auftragsdaten in Adulo immer aktuell in den jeweiligen Excel-Sheets verwendet werden. Für die vielen bestehenden Auswertungen wäre eine solche Anpassung bereits ausreichend.

Vor allem in der Bestellverwaltung wäre eine zentrale Anwendung wichtig, weil hier keine einheitliche Vorgehensweise im Unternehmen anzutreffen ist. Die notwendigen Daten dafür sollten, ebenso wie die Daten der Auftragsverwaltung, in einer eigenen Datenbank abgelegt werden.

5.2 Festlegung der Datenhaltung

Zur Schaffung eines zentralen Zugriffspunkts müssen die Daten in der Fensterbau-Software und die vorhin angeführten Informationen in einer Datenbank zusammengefasst werden. Die Daten des Fensterbausystems werden, wie bereits ausgeführt, mit dem Pervasive PSQL-Server v10.1 SP 1 gespeichert. Sie sollen in dieser Form bestehen bleiben.

Für alle zusätzlichen Informationen muss nun der geeignetste Speicherort gefunden werden. Die Informationen sollen dabei in einem relationalen System gespeichert werden, damit bestehende Systeme wie auch zukünftige Anwendungen davon profitieren können. Hier steht ein SQL-Server 2012 Express von Microsoft zur Verfügung. Mit diesen Vorgaben sind folgende Möglichkeiten sinnvoll:

5.2.1 Variante 1: PSQL-Server

Bei dieser Variante wird nur das DBMS von Pervasive genutzt. Alle zusätzlichen Informationen werden im bestehenden PSQL-Server eingefügt. Dazu wird eine zusätzliche Datenbank mit den benötigten Tabellen angelegt. Damit befänden sich alle Daten im selben System.

Vorteile:
- Zugriff auf nur ein DBMS notwendig
- nur eine Datenschnittstelle notwendig
- Einarbeitung in nur ein System notwendig

Nachteile:
- Funktionsumfang ist gegenüber SQL-Server schlechter
- Integration in Visual Studio ist schlechter
- SQL-Server wird zum Persistieren verwendet (dann wieder zwei Systeme)
- Probleme mit nicht vorhanden Fremdschlüsseln, Indizes usw.
- Zugriff aus .NET erfolgt per ODBC und nicht direkt mit SQL

5.2.2 Variante 2: SQL- und PSQL-Server unabhängig

Hier werden beide zur Verfügung stehenden Systeme verwendet, allerdings ohne Koppelung. Die Anwendung greift auf beide Systeme zu und regelt den Datenfluss.

Vorteile:
- größte Flexibilität im Code
- Bestanddatenbank bleibt unverändert

Nachteile:
- viel höherer Programmieraufwand
- zusätzlicher Synchronisationsservice notwendig
- Zugriff nur durch eine Instanz sinnvoll (Synchron.-Probleme)
- keine einheitlichen Views in SQL möglich

5.2.3 Variante 3: Trigger in PSQL, Replikat im SQL-Server

Hier werden sogenannte Trigger in der bestehenden Datenbank hinzugefügt, die bei Änderung von Daten in Tabellen automatisch aufgerufen werden. Die Trigger-Funktionen duplizieren dabei die Änderungen in eine temporäre Datenbank. Durch einen zusätzlichen Dienst werden diese Änderungen mitverfolgt und damit ein Replikat der Pervasive-Datenbank im SQL-Server aktuell gehalten. Die zusätzlichen Daten für das IS werden ebenfalls im SQL-Server gespeichert.

Vorteile:
- durch Trigger nur lesenden Zugriff in den Fensterbaudaten
- Leistung nur bei Änderung notwendig (kein Polling)
- Alle relevanten Daten des IS sind im SQL-Server vorhanden.
- Zugriff aus der Anwendung auf nur ein System

Nachteile:
- leichter Eingriff in Produktionsdatenbank (Erstellen der Trigger)
- Redundante Datenhaltung (im PSQL und SQL-Server)
- Mehrere Systeme müssen aufeinander abgestimmt werden.
- Änderungen an der Datenbasis sind sehr aufwändig.

5.2.4 Variante 4: SQL-Server und Linked-Server auf PSQL

Im SQL-Server gibt es die Möglichkeit, fremde Systeme mittels sogenannten Linked-Servern zu verbinden. Dafür muss das Fremdsystem eine OLE-DB-Unterstützung besitzen. Im Fall von PSQL ist diese Unterstützung mittels der OLE DB zu ODBC-Treiber möglich. Im SQL-Server kann dann direkt per SQL auf diese Daten zugegriffen werden.

Vorteile:
- Zugriff auf nur ein DBMS notwendig
- Änderungen der Datenbasis sind einfach durchzuführen
- nur eine Datenschnittstelle notwendig
- Bestanddatenbank bleibt unverändert
- vollständig getrennte Systeme, größte Übersichtlichkeit der Lösung

Nachteile:
- eventuell Performance Probleme
- Kompatibilitätsprobleme der DBMS (32/64 Bit, Zeichendarstellung)
- mehrere Systeme müssen aufeinander abgestimmt werden

5.2.5 Entscheidung

Nach weiterer Recherche wird klar, dass die Variante 3 ausscheidet. Beim Erstellen von Triggern kann die Btrieve-Schnittstelle nicht mehr verwendet werden [pervasive3, S. 156]. Diese Schnittstelle wird von der Software Adulo allerdings benötigt, wie eine Rückfrage mit dem Hersteller ergeben hat.

Ein Test der verbleibenden Varianten zeigt enorme Probleme beim Zugriff auf die Daten aus .NET. Hier führt das Fehlen von Primärschlüsseln in den Tabellen der Bestanddatenbank zu inkorrekten Modellen der O/R-Mapper. Ein fehlerfreier Zugriff kann nur mittels Verwendung von direkten SQL-Statements per ADO.NET erreicht werden. Dies würde allerdings den Aufwand zur Erstellung einer Software mit Variante 1 und 2 deutlich erhöhen.

Ein Hauptkriterium für die zentrale Datenbasis des IS ist die Möglichkeit des Zugriffs aus verschiedenen Anwendungen und aus den bestehenden Auswertungen und Reports (vor allem Excel). In den Tests funktioniert die Variante laut Abschnitt 5.2.4 in diesem Bereich am besten. Sie ist übersichtlich und voll funktionell, erfüllt die notwendigen Kriterien und wird von den vorhandenen Microsoft-Technologien am besten unterstützt. Deshalb wird dieser Variante der Vorzug gegeben.

Der Zugriff auf die Daten kann somit laut Abbildung 9 durchgeführt werden. Als Zugriffsarten stehen dabei SQL, ODBC und SQL per Entity-Framework zur Verfügung.

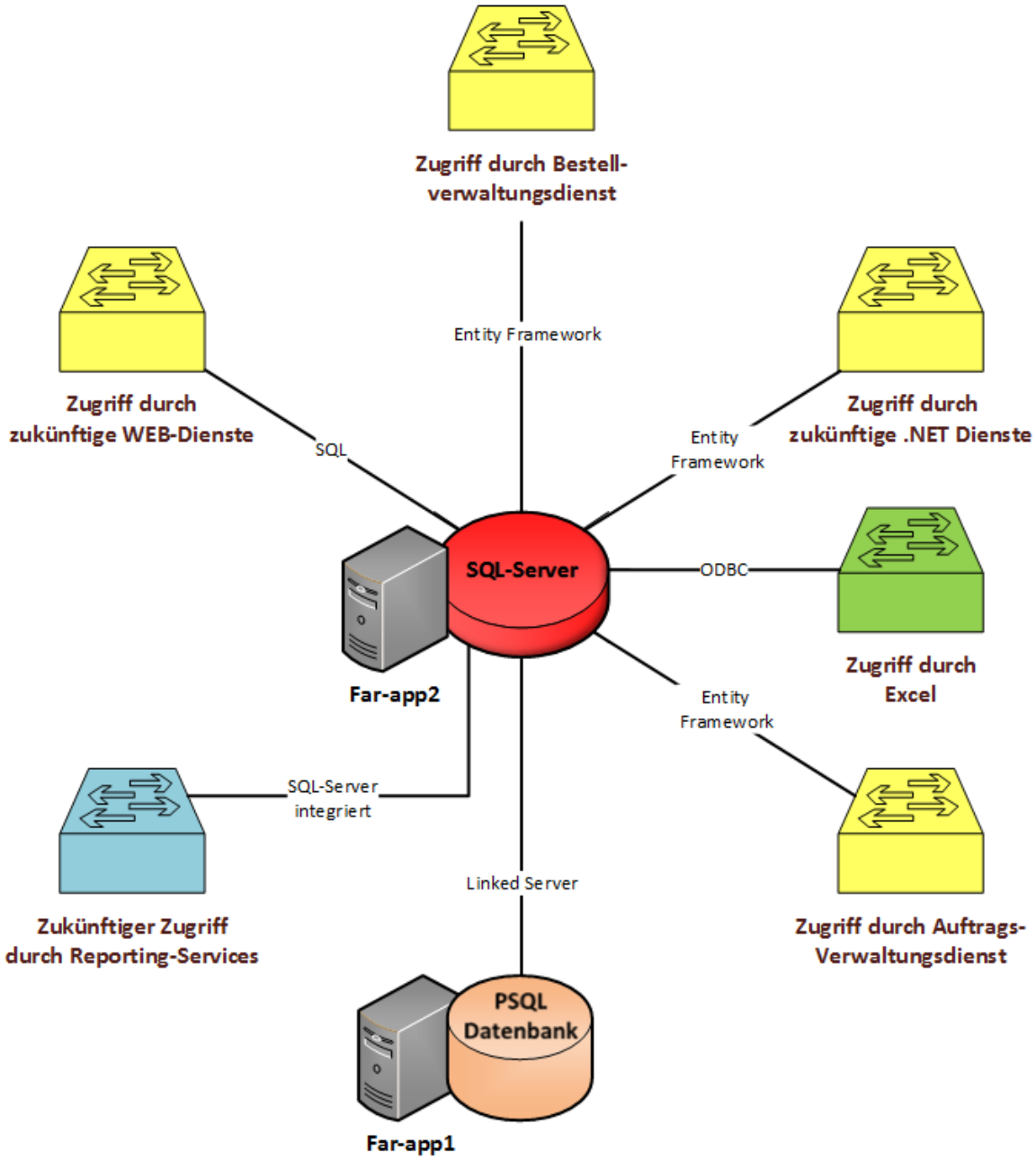


Abbildung 9: Mögliche Datenzugriffsarten

5.3 Modellierung der Daten für die Auftragsverwaltung

5.3.1 Daten im PSQL-Server

In diesem Abschnitt sollen die vorhandenen Daten der Software Adulo näher beleuchtet werden. Damit kann definiert werden, welche Felder aus dem Bestand verwendet werden und welche Attribute neu implementiert werden müssen.

Bei der vorhandenen Datenbank im Pervasive-DBMS zeigt die Analyse, dass pro Tabelle jeweils eine eigene DAT-Datei existiert. Durch Untersuchen der Ordnerstrukturen mit diesen DAT-Dateien wird die Unterteilung in drei Hauptbereiche klar:

- *Daten* enthält alle Belegdaten und die Nummernkreise
- *Stammdat* enthält alle Stammdaten
- *Kunden* enthält alle Kundendaten

Um Zugriff auf die einzelnen Tabellen zu bekommen, werden sogenannte Dictionary-Files benötigt, welche die Verbindung zwischen transaktioneller Datendatei und dem relationalen Zugriffssystem herstellen. Diese Dictionary-Files werden von Adulo auf Anfrage beim Support bereitgestellt. Nach Installation der DDF-Dateien ist der relationale Zugriff per SQL auf die Tabellen mittels PCC (Pervasive Control Center) möglich.

Um einen genauen Überblick aller verwendeten Tabellen und deren Attribute zu erhalten, wird bei allen Tabellen das Schema als SQL-Skript exportiert und danach analysiert. Die Auftragsdaten, die für die Auftragsverwaltung interessant sind, liegen in der Tabelle ‚Auf-Dir'. Die Daten der Kunden sind in der Tabelle ‚KundenDaten' im Kundenbereich, ebenfalls in einer eigenen Datei gespeichert. Die Beziehung der beiden Tabellen wird durch die Eigenschaft ‚IntKundenNr' hergestellt, allerdings ohne referenzieller Integrität und ohne Festlegung von Primär- und Fremdschlüssel in der Datenbank. Die Zuordnung erfolgt nur durch die Anwendung.

5.3.2 Daten im SQL-Server

Damit die notwendigen Informationen für die Abwicklung eines Auftrags im Bereich Kunststofffenster festgelegt werden können, müssen zuerst die aktuell verwendeten Daten ausgewertet werden.

Die Analyse der bestehenden Auftragsabwicklung gestaltet sich dabei etwas schwieriger als gedacht, weil hier keine Dokumentation oder Prozessbeschreibung vorliegt. Es werden deshalb mit den zuständigen Bearbeitern verschiedene Auftragsfälle simuliert und die dafür benötigten Daten und Listen aufgezeichnet. Die verwendeten Excel-Sheets werden danach einzeln auf die darin gespeicherten Informationen hin analysiert.

Die Auswertung des Auftragsprozesses eines Fensterauftrages zeigt die Verwendung folgender Informationen auf:

- KürzelAV
- KürzelVK
- Montageart
- Zusatz
- Bereich
- ProfillkennNr
- FarbZusatz
- TypZusatz
- AnzahlHT
- AnzahlNT
- AnzahlSB
- AnzahlPSK
- AnzahlFalter
- AnzahlHST
- StkVergabe
- EHSonderbau
- EHStandard

Zur Durchführung der gestellten Aufgabe werden noch folgende Eigenschaften eingeführt:

- ErstellDatum
- Endtermin
- Montagebeginn
- Produktionsbeginn
- Auftragsart
- InMontageÜbernommen
- InProduktionÜbernommen
- Auftragsstatus

Nach erfolgter Sortierung, Zusammenfassung und Zuordnung der einzelnen Entitäten wird das Modell laut Abbildung 10 erstellt. Die Tabelle der Auftragsdaten stellt dabei den zentralen Zugriffspunkt der hinzugekommenen Informationen dar.

Die Erweiterung der Daten für zukünftige Zusatzapplikationen, wie zum Bsp. eine Bestellverwaltung, kann bei Bedarf durchgeführt werden. Das Modell ist hierzu flexibel erweiterbar. Zusätzliche Daten aus der Fensterbau-Software müssen mit jeweils einem eigenen Linked-Server verbunden werden. Die Erweiterungen im SQL-Server können direkt vorgenommen werden.

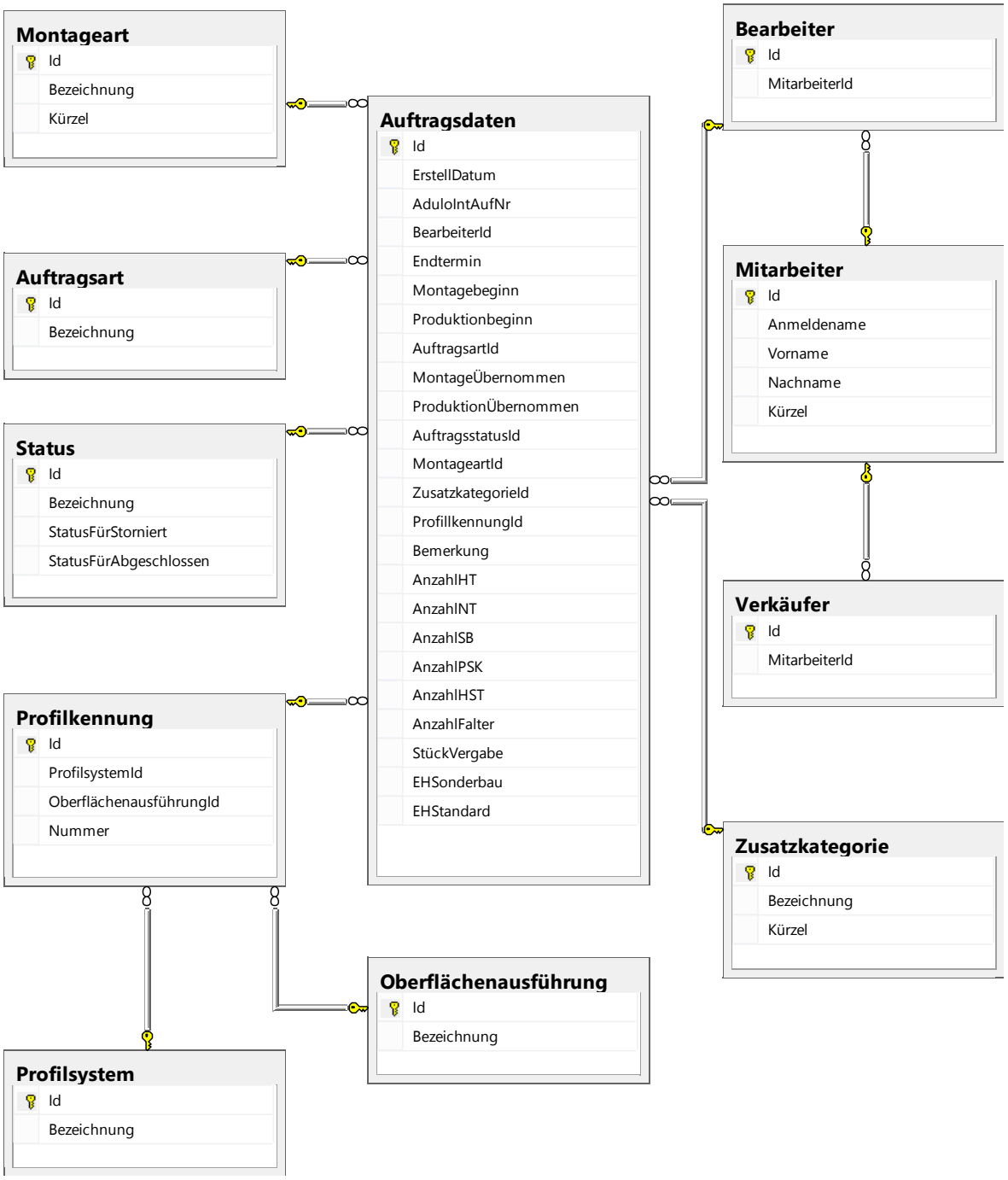


Abbildung 10: Zuordnung der Erweiterungs-Entitäten

6 Modellierung der Software

Nachdem im vorherigen Kapitel die Datengrundlage für das Informationssystem definiert wurde, sollen in diesem Kapitel die Software-Grundlagen festgelegt werden. Dazu werden die Architektur der Software, das Softwaredesign und die verwendeten Technologien bestimmt. Diese Definitionen werden als Konzept für das gesamte IS getroffen. Die Feinentwürfe und die Umsetzung berücksichtigen gezielt den Bereich der Auftragsverwaltung.

6.1 Architekturentwurf

Beim Architekturentwurf soll die grundlegende Struktur des Informationssystems festgelegt werden. Da es sich bei der Findung der richtigen Architektur meist um einen iterativen Prozess handelt [geirhos 2012, S. 52], werden zuerst die Ziele und die dafür notwendigen Komponenten ausgearbeitet. Danach können passende Architekturmodelle verglichen und ausgewählt werden. Im Laufe der Entwicklung des Systems kann dieser erste Entwurf allerdings noch geändert werden.

6.1.1 Ziele der Architektur

Die Ziele der Architektur leiten sich direkt aus den Zielen des Informationssystems aus Abschnitt 4.1 ab. Besonderen Einfluss auf Architektur des Informationssystems besitzen dabei die geforderte Durchgängigkeit der Anwendung sowie die einfache und stufenweise Erweiterbarkeit.

Weitere Kriterien, die zu berücksichtigen sind, sind der Mehrbenutzerzugriff, die Verteilung der Anwendung auf alle Abteilungen und die Schnittstellen zu den Bestandssystemen.

6.1.2 Festlegung der Komponenten

Zu Beginn der Architekturfestlegung müssen alle notwendigen Komponenten gefunden werden. Sind diese benannt, kann mit der Zuordnung der Komponenten zueinander ein Modell für eine Software-Architektur erstellt werden.

Beim grundlegenden Konzept dieses IS werden auch die Bereiche Bestellverwaltung und Angebotswesen miteinbezogen. Bei der Implementierung werden nur mehr die Komponenten der Auftragsverwaltung berücksichtigt.

Aus den bisher gestellten Anforderungen ergeben sich folgende Komponenten:

- Schnittstelle zu Adulo
- Auftragsverwaltung
- Aufträge
- Bestellverwaltung
- Bestellungen
- Schnittstelle zu Montage
- Schnittstelle zu Produktion
- Client-Modul für Produktion
- Client-Modul für Montage
- Client-Modul für Techniker/Verkäufer
- Client-Modul für Lager
- Client-Modul für Einkäufer
- Benutzerverwaltung
- Rechteverwaltung
- Schnittstelle zu Excel-Auswertungen

Folgende Komponenten existieren bereits:

- Software Adulo
- Datenbank Pervasive
- Datenbank SQL-Server
- Applikation Server
- Active Directory
- Excel-Auswertungen

6.1.3 Festlegung der Architektur

Im ersten Schritt werden die Komponenten nach Zusammengehörigkeit geordnet, danach erfolgt eine erste Strukturierung. Diese zeigt, dass getrennte Applikationen sinnvoll sind, die (teilweise) dieselben Datenzugriffsstrukturen sowie Autorisierungsfunktionen verwenden.

Mit der Festlegung der Datenhaltung im SQL-Server stehen für die Applikationen somit folgende Möglichkeiten des Zugriffs zur Verfügung:

6.1.3.1 Mehrere verteilte Anwendungen mit Direktzugriff auf das DBMS

Bei dieser Variante werden entweder eine oder mehrere vollständige Anwendungen bei jedem Client installiert. Diese greifen dann auf das zentrale DBMS zu (siehe Abbildung 11). In der jeweiligen Applikation müssen die Regeln zum gemeinsamen Datenzugriff integriert werden, meist wird dies durch bewusstes Sperren von Datensätzen erreicht. Die vorhandene Fensterbau-Software Adulo funktioniert nach diesem Prinzip.

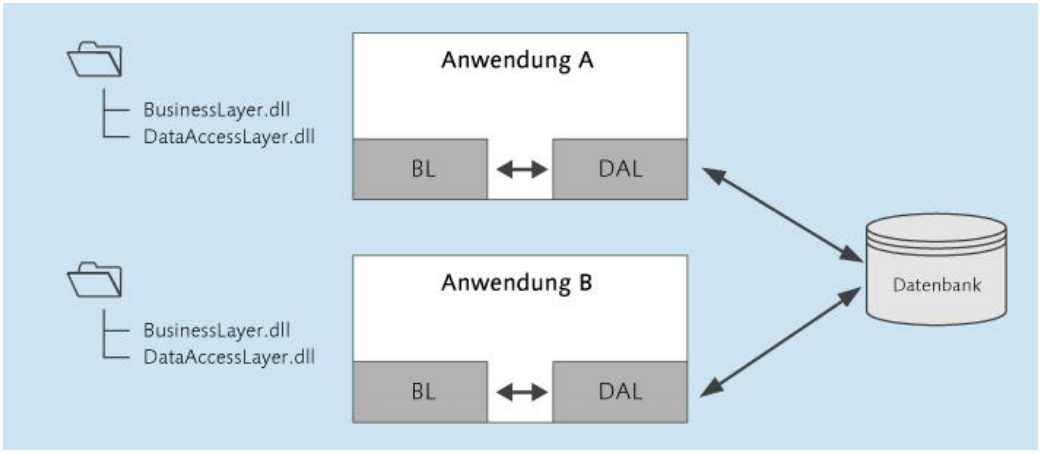


Abbildung 11: Verteilte Anwendungen [geirhos 2013]

6.1.3.2 Eine klassische Client-Server Lösung

Bei dieser Variante läuft am Server eine (oder mehrere) Anwendungen, welche die Anforderungen der Clients entgegennimmt, verarbeitet und beantwortet. Der Client dient hauptsächlich nur als Präsentationsschicht. Die Datenzugriffsschicht und die Geschäftslogik sind in der Serversoftware ebenso integriert wie die Regeln für den gemeinsamen Zugriff.

6.1.3.3 Eine serviceorientierte Architektur

Bei einer serviceorientierten Architektur erfolgt die Aufteilung der Funktionen des Gesamtsystems auf einzelne Dienste. Ein Dienst ist dabei ein abgeschlossenes Stück Software, das definierte Dienstleistungen zur Verfügung stellt (siehe Abbildung 12). Er besteht dabei meist aus einer logisch zusammengehörigen Funktionssammlung. Die Funktionen und Methoden der Dienste können von Client-Applikationen, aber auch von anderen Diensten genutzt werden. Übergeordnete Dienste können ebenso zusätzliche Funktionen für Anwendungen bereitstellen, indem sie vorhandene Dienste koordinieren (Orchestration Services).

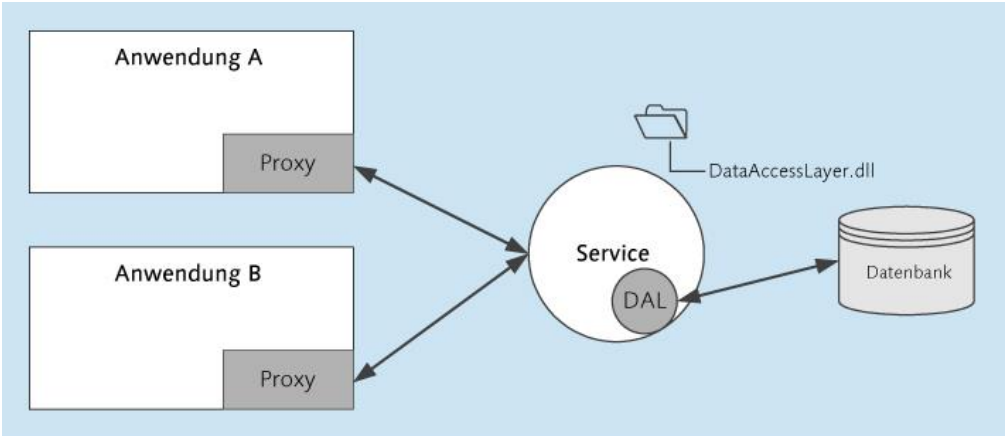


Abbildung 12: Serviceorientierte Architektur [geirhos 2013]

Die Dienstleistung besteht aus einem Serviceaufruf und Daten, die übergeben und zurückgegeben werden. Sie wird durch einen Vertrag (Contract) definiert. Dieser wird mithilfe der Web Services Description Language beschrieben. Die Kommunikation geschieht standardisiert mittels Nachrichten [geirhos 2013, S. 116].

Durch die lose Koppelung der Dienste untereinander und zu den Anwendungen können die Entwicklung sowie Änderungen oder Erweiterungen einzelner Dienste viel einfacher erfolgen. Eine SOA (Serviceorientierte Architektur) kann ständig um weitere Services ergänzt werden und so stetig wachsen. Sie ist dabei nicht auf homogene Systeme beschränkt. Sie eignet sich daher besonders zur Verwendung für die Unternehmens-IT, also der Abbildung von Geschäftsprozessen in einem Unternehmen [geirhos 2013, S. 119].

6.1.3.4 Vergleich und Entscheidung

Die funktionalen Anforderungen an das IS können von allen 3 Architekturen beinahe gleichermaßen erfüllt werden, einzig der gemeinsame Zugriff unterscheidet sich. Deshalb erfolgt der Vergleich hauptsächlich anhand der nichtfunktionalen Anforderungen.

Kriterien	Gewichtung	Verteilte Anwendungen	Client-Server Architektur	Service-orientierte Architektur
Erweiterbarkeit	18	6	12	18
Gleichzeitiger Zugriff	9	3	9	9
Wiederverwendbarkeit	15	5	10	15
Umsetzungsaufwand	9	9	6	3
Skalierbarkeit	4	2	4	4
Performance	12	12	12	4
Verwaltbarkeit	6	4	6	6
Integration	6	6	4	6
Vorhandenes Know-How	6	4	2	2
Komplexität	9	9	6	6
Sicherheit	6	2	4	6
		62	**75**	**79**

Tabelle 5: Vergleich der Architekturen

Der Vergleich anhand der in Tabelle 5 aufgeführten Kriterien zeigt, dass sich die serviceorientierte Architektur für diese Aufgabe am besten eignet. Vor allem die geforderte Erweiterbarkeit und die gute Wiederverwendbarkeit der Komponenten sind hier entscheidend.

6.1.4 Zusammenfassung der Komponenten

Nach Festlegung der Architektur können die Komponenten zu einzelnen Diensten bzw. Funktionsgruppen zusammengefasst werden. Durch logische Gruppierungen der Komponenten ergibt sich die Architektur laut Abbildung 13.

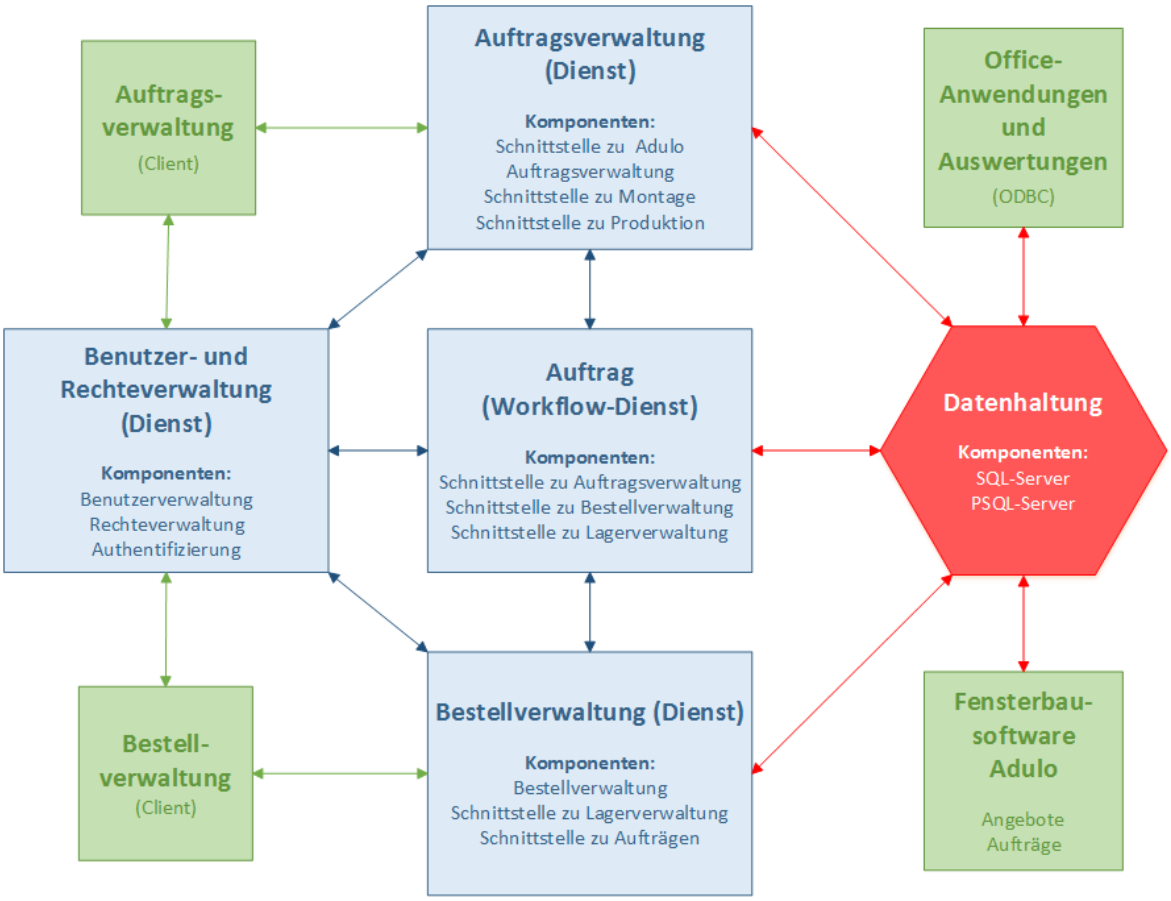


Abbildung 13: Architekturübersicht

Aufgrund der vorgegebenen Erweiterungsfähigkeit und Wiederverwendbarkeit der Komponenten wird das IS durch Einführung der drei Layer „Daten“, „Business“ und Presentation“ strukturiert.

Durch Konzeption des IS als eine serviceorientierte Anwendung wird die Präsentationslogik nur in der Clientanwendung abgebildet. Die Geschäftslogik soll durch die entsprechenden Dienste verwirklicht werden. In diesen erfolgt der Zugriff auf den Daten-Layer, der aus der Datenbank und der dazugehörigen Zugriffsschicht besteht.

6.2 Softwaredesign

Das Softwaredesign verfeinert den Architekturentwurf mit konkreten Angaben über die zur Umsetzung verwendeten Technologien. Am Ende diese Abschnitts sollten alle verwendeten Technologien definiert und somit der genaue Rahmen zur Umsetzung vorgegeben sein.

6.2.1 Datenzugriffsverfahren

In Kapitel 5 wurde als zentrales DBMS der vorhandene SQL-Server 2012 Express ausgewählt. Jetzt sollen verschiedene Arten des Zugriffs verglichen und anschließend die beste Variante ausgewählt werden.

Mit der Entscheidung für den SQL-Server sind folgende Möglichkeiten für den Zugriff gegeben:

- ADO.NET
- LINQ to SQL
- ADO.Net Entity-Framework

Bei ADO.NET handelt es sich um mehrere Klassen, die eine verbindungslose Zugriffsmethode auf die Datenbank realisieren. Der Zugriff erfolgt mittels Connection-Objekten, welche die eigentliche Verbindung zur Datenbank erstellen. Gearbeitet wird entweder mit DataReadern (für schnellen, rein lesenden Zugriff) oder mit DataSets. Die DataSets stellen eine (teilweise) Kopie der Daten im Arbeitsspeicher der Anwendung dar. Das Zusammenwirken der Einzelkomponenten ist in Abbildung 14 ersichtlich. Das ADO.Net-Framework kümmert sich um die eigentlichen Datenabfragen und Datenspeicherungen. Diese werden mittels SQL realisiert.

Bei LINQ (Language Integrated Query) handelt es sich um eine einheitliche Abfragesprache für unterschiedlichste Datenquellen [kansy 2013, S.375]. LINQ to SQL erweitert diese ähnlich dem ADO.NET Entity-Framework um einen O/R Mapper. Dieser stellt die Verbindung zwischen den relationalen Daten und den objektorientierten Klassen dar. Der Zugriff auf diese erfolgt mit einer dem Standard SQL sehr ähnlichen Abfragesprache im Programmcode.

Das ADO.NET Entity-Framework wurde bereits in Abschnitt 3.2 näher erläutert.

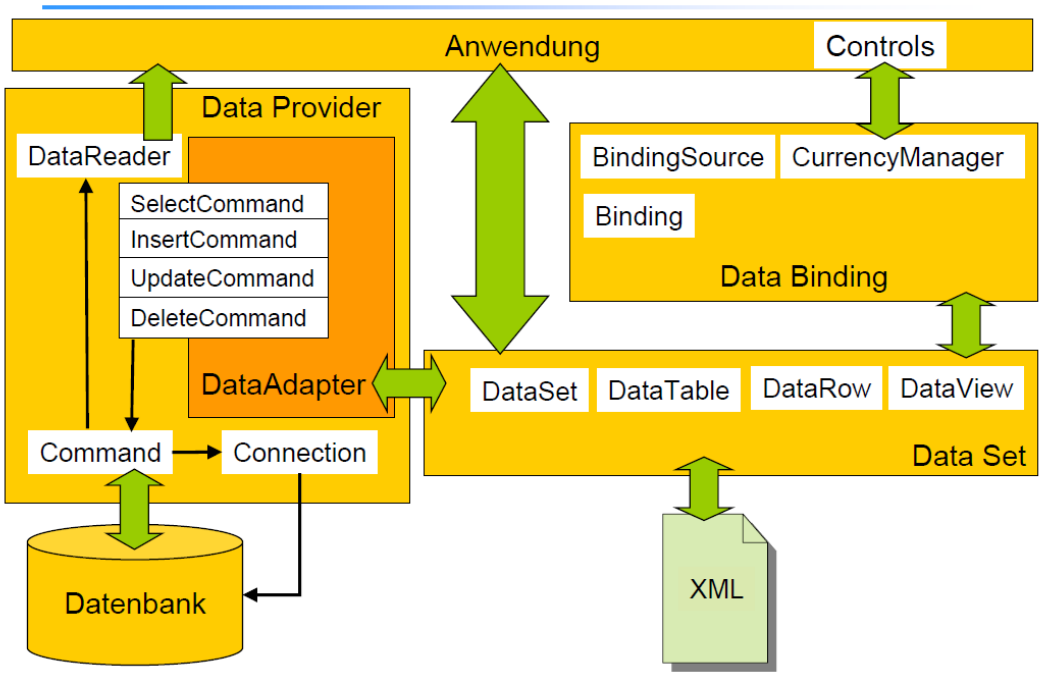


Abbildung 14: Architektur ADO.Net

(Quelle: Vorlesung Objektorientierte Programmierung, Prof. Delport)

Um alle 3 Varianten besser vergleichen zu können, werden die wichtigsten Kriterien in Tabelle 6 aufgeführt. Die Kriterien „Programmieraufwand“ und „Performance“ wurden für Anwendungen kleiner bis mittlerer Größe bewertet. Beim Vergleich der Kriterien wird deutlich, dass sich ADO.NET aufgrund des direkten Zugriffs auf SQL für große, performancekritische Anwendungen am besten eignet. Im direkten Vergleich von LINQ to SQL und EF fällt die Entscheidung vor allem aufgrund der Kriterien „Änderungs- und Erweiterbarkeitsaufwand“, sowie „zukünftige Unterstützung durch Microsoft“ zu Gunsten des Entity-Frameworks aus.

Kriterium:	**ADO.NET**	**LINQ to SQL**	**Entity-Framework**
Performance	Hoch	Mittel	Mittel
Programmieraufwand	Hoch	Niedrig	Mittel
Änderungsaufwand	Hoch	Mittel	Niedrig
Flexibilität	Hoch	Niedrig	Mittel
Erweiterbarkeitsaufwand	Hoch	Mittel	Niedrig
Typsicherheit	Nicht gegeben	Gegeben	Gegeben
Wartungsaufwand	Mittel	Niedrig	Niedrig
Skalierbarkeit	Hoch	Schlecht	Mittel
Lösung ausgereift	Hoch	Mittel	Mittel
Zukünftige Unterstützung d. Microsoft	Mittel	Schlecht	Hoch

Tabelle 6: Vergleich Datenzugriffsverfahren nach Kriterien

6.2.2 Windows Communication Foundation

Die WCF (Windows Communication Foundation) wird zur Kommunikation mit den Diensten verwendet. Sie ist eine dienstorientierte Plattform, die in einer einheitlichen API dem Programmierer verschiedene Techniken zur Verfügung stellt. Sie stellt ein umfangreiches und mächtiges Framework bereit, um nahezu alle Kommunikationsanforderungen abzudecken, sodass sie einerseits die Vorteile anderer Verfahren in sich vereint und sie andererseits dadurch ersetzt [geirhos 2013, S. 433].

6.2.3 Hosting

Die richtige Wahl der Hosting-Umgebung ist wichtig, hat diese doch maßgeblichen Einfluss auf Performance, Funktionalität, Stabilität und Deployment der Anwendung [geirhos 2013, S. 472]. Prinzipiell kommen für das fertige IS folgende Hosting-Möglichkeiten in Frage:

6.2.3.1 Selfhosting

Die Vorteile beim Selfhosting sind die volle Kontrolle über die Services (vor allem über Start und Beendigung), die Ausgabe von Statusinformationen auf der Konsole und die schnelle Implementierung. Die Nachteile sind das notwendige manuelle Starten des Hosts (auch am Server – hierzu muss ein Benutzer angemeldet sein), außerdem muss sich die Host-Anwendung um alle Belange selber kümmern (Bsp. Fehlerbehandlung auf Serviceebene oder Monitoring) [vergl. geirhos 2013, S. 475].

Ein spezieller Fall von Selfhosting ist die Verwendung der Managed Windows Services (das sind die NT-Services mit Managed-Code). Hier kann ein Start automatisch (ohne Anmeldung am Server) erfolgen, die Dienste können einfach beendet und wieder gestartet werden und es können Aktionen, die im Fehlerfall durchgeführt werden sollen, angegeben werden. Allerdings muss man sich auch hier um viele Details selber kümmern und die Einrichtung ist relativ umständlich, was für sich ständig ändernde Dienste nachteilig ist.

6.2.3.2 Hosting am IIS (Internet Information Server)

Hier sind die Vorteile das einfache Deployment (im Idealfall reicht ein einfaches Kopieren zum Austausch während der Laufzeit) und die eingebauten Funktionen des Internet-Information-Servers (Bsp. Recycling und Health Monitoring). Als Nachteil ist die Zuständigkeit des IIS für die Lebensdauer der Dienste zu nennen (beim Recyceln der Appdomain werden die Services und damit auch die Sessions beendet). Vor der Version 7.0 ist außerdem nur die Verwendung von http (Hyper Text Transfer Protokoll) als Protokoll möglich [vergl. geirhos 2013, S. 483]. Ab Version 7.0 kann man mit der Nachinstallation der WPAS (Windows Process Activation Services) die Protokolle MSMQ (Microsoft Message Queuing), net.tcp und Named-Pipes verwenden [geirhos2013, S. 491].

6.2.3.3 AppFabric

AppFabric ist eine Ergänzung zum IIS, die einen Dashboard, sowie weitere Konfigurationsmöglichkeiten per Power Shell oder mittels einer GUI-Erweiterung (Graphical User Interface) bietet. Mit AppFabric können Dienste am IIS übersichtlicher verwaltet werden, das Deployment wird weiter vereinfacht, bessere Tracking-Funktionen sind ebenso vorhanden, wie Monitoring von WCF- und Workflow-Services.

Für das IS kommen nur Selfhosting mit Windows NT-Services, Hosting am IIS mit WAS und Hosting mit AppFabric in Frage. Der Vergleich nach den Kriterien in Tabelle 7 zeigt jedoch deutlich, dass Self-Hosting für diese Aufgabenstellung nicht geeignet ist. Im Vergleich zwischen IIS und AppFabric zeigt Letzterer deutlich mehr Funktionen, allerdings auch mehr Einrichtungsaufwand. Aufgrund der zusätzlichen Funktionen, vor allem die WF-Persistierung und das Health-Monitoring von WF und WCF, fällt die Entscheidung zu Gunsten von AppFabric als Host aus.

Kriterien	Self Hosting	IIS	AppFabric
Health Monitoring	nicht unterstützt	teilw. unterstützt	gut unterstützt
Aktivitäts- und System Monitoring	schlecht unterstützt	unterstützt	gut unterstützt
Aktivierung	nicht unterstützt	gut unterstützt	gut unterstützt
Instance Management	nicht unterstützt	unterstützt	gut unterstützt
Isolation	nicht unterstützt	unterstützt	unterstützt
Skalierbarkeit	nicht unterstützt	unterstützt	unterstützt
Sicherheit	unterstützt	unterstützt	unterstützt
Konfigurierbarkeit	schlecht unterstützt	unterstützt	gut unterstützt
Tool-Unterstützung	niedrig	mittel	hoch
Komplexität	niedrig	niedrig	mittel
Entwicklungsaufwand	hoch	mittel	mittel
Deploymentaufwand	hoch	niedrig	niedrig
Einrichtungsaufwand	niedrig	mittel	hoch
Unterstützung WF	nicht direkt	nicht direkt	gut unterstützt
Unterstützung WCF	nicht direkt	nicht direkt	gut unterstützt

Tabelle 7: Vergleich der Hosting-Möglichkeiten

6.2.4 Objektorientierte Analyse und -design

Die Modellierung der Klassen wird für den Bereich der Auftragsabwicklung behandelt. Sie wird hier hauptsächlich durch den O/R-Wrapper des Entity-Frameworks durchgeführt.

Dieser erzeugt aus allen Entitäten (und auch Sichten) eigene Klassen mit den dazugehörigen Eigenschaften und Methoden. Dazu gehören auch die entsprechenden Navigationseigenschaften, die aus den Beziehungen abgeleitet werden.

Als Hauptklasse dient in diesem Fall die Klasse ‚Auftragsdaten'. Sie stellt die Struktur eines Auftrags in der Auftragsverwaltung dar. Als Austauschobjekt zwischen Dienst und Client sowie zwischen zukünftigen Diensten dient eine vereinfachte Ausführung ohne Navigationseigenschaften zu anderen Klasse. Diese Klasse wird aus einer entsprechenden Sicht im SQL-Server abgeleitet und ‚Auftraglistung' genannt.

6.2.5 Client-Anwendungen

Die Client-Anwendungen werden als WPF-Anwendungen (Windows Presentation Foundation) ausgeführt. Als Alternative gäbe es noch Win-Forms oder auch Browser-Anwendungen wie zum Beispiel ASP.Net oder Silverlight. Die Wahl für WPF fällt aufgrund einer bereits vorhandenen Anwendung, die als Pilot für die optische Ausführung solcher Art von Clients diente. Außerdem wird WPF von Microsoft als zukünftige Technik zur Oberflächengestaltung empfohlen [msdnwpf1]. Ein weiterer Vorteil ist das vorhandene Know-How.

Die Windows Presentation Foundation ist ein Grafik-Framework, das Teil des .NET Frameworks seit Version 3.0 ist. Dabei werden die Präsentations- und die Geschäftslogik getrennt. Die Beschreibung wird mit der Auszeichnungssprache XAML (basierend auf XML) durchgeführt [kühnel 2012, S.759].

6.2.6 Authentifizierung und Autorisierung

Die Autorisierung soll laut Vorgabe im Abschnitt 4.3.4 rollenbasiert erfolgen, das heißt jeder Benutzer hat ihm zugeordnete Rollen. Diese Rollen steuern die möglichen Funktionen und Berechtigungen im Informationssystem.

Die WCF stellt folgende rollenbasierte Autorisierungsmöglichkeiten zur Verfügung [meier 2008, S. 101]:

- Windows-Gruppen (für eine geringe Anzahl an Rollen in einer Domäne)
- ASP.NET Rollen (für unterschiedliche Rollen-Provider)
- benutzerdefinierte Rollen (für spezielle Bedürfnisse)

Da es sich beim Informationssystem um eine Anwendung handelt, die komplett innerhalb der Windows-Domäne ausgeführt wird und nur eine geringe Anzahl an unterschiedlichen Rollen zu erwarten ist, ist die Autorisierung mit Windows-Gruppen ausreichend.

7 Implementierung

Das Konzept des IS wurde in den vorherigen Kapiteln festgelegt. Damit wurden auch die notwendigen Entscheidungen für die Teilumsetzung im Bereich der Auftragsverwaltung getroffen und es kann nun endlich die Programmierung beginnen.

Dazu werden zuerst die Entwicklungsumgebung, die Testdaten und das Testsystem eingerichtet, danach folgt die eigentliche Implementierung.

7.1 Vorbereitungen

Das Ziel dieses Abschnitts ist das Schaffen von notwendigen Voraussetzungen für die Entwicklung und das Testen des Informationssystems.

7.1.1 Entwicklungsumgebung einrichten

Als Entwicklungscomputer wird ein virtueller Windows 8-PC in einer Vmware-View-Infrastruktur eingerichtet. Mit dieser Lösung ist der Zugriff auf den PC auch außerhalb von Farkalux möglich. Auf diesem PC werden neben den notwendigen Office Programmen auch das Fensterbauprogramm Adulo Version 11.9h sowie ein Pervasive Client Version 11.31 und das Pervasive Control Center in derselben Version installiert.

Als Entwicklungsumgebung wird Visual Studio 2012 Professionell installiert und direkt im Anschluss auf Update 3 aktualisiert. Als Quellcodeverwaltung dienen die Team Foundation Services, in welchen ein neues Projekt eingerichtet wird.

Die Paketverwaltung wird mit der aktuellsten Version von NuGet realisiert. Die Erweiterung von Visual Studio wird mit dem Programm Resharper in der Version 7.1.3 durchgeführt.

Zur Verwaltung des SQL-Servers wird das SQL-Management-Studio 2012 installiert. Hier muss vor der Installation das Windows-Gebietsschema von Deutsch (Österreich) auf Deutsch (Deutschland) geändert werden, ansonsten bricht das Setup mit einer Fehlermeldung ab.

Nach Installation der SQL-Server-Management-Tools im Modus „vollständig“ steht auch ein SQL-Profiler zur Verfügung. Mit dem Management-Studio wird erfolgreich eine Testverbindung zur Express-Instanz des SQL-Servers am Applikations-Server ‚far-app2‘ eingerichtet. Im Anschluss daran wird die CLR Unterstützung auf diesem Server eingeschaltet. Damit ist die Ausführung von .NET-Code am SQL-Server möglich.

7.1.2 Upgrade Pervasive PSQL

Die Verbindung der Datenbanken soll mittels eines Linked-Servers im SQL-Server realisiert werden. Um eine Verbindung zu einem Fremd-Server herstellen zu können, muss dazu entweder ein eigener Datenanbieter existieren oder es wird ein bestehender OLE DB Anbieter verwendet.

In diesem Fall eignet sich der Anbieter „Microsoft OLE DB Provider for ODBC Drivers“, weil Pervasive einen solchen ODBC-Treiber bereitstellt. Die benötigte 64-Bit-Variante dieses Treibers ist laut [pervasive3, S. 2] allerdings erst ab Version 11 des PSQL-Servers und hier ab dem Update SP3 enthalten. Das heißt hier muss eine Aktualisierung erfolgen.

Die Hard- und Softwareanforderungen für Version 11 SP3 werden vom Zielsystem erfüllt. Die Version wird ebenfalls von der Fensterbau-Software unterstützt. Eine Überprüfung der Client-Kompatibilität zeigt, dass die vorhandenen Clients in der Version 10 mit einem Server in Version 11 voll kompatibel sind. Diese Voraussetzung ist wichtig, weil in der Buchhaltung die Workgroup-Engine von Pervasive in der Version 10.2 verwendet wird, die ebenfalls den Client beinhaltet. Eine Aktualisierung auf eine neuere Version wird von der Finanzbuchhaltungssoftware allerdings nicht unterstützt.

Da alle notwendigen Voraussetzungen gegeben sind, wird eine Aktualisierung durchgeführt. Dazu wird eine Upgrade-Lizenz für 20 Benutzer geordert. Nach einer vollständigen Sicherung des Applikations-Servers wird die vorhandene PSQL-Version deinstalliert und direkt im Anschluss die neue Version 11.30.030.000 installiert. Ein direktes Update ist aufgrund des Architekturwechsels von 32 auf 64 Bit nicht möglich. Nach Einspielen der aktuellsten Updates (Update 4 Release Mai 2013) wird der Server neu gestartet.

Im Test funktioniert Adulo lokal am Server und auch auf den bestehenden Clients ohne Probleme. Aus Performance-Gründen werden alle Clients (außer der erwähnte Buchhaltung-PC) ebenfalls auf obige Serverversion aktualisiert. Ein weiterer Test zeigte keine Probleme, allerdings auch keine Performance-Verbesserungen.

Nach einer Woche Produktionsbetrieb werden die Lizenzierungsinformationen aktualisiert, das heißt die neue Version wird permanent freigeschaltet.

7.1.3 Entwicklungs-Datenbank erstellen

Die Datenbank, die zur Entwicklung verwendet wird, sollte der Produktionsdatenbank sehr ähnlich sein. Dies ermöglicht realistischere Performance-Einschätzungen während der Erstellung der Software. Aus diesem Grund wird die momentan verwendete Produktionsdatenbank mittels einer Offline-Kopie der Datenbankdateien geklont. Diese Kopie der Produktionsdatenbank wird in einer eigenen Sektion am Produktivserver eingerichtet. Somit werden derselbe Server sowie dieselben Daten zur Software-Entwicklung verwendet und gleichzeitig eine Trennung zum Produktionssystem aufrechterhalten.

Die Datenbanken werden mittels PCC laut den 3 Datenbereichen der Fensterbau-Software als ‚DevTestAduloStammdat', ‚DevTestAduloDaten' und ‚DevTestAduloKunden' eingerichtet.

Als direkte Verbindung für das Softwaresystem fungiert die Datenbank ‚DevTestAdulo', die in der Express-Instanz des SQL-Servers am Applikationsserver ‚far-app2' eingerichtet wird. In dieser Instanz werden auch die Verbindungsserver-Objekte zur Pervasive-Datenbank erzeugt. Dazu muss der Client von Pervasive vollständig installiert werden. Während der Installation wird dabei der notwendige ODBC-Treiber in der 64-Bit-Variante hinzugefügt.

Jetzt kann der Verbindungsserver eingerichtet werden. Dies geschieht unter Verwendung des Connection Strings „Driver={Pervasive ODBC Interface}; ServerName=far-app1; DBQ=DEVTESTADULODATEN; TransportHint=TCP; PvTranslate=auto" laut [pervasive5, S. 454].

Eine Testabfrage mittels SELECT liefert keine Daten, sondern die Fehlermeldung, dass der angeforderte Vorgang nicht durchgeführt werden kann. Eine Recherche in dem Support-Forum von Pervasive zeigt die DEP (Data Execution Prevention) als mögliches Problem in Verbindung mit diesem Fehler auf. Nach Deaktivierung und anschließendem Neustart des Servers funktioniert obige SELECT-Abfrage wie erwartet. Die DEP verbleibt nun im Status deaktiviert, weil diese Einstellung nicht in den Sicherheitsrichtlinien von Farkalux gefordert wird.

Die beiden Tabellen der Pervasive-Datenbank werden mit jeweils eigenen Linked-Servern eingebunden (‚DEVTESTLINKADULODATEN', ‚DEVTESTLINKADULOKUNDEN'). Eine SELECT-Abfrage wird allerdings mit folgender Fehlermeldung quittiert:

```
Der OLE DB-Anbieter 'MSDASQL' für den Verbindungsserver 'DEVTESTLINKADULODATEN' hat die Meldung '[Pervasive][ODBC Client Interface]Invalid date, time or timestamp value.' zurückgeben.
Meldung 7330, Ebene 16, Status 2, Zeile 3
Eine Zeile kann nicht vom OLE DB-Anbieter 'MSDASQL' für den Verbindungsserver 'DEVTESTLINKADULODATEN' abgerufen werden.
```

Die Abfrage von einzelnen Attributen funktioniert jedoch. Eine Recherche im Supportforum des Herstellers zeigt ein mögliches Problem mit dem Format von Datums-Attributwerten in Verbindung mit ODBC zu OLE-DB. Ein Test mit der gezielten Abfrage des Attributs ‚AufMassTermin' erzeugt den Fehler. Von diesem Formatfehler sind folgende Attribute betroffen:

[AufMassTermin], [FertTerminGeplant], [WunschTermin], [MontageTermin], [Abnahme-Termin], [KundenDienstTermin]

Da diese Felder bei Farkalux weder durch das Programm Adulo noch durch das IS genutzt werden, hilft die Erstellung einer View pro Tabelle, die alle Spalten außer den obigen

beinhaltet. Der Vorteil dieser Variante ist außerdem die IntelliSense-Unterstützung der View-Bezeichner im SQL-Server.

7.1.4 Testsystem einrichten

Zum Testen des Informationssystems bzw. des gesamten Prozesses der Auftragsabwicklung wird eine funktionsfähige Installation der Fensterbau-Software Adulo benötigt. Diese muss auch auf dieselben Testdaten zugreifen wie das IS. Zu diesem Zweck wurden beim Kopieren der Testdaten auch alle relevanten Systemdaten miteinbezogen. Am Produktionsserver ‚far-app1' wird in einem eigenen Verzeichnis ein Ordner mit der Bezeichnung ‚Default' angelegt und alle Dateien in diesen Ordner verschoben. Der übergeordnete Ordner wird als ‚DevTestAdulo' freigegeben.

Neben dem Ordner Default wird noch ein Ordner ‚Dev2009bis2013' angelegt. In diesem werden die Ordner ‚Stammdat' und ‚Jobs' aus den aktuellen Produktionsdaten kopiert. Im PCC werden die Speicherorte aller drei Datenbankdateien dementsprechend angepasst.

In Adulo gibt es die Möglichkeit, mit verschiedenen sogenannten Sektionen zu arbeiten. Hier handelt es sich um verschiedene Datenquellen, die mit derselben Installation genutzt werden können. Beim Start der Software erfolgt eine Abfrage, mit welcher Datensektion gearbeitet werden soll.

Um eine neue Sektion einzurichten, muss in der Windows-Registry entweder in HKEY_CURRENT_USER oder HKEY_LOCAL_MACHINE ein Schlüssel namens „Data Sources" im Schlüssel Software\Adulo\Adulo erstellt werden. In diesem Schlüssel muss eine neue Zeichenfolge mit dem Namen der Sektion und dem Speicherort der Daten als Inhalt angelegt werden (zum Beispiel "Farkalux Dev"="P:\\\\GS_11").

Anschließend wird auf dem Testsystem eine Netzlaufwerk-Verbindung mit dem Buchstaben P: auf den Freigabepfad eingerichtet. Die Software Adulo kann nun auf die Testdaten zugreifen.

Jetzt sind alle notwendigen Voraussetzungen für den Start der Software-Entwicklung unter Verwendung einer entsprechenden Testumgebung erfüllt.

7.2 Implementierung des Systems

Das Ziel der Implementierung ist einerseits die Erstellung der grundlegenden Datenbasis im SQL-Server sowie andererseits die Erstellung der Dienste und Clientprogramme auf Grundlage der Modellierungen aus den vorangegangenen Kapiteln.

7.2.1 Implementierung des Daten-Layers

Die Implementierung des Daten-Layers umfasst die Erstellung und Einrichtung der Datenbank und die Implementierung des Zugriffs darauf. Die Beschreibung der Schnittstellen zu den nächsthöheren Ebenen stellt den Abschluss dieses Abschnitts dar.

Die Strukturierung der Daten und der Zugriff darauf sind eine Kernaufgabe dieser Arbeit. Die Daten stellen gewissermaßen das Herz des Informationssystems dar, die Konsolidierung derselben ist ein Hauptziel und für den zukünftigen Erfolg des Informationssystems sehr wichtig.

7.2.1.1 Einrichten der Tabellen in der Datenbank

Der Zugriff auf die Daten soll aus dem Informationssystem selber, aus dem Fensterbauprogramm und zusätzlich noch aus diversen Auswertungen in Form von Excel-Diagrammen erfolgen. Diesen unterschiedlichen Zugriffsarten wurde hauptsächlich durch die Entscheidung für den Microsoft SQL-Server als zentralen Zugriffspunkt Rechnung getragen. Bei der eigentlichen Strukturierung der Daten müssen jetzt noch die speziellen Schnittstellen dafür berücksichtigt werden.

Die zum Entwickeln verwendete Datenbank ‚DevTestAdulo' im SQL-Server auf ‚far-app2' existiert bereits, die entsprechenden Verbindungsserver ebenfalls. Die zu erwartende Größe der Datenbank ist mit circa 1500 Datensätzen pro Jahr keine große Herausforderung und macht keine zusätzlichen Überlegungen notwendig. Jetzt können die einzelnen Tabellen in der Datenbank erzeugt werden. Als Vorlage dient das Modell aus Abschnitt 5.3.2.

Allen Tabellen gemeinsam ist dabei ein ID-Attribut, das den Primärschlüssel sowie die Identitätsspezifikation darstellt und automatisch vom DBMS vorgegeben wird. Als Datentyp für Textfelder wird generell der Typ nvarchar (= national variable character) mit einer maximalen Längenbegrenzung verwendet. Dies stellt die Speicherung im Format Unicode sicher und es wird nur der tatsächlich für die Speicherung eines jeden Datenwerts benötigte Speicherplatz verwendet [panther 2012, S. 78].

Sind alle Tabellen angelegt, werden die Relationen und die referenziellen Integritäten zueinander eingerichtet. Wichtig bei der Erstellung der Beziehungen ist die Option „Erzwingen der Fremdschlüsseleinschränkungen". Diese wird auf „Ja" gesetzt, dadurch lässt das DBMS keine Änderungen an den Daten der Spalten zu, welche die Fremdschlüsselbeziehungen ungültig machen würde [msdnsql1]. Bei den „INSERT und UPDATE-Spezifikationen" wird „Keine Aktion" ausgewählt. Dadurch ist sichergestellt, dass anstelle der Weitergabe der Änderungen an die Unterobjekte eine Fehlermeldung erzeugt wird. Diese muss vom Programm entsprechend behandelt werden. Die Daten in der Datenbank bleiben dadurch konsistent.

Für den Zugriff auf die Daten per Entity-Framework sind die Voraussetzungen im SQL-Server somit geschaffen. Für den Zugriff per ODBC aus Excel oder anderen Office-Programmen müssen noch entsprechende Sichten erstellt werden. Eine Hauptsicht mit allen verwendeten Attributen wird hierzu als ‚AuftragGesamt' angelegt. Diese sollte für die momentan vorhandenen Auswertungen reichen. Wird noch eine spezielle Sicht benötigt, wird diese während der Einbindung in die entsprechende Auswertung erstellt.

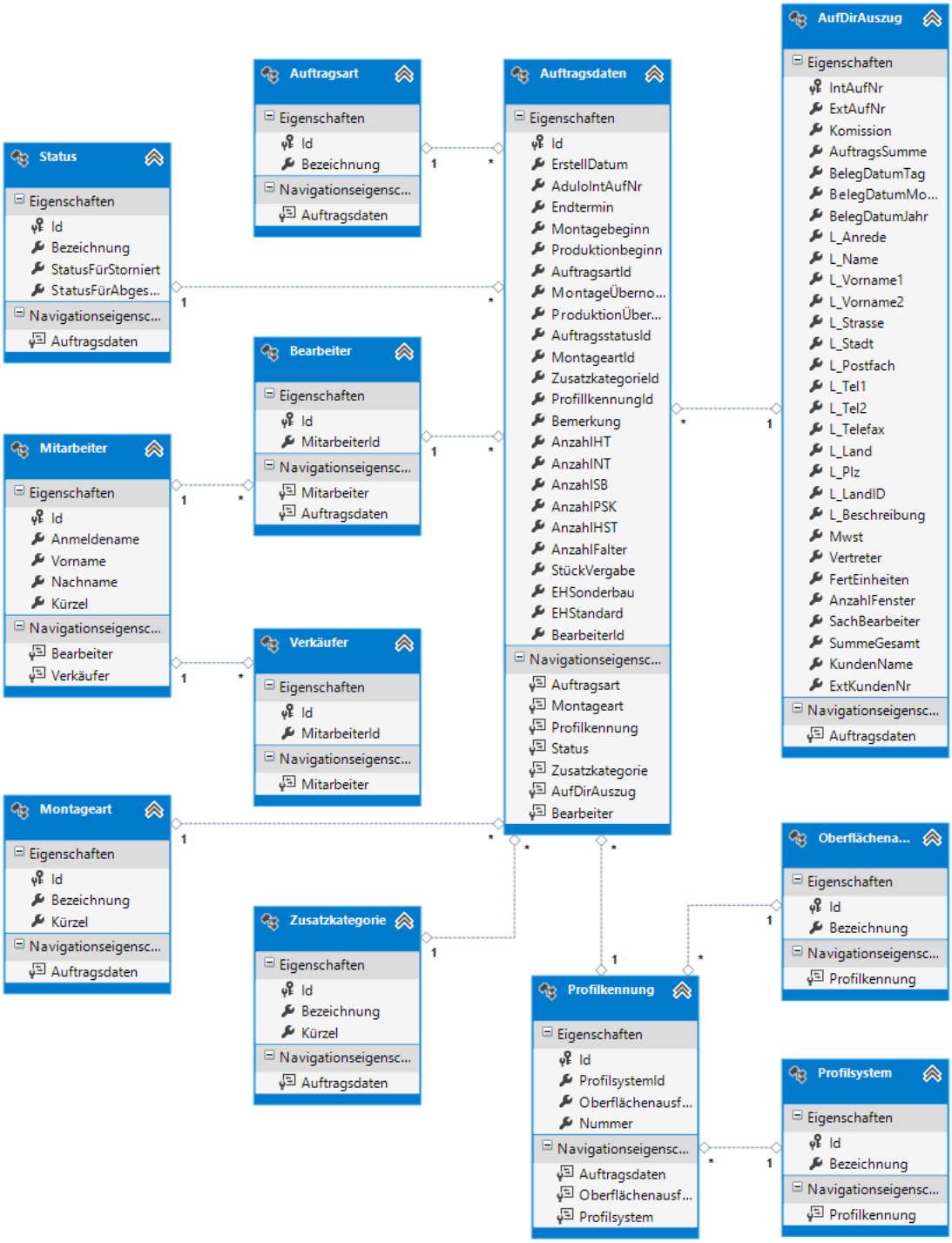


Abbildung 15: Das Entity-Framework-Modell des IS

7.2.1.2 Erstellen des EF-Modells

Zur Erstellung der Datenzugriffsschicht in .NET kommt wie bereits erwähnt der Database-First-Ansatz zur Anwendung. Dabei wird in der Entwicklungsumgebung eine Verbindung zur Datenbank hergestellt und alle notwendigen Tabellen und Sichten importiert. Im Modelldesigner werden die Tabellenzuordnungen automatisch übernommen (siehe Abbildung 15). Die Zuordnungen der Entitäten zu den Klassen werden auf den Standardeinstellungen belassen. Bei der Einbindung der beiden Views ‚AufDirAuszug' und ‚Kunden-Auszug' wird bei allen skalaren Eigenschaften außer dem Entitätsschlüssel der Zugriffsmodifizierer des Setters auf „private" festgelegt. Dadurch wird die Sicht außerhalb der Klasse als schreibgeschützt implementiert, was den Vorgaben aus Abschnitt 4.3.4 entspricht. Dieser Umstand ist bei der Ausführung des Business-Layer zu berücksichtigen.

Da der Daten-Layer als komplett eigenständige Schicht implementiert werden soll, wird dieser in einem eigenen Projekt als Klassenbibliothek entwickelt. Ein Austausch des Daten-Layers kann dann unabhängig von der Anwendung erfolgen. Bei entsprechender Konfiguration einfach durch Austausch der DLL-Datei.

Einige Anpassungen im Modell erzeugen nach erfolgreichem Einchecken die Fehlermeldung „Die an dieser Transaktion beteiligten XmlModells konnten nicht bearbeitet werden!". Eine Recherche in MSDN-Foren (Microsoft Developer Network) ergibt einen bereits bei Microsoft bekannten Bug als Ursache dieses Fehlers. Der Fehler tritt nur auf, wenn eine Quellcodeverwaltung aktiv ist, ein Entity-Data-Modell vorhanden ist und der lokale Pfad des Projekts das Zeichen „#" enthält. Nach Änderung des Pfad-Namens funktioniert die Änderung wieder wie geplant.

Sollten Änderungen an der Datenstruktur notwendig werden, müssen diese zuerst in der Datenbank abgebildet werden. Danach kann das Modell aus der Datenbank aktualisiert werden, um die Änderungen in die Klassenbibliothek zu übernehmen. Falls an der Bibliothek Erweiterungen durchgeführt werden müssen, sollten diese als eigenständige Dateien in derselben Klasse ausgeführt werden. Die automatisch erzeugten Klassen sind zu diesem Zweck als „partial" gekennzeichnet, wodurch sie über mehrere Quelltextdateien verteilt werden können [louis 2013, S. 242]. Die Implementierung der ToString-Methode in den Klassen ‚Auftraglistung' und ‚Auftragsdaten' wird auf diese Weise durchgeführt. Sie wird fürs Logging benötigt. Außerdem sorgt diese Methode für eine bessere Ansicht im Debugger.

7.2.1.3 Tests und Probleme

Die ersten Tests erfolgen manuell mithilfe einer WPF-Anwendung. Diese verwendet den Daten-Layer direkt, die abgefragten Daten werden in einem DataGrid angezeigt. Der erste Test zeigt, dass die von der Pervasive-Datenbank erhaltenen String-Daten mit CHAR‘0‘-Zeichen bis zur maximalen Zeichenlänge aufgefüllt werden. Außerdem funktioniert die Darstellung von Sonderzeichen wie ä, ö, ü, ß nicht ordnungsgemäß. Bei der Darstellung

der Sonderzeichen besteht ein Codierungs-Problem. Durch den direkten Zugriff auf die ODBC-Schnittstelle auf dem Zielserver per Connection-String kann die Encodierung mittels PvTranslate-Attribut nur entweder auf „Auto“ oder auf „Aus“ gestellt werden [pervasive5, S.460]. Bei der Angabe eines System-DSN (Data Source Name) kann die Encodierung auch auf „OEM/ANSI“ eingestellt werden. Deshalb wird ein System-DSN mit Ziel ‚farapp1‘ erstellt. Dieser DSN stellt den Zugriffspunkt für die ODBC-Verbindung dar. In der erweiterten Konfiguration kann die vorhin angeführte „OEM/ANSI-Encodierung“ ausgewählt werden. Der Verbindungsserver wird neu erstellt, dieses Mal mit Angabe der System-DSN ‚DEVTESTADULODATEN‘ als Datenquelle, ‚AufDir‘ als Speicherort und ‚DEVTESTADULODATEN‘ als Katalog. Eine erneute Abfrage per Select-Befehl im SQL-Management-Studio bestätigt die korrekte Anzeige der Sonderzeichen.

Das Problem mit den Füllzeichen am Ende der übergebenen String-Werte kann mit folgenden Varianten gelöst werden:

- in einer eigenen Klasse mit den entsprechenden Methoden in den Gettern
- am SQL-Server mit einer entsprechenden Funktion
- am Pervasive SQL-Server mit einer entsprechenden Funktion

Die Variante mit der eigenen Klasse hat die Nachteile, dass die Umwandlung erst in einer höheren Ebene erfolgt, der direkte Datenzugriff per ODBC bleibt davon unberührt. Außerdem ist die Performance um einiges schlechter.

Gewählt wird die Variante der Umwandlung am PSQL-Server. Die Umwandlung direkt an der Datenquelle ist laut Logik die effizienteste Art der Datenmanipulation. Sie wird mittels einer neu erstellten Sicht und der Funktion ‚convert(Spaltenname, „SQL_VARCHAR")‘ bei allen notwendigen Spalten durchgeführt. Die Umwandlung vom Typ CHAR mit fixer Zeichenlänge auf den Typ SQL_VARCHAR mit variabler Länge löst dabei das Problem.

Die direkte Anbindung an ein Steuerelement zur Anzeige der Daten in WPF kann mit den momentanen Klassen noch nicht optimal durchgeführt werden, weil die Informationen teilweise in Subklassen vorhanden sind. Um diese direkte Bindung zu bewerkstelligen, wird die Klasse ‚ListAuftrag‘ eingeführt, die hauptsächlich der Datenübertragung zwischen Dienst und Client dient. Sie wird mit dem Attribut [DataContract] und die jeweiligen Felder und Eigenschaften mit [DataMember] gekennzeichnet. Dies weist die WCF an, diese Daten in Form von Nachrichten zu übertragen.

Ein Test der neu implementierten Klasse zeigt enorme Performance-Probleme beim Abrufen der Daten. Am Client entsteht eine Proxy-Timeout-Exception. Auch eine Installation des SQL-Servers am selben Server der Pervasive-Datenbank bringt keine Besserung. Eine genaue Analyse des Problems und einige Tests am SQL-Server können das Hauptproblem isolieren. Dies ist bedingt durch den Umstand, dass die Auftragsdaten und die Kundendaten der Fensterbau-Software auf verschiedene Datenbanken aufgeteilt sind.

Eine Verknüpfung der Daten mittels INNER JOIN verursacht die lange Bearbeitungsdauer der Anfrage.

Zur Lösung des Problems wird die bestehende Sicht am PSQL-Server um das Feld Kundenname und Kundennummer aus der Tabelle ‚AufDir' erweitert. Die Verknüpfung mittels INNER JOIN am SQL-Server zur Tabelle ‚KundenDaten' wird entfernt. Die Kunden-Daten werden im Entity-Framework-Modell nur mehr als Sicht eingebunden, die Beziehung zu ‚AufDir' wird entfernt. Diese Beziehung wird nur mehr bei Bedarf für einzelne Aufträge im Programmcode hergestellt. Ein erneuter Test bestätigt die akzeptable Reaktionszeit beim Abruf der vollständigen Liste von unter 2 Sekunden.

Zur laufenden Sicherstellung der Zwischenergebnisse werden während der Entwicklung der Software sogenannte Unit-Tests verwendet. Unit-Tests sind kleine automatisierte Tests, die einen abgeschlossenen Code-Bereich möglichst isoliert testen sollen. Ein Unit-Test soll dabei folgende Eigenschaften aufweisen [vergl. geirhos 2013, S. 1051]:

- Er muss Wiederholbarkeit garantieren (gleiche Daten, gleiches Ergebnis).
- Er kennt nur Fail oder Pass.
- Er soll Atomar sein.
- Er muss Reihenfolge-unabhängig sein.
- Er muss isoliert sein (unabhängig von anderen Tests).
- Er soll einfach einzurichten sein.
- Er muss sehr schnell sein (wird oft ausgeführt).

Um das Informationssystem während der Entwicklung zu testen, wird ein eigenes Testprojekt erstellt, in welchem die Testklassen angelegt werden. Diesem wird der Verweis auf die Klassenbibliothek, sowie das NuGet-Paket „Entity-Framework" hinzugefügt. Zum Testen des Daten-Layers wird eine eigene Klasse angelegt. Diese enthält 3 Methoden, wobei die erste das Instanziieren des Datenkontexts und eines Objekts daraus testet. Die zweite Methode liest einen definierten Datensatz aus und überprüft das Ergebnis. In der dritten Methode wird ein neuer Datensatz angelegt, der Datenbank hinzugefügt und direkt im Anschluss erneut ausgelesen.

Alle Tests werden erfolgreich ausgeführt, die prinzipielle Funktion des Daten-Layers ist somit nachgewiesen. Fehler, die durch Änderungen unbeabsichtigt hinzugefügt werden, sollten so auch in weiterer Folge erkannt werden.

7.2.2 Implementierung des Business-Layers

Der Business-Layer ist die Verbindungsschicht zwischen Daten und Anzeige. Er beinhaltet die eigentlichen Geschäftsregeln, also die Aufgabe des MAT-Systems. Für die Auftragsabwicklung besteht der Business-Layer aus den Diensten ‚Auftragsverwaltung' und ‚Aufträge'. Zur Erfüllung dieser Vorgaben bedient sich der Business-Layer der Daten-

schicht, diese kann allerdings physisch entkoppelt sein. Dadurch werden diese beiden Layer zu sogenannten Tiers.

Zur Umsetzung werden nun die verwenden Techniken und Verfahren festgelegt und konfiguriert, danach erfolgt die eigentliche Umsetzung der beiden Dienste.

7.2.2.1 Einführung in die Windows Communication Foundation

Die WCF eignet sich hervorragend zur Erstellung von Diensten (Services), wie sie bei dieser Aufgabenstellung benötigt werden. Ein Service muss dabei, um kommunizieren zu können, wenigstens einen Endpunkt besitzen. Dieser Endpunkt besteht aus einer Adresse, einem Binding und einem Contract (Vertrag). Die Adresse wird als URI (Uniform Resource Identifier) angegeben. Die Bindings beschreiben die Kommunikationsdetails wie zum Beispiel Verschlüsselung, Autorisierung, Art der Nachrichten, usw. Die Contracts sind die Verbindung eines Service nach außen. Sie stellen die Leistung des Dienstes plattformunabhängig zur Verfügung. Die Konfiguration wird in der Datei ‚web.config' in der Sektion ‚system.serviceModel' gespeichert.

Beim Hosten der Dienste im IIS kann die Angabe der Adresse entfallen, diese wird von der hostenden virtuellen Website übernommen. Als Binding wird das TcpNetBinding verwendet. Dieses überträgt die Nachrichten binär und setzt direkt auf TCP auf. Dies ist eine effiziente Variante zwischen WCF-Systemen [steyer 2013, S. 17]. Als Contract dient der Verweis auf die Interface-Schnittstelle des jeweiligen Service, die alle Methoden des Dienstes definiert.

Die vorhin angeführte Datei ‚web.config' ist eine XML-Datei, welche die Konfigurationen der Endpunkte eines WCF-Dienstes beschreibt. In diesem Fall enthält sie einen NetTcp-Endpunkt. Bei diesem wird Port-Sharing auf „True" gesetzt, damit sich mehrere Dienste denselben Port teilen können. Für den Service wird die Veröffentlichung der Meta-Daten per http und die Übermittlung von Fehlermeldungen eingeschaltet. Der relevante Ausschnitt der WCF-Konfiguration der Datei ‚web.config' ist in Abbildung 16 ersichtlich.

```
<bindings>
  <netTcpBinding>
    <binding name="NetTcpBinding" portSharingEnabled="true" >
      <security mode="Transport">
        <transport clientCredentialType="Windows" protectionLevel="None" />
      </security>
    </binding>
  </netTcpBinding>
</bindings>
<behaviors>
  <serviceBehaviors>
    <behavior name="MetadataBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

Abbildung 16: Ausschnitt aus ‚web.config' des Auftragsverwaltungsservice

Bezüglich der Sicherheit der übertragenen Nachrichten wird für das Informationssystem die Verschlüsselung und Signierung auf der Ebene des Transportprotokolls gewählt (security mode="Transport", siehe Abbildung 16). Damit ein Benutzer vom System als solcher erkannt und authentifiziert wird, werden sogenannte Credentials verwendet. Für das IS wird als Typ „Windows" verwendet, d.h. es werden Kerberos- oder NTLM-Tickets (NT LAN Manager) übertragen [vergl. steyer 2013, S. 113].

7.2.2.2 Einrichten des Hosts

Laut Abschnitt 6.2.3 wird als Host der Internet Information Server verwendet. Dieser wird aus Performance-Gründen auf demselben Server installiert wie der verwendete SQL-Server (auf dem Applikationsserver ‚far-app2'). Er könnte aber auch auf einem separaten System laufen. Im IIS werden zwei neue Anwendungspools erstellt, einer zur Entwicklung (‚Development') und einer für das fertige Produktionssystem (‚Production'). Diese werden mit einem jeweils neu erstellten Benutzerkonto ausgeführt, das nur die benötigten Berechtigungen in der Datenbank besitzt. Somit kann nicht versehentlich während der Entwicklung mit den Produktionsdaten gearbeitet werden.

Die Services werden in neu erstellten Websites gehostet, diese hören standardmäßig auf den Port 8080. Die Berechtigungen entsprechen denen der zugehörigen Anwendungspools. Somit erfolgt eine klare Abtrennung durch getrennte Prozesse gegenüber den bestehenden Strukturen – im Fehlerfall sollen andere Systeme davon nicht betroffen sein.

Nach Installation der Windows Prozess Activation Services und den dafür notwendigen Diensten sowie der Aktivierung von ASP.NET am IIS, kann die App-Fabric installiert werden. Mit der entsprechenden Konfiguration ist der Host einsatzbereit. Zusätzlich wird noch das .NET-Framework 4.5 installiert, damit die Dienste mit der aktuellsten Version des Frameworks entwickelt werden können.

Für das Deployment der Dienste werden die kompilierten Assemblys in die entsprechenden Ordner kopiert und jeweils die Dateien ‚web.config'- und ‚service.svc' erstellt. In Visual Studio 2012 kann ein Veröffentlichungsprofil erstellt werden, das die Aktionen für das sogenannte „One-Click-Deployment" festlegt. Dieses wird entsprechend eingerichtet, damit die Dienste komfortabel mit einem Befehl an den Host übertragen werden.

7.2.2.3 Einrichten der Authentifizierung und Autorisierung

Ziel der Authentifizierung ist zu erkennen, welcher Benutzer eine Funktion ausführt. Mit dieser Information kann mittels Autorisierung das Recht auf die Verwendung der Funktion erteilt oder verweigert werden. Im Fall der geforderten rollenbasierten Autorisierung geschieht dies, indem den Benutzern unterschiedliche Rollen zugewiesen werden. Jede Rolle besitzt entsprechende Rechte für die unterschiedlichen Methoden der Dienste.

Im Abschnitt 6.2.6 wurde die Autorisierung in WCF per Windows-Gruppen festgelegt. Hier wird der Benutzer mit dem Windows-Verzeichnisdienst AD (Active Directory) von Micro-

soft authentifiziert. Dies geschieht im Regelfall bei der Anmeldung des Benutzers am Client-PC mit Benutzername und Passwort. Beim Starten einer Anwendung wird die Authentizitäts-Information an die Anwendung übergeben.

Zur effizienteren und übersichtlicheren Vergabe von Zugriffsrechten existieren im AD Gruppen. Sie können mehrere Benutzer, Gruppen oder Objekte zusammenfassen. Benutzer können dabei Mitglieder von keiner, einer oder mehreren Gruppen sein.

Die Zuordnung der Rollen zu den Benutzern erfolgt im AD durch Überprüfen der Gruppenmitgliedschaft des zu autorisierenden Benutzers für die entsprechende Rolle. Die Rollen Verkauf, Technik, Montage und Produktion sind bereits vorhanden und die entsprechenden Benutzer zugeordnet. Diese Struktur reicht im Moment aus.

Die Implementierung in den Diensten und Clients kann entweder imperativ oder deklarativ erfolgen. Bei der imperativen Autorisierung wird im Code explizit durch Aufruf der IsInRole-Methode und entsprechender Verzweigung reagiert. Dadurch kann sehr fein gesteuert werden, an welchen Stellen in der Business-Logik die Autorisierung durchgeführt wird [meier 2008, S. 101].

Bei der deklarativen Autorisierung werden ganze Methoden oder Klassen mit entsprechenden Attributen versehen. Diese Methode ist besonders für WCF geeignet, weil Attribute als Metadaten übertragen werden können. Somit können autorisierte Rollen ausgelesen werden. Die Zugriffsbeschränkung gilt allerdings für die ganze Klasse bzw. Methode [meier 2008, S. 102].

Für das Informationssystem wird die deklarative Methode aufgrund der besseren WCF-Unterstützung und der übersichtlicheren Implementierung verwendet. Die Methoden der Dienste müssen dementsprechend fein aufgeteilt werden. Eine Überprüfung hinsichtlich der Autorisierung ergibt keine Änderung an den Vorschlägen, dies wurde bereits implizit berücksichtigt.

7.2.2.4 Einrichten des Logging und Exception-Handlings

Um die Vorgaben aus Abschnitt 4.3.4 bezüglich des Loggings zu erfüllen, wird pro Auftrag eine eigene Textdatei mit den Logging-Informationen benötigt. Diese soll alle Statusänderungen sowie zusätzliche Daten wie Username, Zeitpunkt, Start und Beendigung des Auftrages und die dazugehörige ID (Identifikator) beinhalten.

Die Logging-Informationen von den Diensten sollen jeweils in ein sogenanntes Rolling-Flat-File geschrieben werden. Hier wird in eine Textdatei geloggt und in dieser werden nach Überschreiten einer gewissen Größe und/oder eines gewissen Alters der Einträge die Logging Informationen überschrieben. Dies wirkt einem Überlaufen der Dateigröße entgegen. Zur Unterstützung des Administrators sollen alle auftretenden Exceptions zusätzlich in das Windows-Event-Log geschrieben werden.

Um diese Vorgaben umzusetzen, wird die Enterprise-Library verwendet. Sie bietet granulare Einstellmöglichkeiten und zusätzlich noch Frameworks für die Daten-Validierung und für das Exception-Handling. Letzteres kann mit dem Logging-Application-Block kombiniert werden, was den Hauptgrund für die Entscheidung zur Verwendung der Enterprise-Library ausmacht.

Die Installation erfolgt per NuGet, zusätzlich wird noch die Konfigurationskonsole als Visual Studio Erweiterung installiert. Mit dieser werden die Kategorien „General“, „Service“, „Database“ und „Workflow“ angelegt. Als Trace-Listener werden ein Event-Log-Listener, ein Rolling Flat-File-Listener und drei File-Listener angelegt und mit den entsprechenden Kategorien verbunden. Die fertige Konfiguration ist in Abbildung 17 ersichtlich.

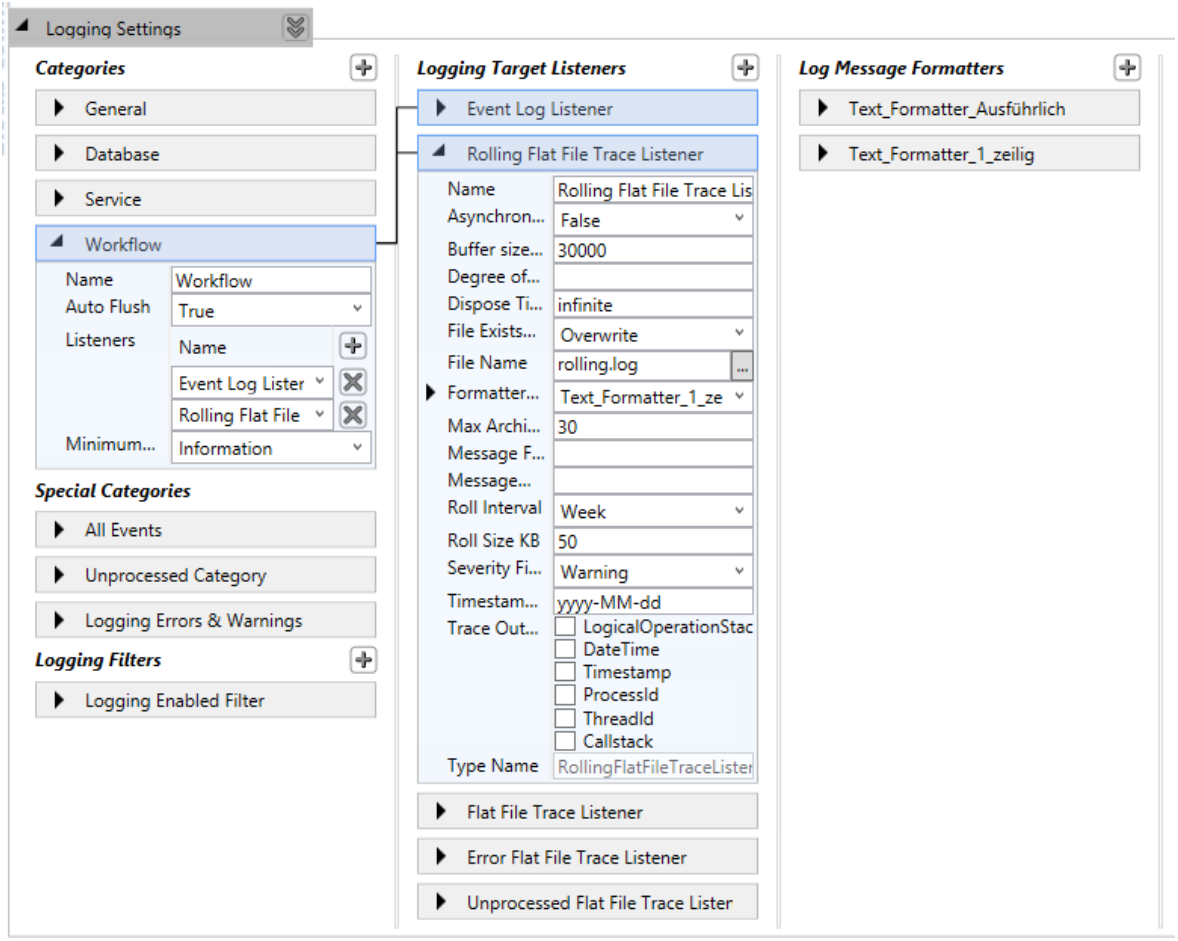


Abbildung 17: Logging-Konfiguration der Enterprise Library

Das eigentliche Loggen selber erfolgt im Programm durch Initialisieren eines statischen Loggers und durch Aufruf der Methode ‚Logger.Write‘ mit Übergabe der entsprechenden Daten. Die Initialisierung eines statischen Objekts in einem Dienst muss allerdings koordiniert werden. Da für jede Anfrage an den Dienst eine eigene Instanz verwendet wird, muss der Status der Initialisierung in einer statischen Variablen gespeichert werden. Außerdem muss mittels eines Semaphors die Initialisierung threadsicher gestaltet werden. Als Semaphor wird eine statische LockObject-Variable eingeführt, diese muss ein Refe-

renztyp sein. Der kritische Abschnitt wird mittels einer lock-Anweisung auf diese statische Variable umschlossen. In Abbildung 18 ist der fertige Code der Methode SetStaticClasses ersichtlich, der auch den Exception-Manager und die Validation-Factory initialisiert.

```
private static bool _isStaticClassesSet = false;    //statische Variable, welche die
Initialisierung der statischen Klassen angibt

private static readonly object LockObject = new object(); //dient als Sperrvariable
private void SetStaticClasses()
{
    if (_isStaticClassesSet) return;
    else
    {
        lock (LockObject) //Damit nicht mehrere Threads gleichzeitig initialisieren
        {
            if (_isStaticClassesSet) return;   //Erneute Abfrage, damit wartender
                                        Thread nicht erneut Initialisierung aufruft
            else
            {
                //Logging initialisieren
                var logWriterFactory = new LogWriterFactory();
                LogWriter defaultWriter = logWriterFactory.Create();  //instanziiert
                                    einen Log-Writer laut Konfiguration in web.config
                Logger.SetLogWriter(defaultWriter);//setzt den statischen Log-Writer
                //ExceptionHandler initialisieren
                var policyFactory = new ExceptionPolicyFactory();
                var exManager = policyFactory.CreateManager(); //instanziiert einen
                                    Exception-Manager laut Konfiguration in web.config
                ExceptionPolicy.SetExceptionManager(exManager); //setzt den
                                                        statischen Exception-Manager
                //Die statische ValidationFactory Klasse mit den Einstellungen aus
                            web.config konfigurieren
                ValidationFactory.SetDefaultConfigurationValidatorFactory(new   Sys-
temConfigurationSource(false));
                _isStaticClassesSet = true; //auf initialisiert setzen
            }
        }
        Logger.Write("Statische Klassen initialisiert", "General", -1, 1, TraceE-
ventType.Verbose, "Initialisierung");
    }
}
```

Abbildung 18: Initialisierung der statischen Elemente

Beim Exception-Handling erfolgt beim Behandeln einer Exception in einem catch-Block ebenso der Aufruf einer statischen Methode, die das Logging, das Ändern auf eine andere Exception oder das Verwerfen übernimmt. Dieses Verhalten wird mittels der Konfiguration gesteuert. Hierzu können verschiedene Policys angelegt werden, welchen die unterschiedlichen Exception-Kategorien zugeordnet werden können. Für das Informationssystem wird eine Policy mit dem Namen ‚Unhandeld' eingerichtet, die alle Exceptions entgegennimmt, diese entsprechend der Logging-Konfiguration loggt und danach weitergibt. Eine exemplarische Verwendung des Exception-Handlings ist in Abbildung 19 ersichtlich.

Alle abgefangenen Ausnahmen sollen in das Windows-Event-Log im Bereich Anwendungen geschrieben werden. Um dieses Loggen zu ermöglichen, muss der Enterprise-Library Logging-Application-Block dem System als Event-Quelle bekannt gegeben werden. Dies

geschieht in der Registry durch Hinzufügen des Schlüssels „[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\Enterprise Library Logging]“ und des Zeichenfolgenwertes "EventMessageFile"="C:\Windows\Microsoft.NET\Framework\v4.0.30319\EventLogMessages.dll". Für die Client-Anwendung gilt derselbe Schlüssel, alternativ kann die Anwendung einmal mit Administratorberechtigung ausgeführt werden, dann wird der Schlüssel automatisch angelegt.

```
try
{
    //zu überwachender Codeabschnitt
}
catch (Exception ex)
{
    bool rethrow = ExceptionPolicy.HandleException(ex, "Unhandled");
    if (rethrow) throw;
}
```

Abbildung 19: Verwendung des Exception-Handling-Application-Blocks

7.2.2.5 Einrichten der Validierung

Die Validierung der Daten vor der permanenten Speicherung in der Datenbank und während der Eingabe am Client stellen eine wichtige Grundfunktion für einen korrekten Datenbestand dar. Zur einheitlichen Durchführung wird dafür ebenfalls ein Block aus der Enterprise-Library verwendet. Der Validation-Application-Block bietet analog zum Logging und zum Exception-Handling zentrale Konfigurierbarkeit in den config-Dateien.

Die Validierungsregeln selber können dabei mittels Attributen bei den entsprechenden Eigenschaften, per zentral angelegten Regeln direkt im Konfigurationsfile oder mittels einer Methode in jeder Klasse umgesetzt werden. Die Auszeichnung mit Attributen ist die einfachste Variante, hier können allerdings nur grundlegende Überprüfungen durchgeführt werden (zum Beispiel Überprüfung auf ein Pflichtfeld, Prüfung des Wertebereichs). Die Definition der Regeln in den Konfigurationsfiles bietet zusätzliche Überprüfungsmöglichkeiten, allerdings müssen diese in den Konfigurationen eines jeden Dienstes repliziert werden.

Die Methode der Self-Validation ist für diese Aufgabenstellung am besten geeignet. Sie bietet die meisten Einstellungsmöglichkeiten, zum Beispiel auch Überprüfung auf Kombinationen von Eigenschaften. Hier wird die Validierungsmethode zentral in jeder Klasse angelegt und ist somit für alle Applikationen, welche die Klasse verwenden, verfügbar. Der Nachteil der Self-Validation ist, dass diese nicht direkt an WPF-Controls gebunden werden kann. Dieser Nachteil kann jedoch durch implementieren der Methoden der IDataErrorInfo-Schnittstelle kompensiert werden.

Wird diese Schnittstelle eingesetzt, so muss eine Methode angelegt werden, die für jede Eigenschaft entweder eine leere Zeichenfolge oder den Validierungsfehler zurückgibt. Somit wären die Validierungsregeln aber an zwei verschiedenen Stellen in der Anwendung bzw. Klasse definiert.

Um diesen Missstand zu beseitigen, wird nun eine allgemeine Methode laut Abbildung 20 eingeführt, die für jede Eigenschaft die entsprechenden Überprüfungen durchführt.

```
private string ValidateProperty(string columnName){
try
{
    switch (columnName)
    {
        case "ErstellDatum":
            if (this.ErstellDatum == null) return columnName + " ist ein Pflicht-
feld!";
            if (this.ErstellDatum < _minDatum) return "Erstelldatum ist ungültig!";
        break;
        case "Montagebeginn":
            if   ((this.Montagebeginn   <   _minDatum)   ||   (this.Montagebeginn   >
this.Endtermin)) return "Montagebeginn ungültig!";
        break;

             //Auszug gekürzt!!

        case "Bemerkung":
            if (this.Bemerkung.Length > 50)
                return "Bemerkung zu lang!";
            break;
        default:
            return string.Empty;
    }
}
catch (NullReferenceException)
{}
return string.Empty;
}
```

Abbildung 20: Auszug der Methode ‚ValidateProperty'

Diese Methode kann direkt aufgerufen werden (siehe Abbildung 21) – dies wird für die Methode IDataErrorInfo genutzt. Der Aufruf aus der Self-Validation-Methode erfolgt dann per Reflektion. Dabei wird der Name jedes Eigenschaftswertes ausgelesen und damit die Methode ValidateProperty aufgerufen (siehe Abbildung 22). Gibt es eine Verletzung der Validierungsregeln wird der entsprechende Text zurückgegeben.

```
//IDataErrorInfo
public string this[string columnName]
{
    get { return ValidateProperty(columnName);
}
```

Abbildung 21: Aufruf der Methode ‚ValidateProperty'

Der Aufruf der Validierung erfolgt im Code explizit durch Aufruf der statischen Methode ‚Validator.Validate()' oder implizit mithilfe der Controls und der IDataErrorInfo-Schnittstelle. Die Validate-Methode liefert dabei eine Auflistung aller Meldungen der Validierungsfehler zurück. Diese kann zur Ausgabe von Informationen an den Benutzer verwendet werden.

Die Funktion dieser Validierung wird zur Sicherstellung des Teilergebnisses mittels eines Unit-Tests überprüft. Dadurch ist außerdem sichergestellt, dass ungewollte Funktionsänderungen aufgedeckt werden.

```
[SelfValidation]
public void Validate(ValidationResults results)
{
    foreach (var prop in typeof(Auftraglistung).GetProperties())
    {
        string msg = ValidateProperty(prop.Name);
        if (!string.IsNullOrEmpty(msg)) results.AddResult(new ValidationResult(msg,
this, "AuftraglistungSelfValidation", "", null));
    }
}
```

Abbildung 22: Die Methode Validate der Self-Validation

7.2.2.6 Erstellen des Auftragsverwaltungsdienstes

Der Auftragsverwaltungsdienst stellt den Zugriffspunkt für sämtliche Operationen der Auftragsbegleitung dar. Er stellt Funktionen zum Anlegen, Ändern und Löschen von Aufträgen im IS bereit. Außerdem können viele Statusänderungen, eine Suche nach Aufträgen und der Export der Neuaufträge für die Produktion und Montage damit durchgeführt werden. Damit diese Aufgaben durchgeführt werden können, stellt der Dienst mehrere Methoden zur Verfügung. Diese Methoden sind im Klassendiagramm in Abbildung 23 ersichtlich.

Beim Aufruf einer Methode soll immer eine eigene Instanz des Dienstes die Methode abarbeiten und die entsprechende Antwort senden. Diese Strategie erzeugt zustandslose Services und verringert somit die Komplexität sowie Probleme in Hinblick auf Nebenläufigkeit und Lastverteilung. Zustandslose Services stellen aus diesen Gründen eine Best Practice im Bereich verteilter Systeme dar [steyer 2013, S. 45]. Dieses Verhalten wird erreicht, indem die Klasse ‚AuftragsverwaltungService' mit dem Attribut [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)] gekennzeichnet wird.

Die Methoden sollen alle auch in einer asynchronen Variante angeboten werden. Dies ist besonders wichtig für zukünftige Clients, die auch mit Fingerbedienung zurechtkommen sollen. Hier gelten üblicherweise maximale Reaktionszeiten von 50ms als akzeptabel für eine flüssige Bedienung [geirhos 2013, S. 217]. Diese Zeit kann bei einem Serviceaufruf schnell überschritten werden. Mit Visual-Studio können asynchrone Methoden durch Anwählen einer entsprechenden Option beim Erstellen des Dienst-Verweises am Client erzeugt werden. Auf Service-Seite ist somit keine Berücksichtigung notwendig.

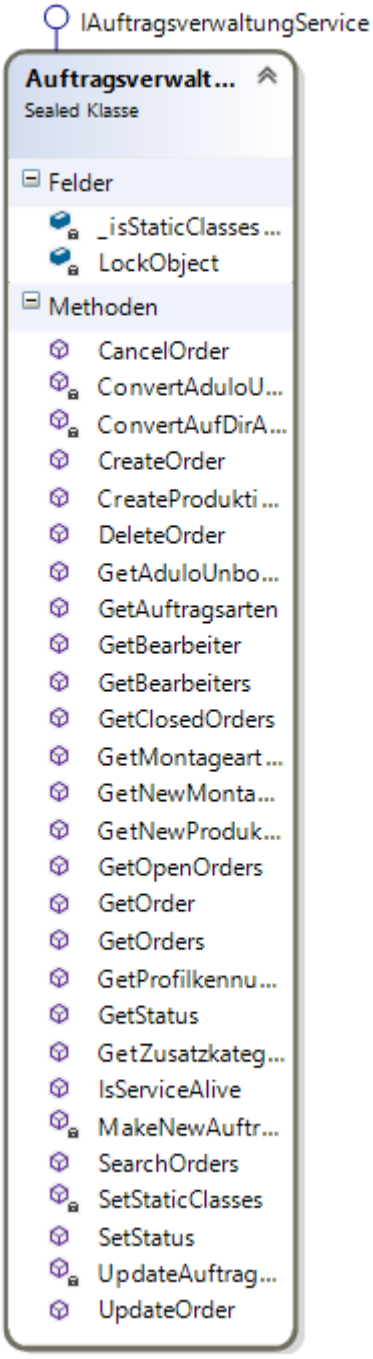


Abbildung 23: Klassendiagramm des Auftragsverwaltungs-Service

Der Auftragsverwaltungsdienst nutzt intensiv den Daten-Layer, der durch Einbindung der ‚AuftragsverwaltungsLibrary' dem Service zur Verfügung gestellt wird. Dazu müssen auch die Pakete des Entity-Frameworks per NuGet eingebunden werden. Außerdem müssen die entsprechenden Konfigurationselemente und die Verbindungszeichenfolge in der bereits erwähnten ‚web.config'-Datei erstellt werden. Eine Verwendung der Datenschicht ist am Beispiel der einfachen Methode ‚GetOrders' in Abbildung 24 ersichtlich. Die Methode liefert als Ergebnis eine Auflistung aller in der Datenbank vorhandenen Aufträge, nach Auftragsnummer aufsteigend sortiert.

```
[PrincipalPermission(SecurityAction.Demand, Role = "Farkalux")]
public List<Auftraglistung> GetOrders()
{
    SetStaticClasses();
    Logger.Write("GetOrders: Gestartet von User " + Thread.CurrentPrincipal.
Identity.Name);
    var auftraege = new List<Auftraglistung>();
    try
    {
        using (var db = new DevTestAduloEntities())
        {
            auftraege.AddRange(db.Auftraglistung.OrderBy(a => a.Auftragsnummer));
        }
    }
    catch (Exception ex)
    {
        bool rethrow = ExceptionPolicy.HandleException(ex, "Unhandled");
        if (rethrow) throw;
    }
    Logger.Write(auftraege.Count.ToString() + " Aufträge ausgegeben");
    return auftraege;
}
```

Abbildung 24: Die Methode ‚GetOrders'

Die Umsetzung der Methode GetAduloUnboundOrders bereitet einige Probleme. Diese Funktion soll eine Auflistung aller Aufträge in der Fensterbau-Software, denen kein Auftragsstatus im IS zugeordnet ist, liefern. Dies kann mit LINQ-to-Entities mit der Methode ‚Except' und der Verwendung des Fremdschlüssels AduloIntNr in beiden relevanten Tabellen realisiert werden. Diese Abfrage zeigt sich sehr performant, dies wird mittels Verwendung des Remote-Debuggers und dem Setzen von entsprechenden Haltepunkten überprüft. Bei dieser Art des Debuggens wird am Host-Server ein entsprechender Dienst installiert und gestartet, mit welchem über Visual Studio eine Verbindung zu einem lokalen Prozess am Host hergestellt werden kann. Beim Erreichen eines Haltepunktes werden dann vom laufenden Prozess beinahe dieselben Informationen angezeigt wie bei einer lokalen Debugging-Sitzung.

Das Problem entsteht allerdings bei der Übertragung der Liste auf den Client. Da in der Abfrage nur der Ausschluss der gemeinsamen Eigenschaft ‚AduloIntNr' ausgewertet werden kann, ist das Zwischenergebnis eine Auflistung aller internen Auftragsnummern in Adulo, die nicht im IS existieren. Die zu jeder internen Nummer zugehörigen Daten müssen allerdings Satz für Satz nachgeladen werden. Im SQL-Profiler zeigt sich dies anhand vieler einzelner Select Abfragen. Diese müssen alle vom SQL-Server angenommen und einzeln an den PSQL-Server weitergeleitet werden. Dadurch entsteht unzumutbare, durchschnittliche Ladezeit von ca. 15 sec.

Da im DBMS keine Relationen zur Tabelle AufDirAuszug vorhanden sind, funktioniert auch das sogenannte Eager-Loading (vorausschauendes Laden von verknüpften Entitäten) nicht. Eine Verlagerung der Abfrage auf den SQL-Server bringt zwar eine kleine Verbesserung, aber immer noch nicht das gewünschte Ergebnis.

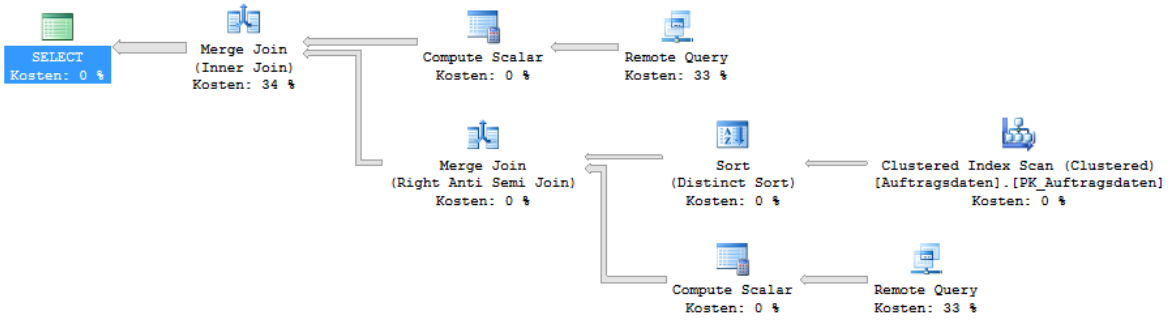


Abbildung 25: Geschätzter Ausführungsplan der Abfrage ‚AduloUnboundOrders'

Eine Analyse der Abfrage mit Hilfe der Auswertung des geschätzten Ausführungsplans im SQL-Server fördert dann die optimalste Lösung zu Tage. Diese wird in Form einer Sicht im SQL-Server umgesetzt. Das Abrufen der Sicht im SQL-Server dauert 2 sec, der Aufruf der Dienstfunktion dauert 3 sec. Der Ausführungsplan ist in Abbildung 25 ersichtlich. Die entscheidende Änderung ist die Verwendung von „NOT IN" anstelle der Except-Funktion. Die fertige Abfrage ist in Abbildung 26 zu sehen. Sie wird im EF-Modell eingebunden und steht somit im Dienst als Klasse zur Verfügung. Die korrekte Funktion dieser Methode wird ebenfalls mit einem Unit-Test sichergestellt.

```
SELECT dbo.AufDirAuszug.IntAufNr, dbo.AufDirAuszug.ExtAufNr, … --Abfrage gekürzt
FROM   dbo.AufDirAuszug INNER JOIN
      (SELECT        IntAufNr AS Nr
       FROM            dbo.AufDirAuszug AS AufDirAuszug_1
       WHERE        (IntAufNr NOT IN
            SELECT   AduloIntAufNr AS Nr
            FROM     dbo.Auftragsdaten))) AS A ON A.Nr = dbo.AufDirAuszug.IntAufNr
```

Abbildung 26: Abfrage am SQL-Server für die Sicht ‚AduloUnboundOrders'

Der Export aller neu angelegten Aufträge im System für die Produktionssteuerung wird mit der Funktion ‚CreateProduktionExport' realisiert. Um die Daten in ein Excel-kompatibles Format zu speichern, stehen folgenden Möglichkeiten zur Verfügung:

- Zielformat .xlsx durch Verwendung der excel.interop Bibliotheken
- Zielformat Open XML
- Zielformat Closed XML (über Umweg einer XML-Datei)
- Zielformat csv

Die Verwendung der Excel.interop-Bibliotheken scheidet aus, weil dafür Excel auf dem Zielsystem installiert sein muss. Die Lösungen mit den XML-Formaten sind relativ aufwändig umzusetzen, dies gilt vor allem für Open XML.

Die Exportliste als kommagetrennte Liste im csv-Format ist für diesen Anwendungsfall ausreichend. Die importierten Datensätze werden in der Produktionsliste ohnehin neu formatiert, weshalb nur die Tabelleninhalte wichtig sind. Die csv-Datei kann mittels eines Streamwriters erzeugt werden. Die erzeugte Liste wird manuell per Excel geöffnet und die neuen Datensätze werden in die Zieltabelle übertragen. Beim ersten Test stellt sich her-

aus, dass die Umlaute nicht korrekt dargestellt werden. Damit die Export-Datei korrekt gespeichert wird, muss der Streamwriter mit dem Attribut „Encoding.Default“ initialisiert werden.

7.2.2.7 Erstellen des Auftragsdienstes

Der Auftragsdienst stellt die eigentliche Abbildung des Prozesses eines Kunststofffensterauftrags im System dar. Diese Abbildung wird mittels der in Kapitel 3.4 vorgestellten Workflow-Foundation bewerkstelligt. Mit der WF können prozessorientierte Problemfälle grafisch umgesetzt werden. Ein Ausschnitt des entsprechenden Workflows ist in Abbildung 27 ersichtlich.

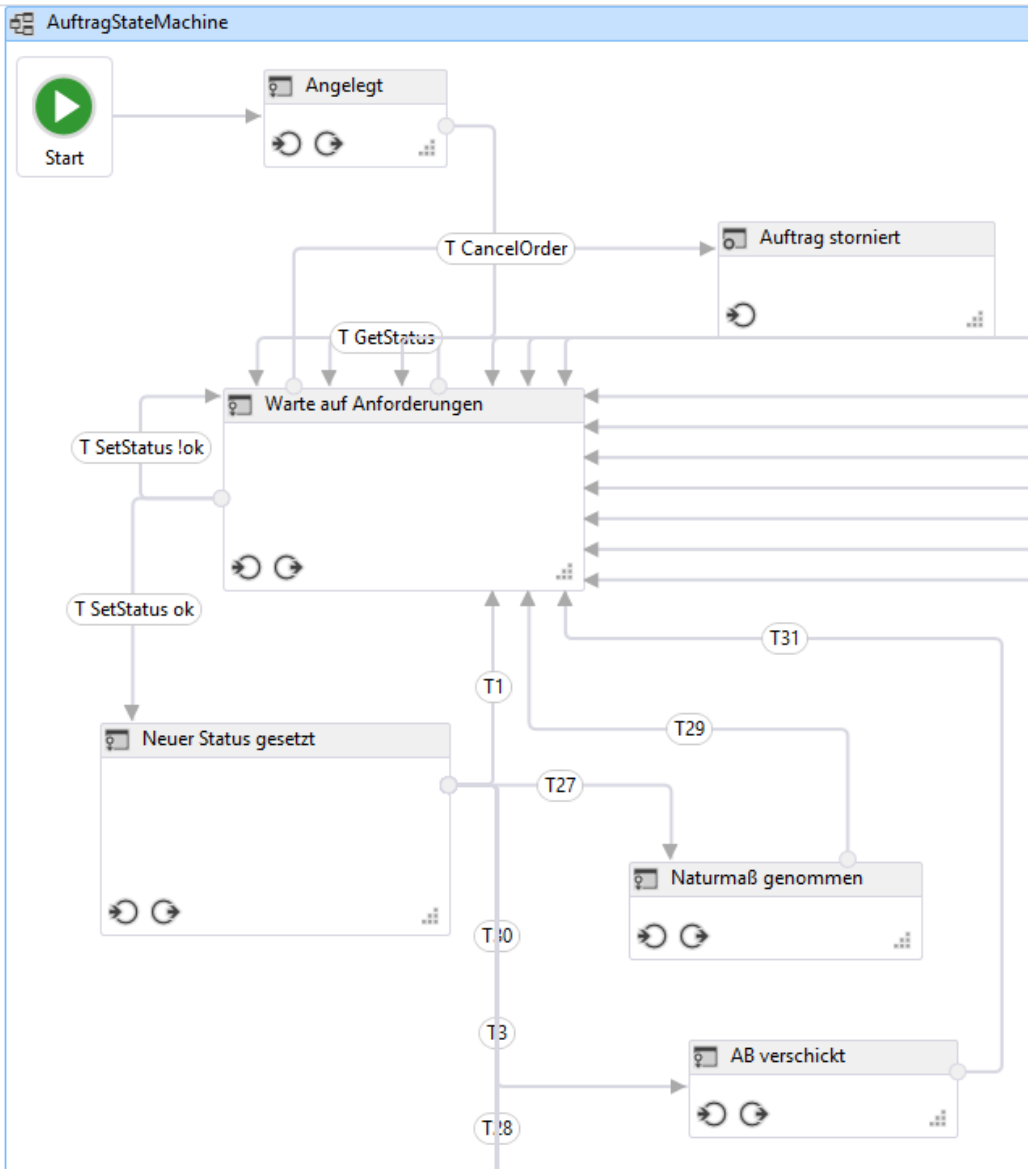


Abbildung 27: Ausschnitt des AuftragService-Workflows

Damit Workflows in einem Service verwendet werden können, muss entweder im Service selbst ein Workflow-Host implementiert werden oder in Verbindung mit WCF direkt gehos-

tet werden. Somit besteht die Möglichkeit, einen Workflow eigenständig ablaufen zu lassen. Diese Art der Implementation wird Workflow-Service genannt [steyer 2013, S. 489].

AppFabric bietet als Host für Workflow-Services zusätzliche Monitoring-Funktionen wie erweitertes Logging und ein übersichtliches Dashboard. Außerdem können Konfigurationseinstellungen komfortabler geändert bzw. eingerichtet werden. Damit alle Funktionen von AppFabric verwendet werden können, muss der Dienst mit einem zusätzlichen Net.Pipe Endpunkt ausgestattet werden. Dies geschieht durch Anpassen der Sektion ServiceModel in der web.conifg-Datei (siehe Abbildung 28). Zusätzlich muss in der hostenden Site named.pipe als Binding hinzugefügt werden.

```
<system.serviceModel>
    <extensions>
      <behaviorExtensions>
        <add    name="workflowServiceTrace"    type="Microsoft.Activities.Extensions.
Diagnostics.WorkflowServiceTraceElement, Microsoft.Activities.Extensions" />
      </behaviorExtensions>
    </extensions>
    <bindings>
      <netNamedPipeBinding>
        <binding name="NetPipeBinding" />
      </netNamedPipeBinding>
```

Abbildung 28: Auszug aus ‚web.config' für den Auftrag-Service

Für das verbesserte Debuggen von Workflow-Services gibt es von Microsoft die Activity-Extensions. Wird diese Erweiterung als Assembly dem Projekt hinzugefügt und der Dienst entsprechend konfiguriert (siehe Abbildung 28, Abschnitt „behaviorExtensions"), können beim Remote-Debuggen zusätzlich zur aktuellen Aktivität auch Variableninhalte angezeigt werden.

Jeder Workflow besteht aus Aktivitäten, die logisch miteinander verbunden werden und wiederum selbst aus mehreren Aktivitäten bestehen können. Als Aktivitäten kommen entweder vorgefertigte Aktionen oder aber selbst erstellte Code-Activities in Frage. Prinzipiell sind 3 Arten von Workflows möglich:

- Sequentielle Workflows
- Flowchart Workflows
- StateMachine Workflows

Für diese Aufgabenstellung kommt nur der StateMachine-Workflow in Frage, weil nur bei dieser Art Sprünge zu vorhergehenden Zuständen möglich sind. „Mit dieser Form der Umsetzung können aus der Informatik bekannte Zustandsautomaten umgesetzt werden. Dabei ist ein System zu einem gewissen Zeitpunkt immer in einem gewissen Zustand. Davon kann es mehrere geben und das System kann auf Ereignisse reagieren, indem es den Zustand wechselt, was man gemeinhin als Zustandsübergang bezeichnet. Solche Übergänge sind nun begleitet von Aktionen" [geirhos 2013, S. 894).

Der Start einer Workflow-Service-Instanz erfolgt durch Aufruf einer Receive-Aktivität. Diese muss die Eigenschaft „CanCreateInstance“ auf „true“ gesetzt haben. In unserem Fall ist dies eine zusammengesetzte Receive/SendReply-Methode mit dem Namen ‚StartAuftrag‘, die eine Rückmeldung über den Erfolg in Form einer Instanz-ID liefert (siehe Abbildung 29). Damit die Nachrichten und die Antworten darauf den Instanzen bzw. den Empfängern zugeordnet werden können, wird ein Konzept namens Korrelation verwendet. Hier kann entweder die Context-Based-Correlation oder die Contend-Based-Correlation verwendet werden. Bei der ersten Variante wird der Workflow-Context zur Zuordnung genutzt. Bei der zweiten Variante kann ein beliebiges eindeutiges Kriterium dafür verwendet werden, in unserem Fall eignet sich dafür die ID aus den Auftragsdaten hervorragend. Die Zuordnung von Antworten oder Retournachrichten aus Receive/SendReply oder Send/Receive-Aktivitäten erfolgt zusätzlich mit Request-Reply-Correlation.

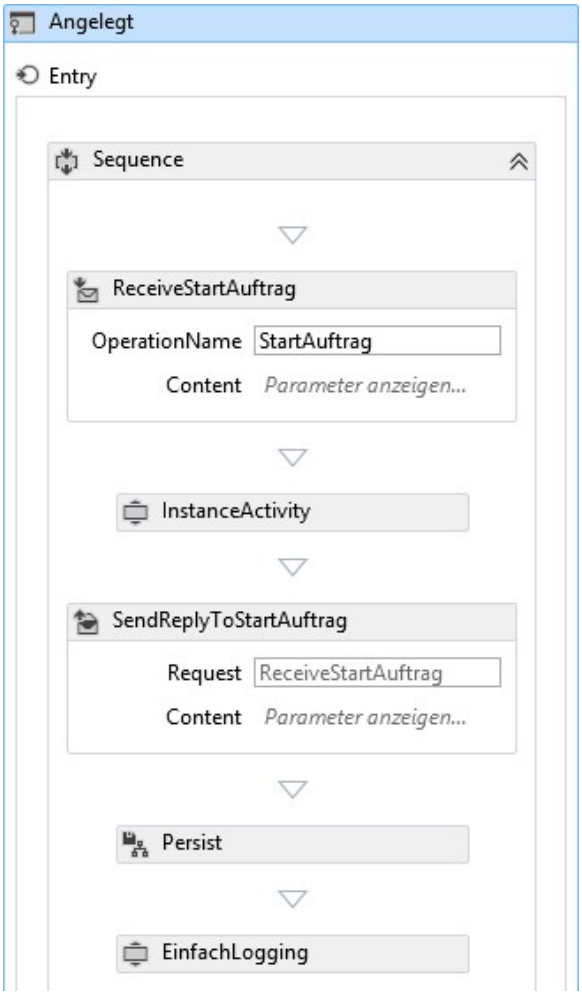


Abbildung 29: Auszug des Status „Angelegt“ mit der ‚StartAuftrag‘-Aktivität

Generell stellen Receive-Aktivitäten die Methoden des Workflow-Services dar. WCF erkennt diese Aktivitäten und erzeugt die entsprechenden Contracts. Bei der Verwendung von mehreren Receive-Aktivitäten funktioniert dies jedoch nicht mehr korrekt. Es werden jetzt 2 unterschiedliche Contracts mit separaten Endpunkten erzeugt. Die Ursache ist ein anderer Name in der Workflow-Eigenschaft „ConfigurationName“. Dieser wird gleich den

Contracts in den Receive-Aktivitäten benannt, dann funktioniert auch die Erzeugung der Dienstmethoden wieder korrekt.

Der Status des Auftrags sowie die nächstmöglichen Zustände werden in Variablen gespeichert und können mit der Methode ‚GetStatus' abgefragt werden. Eine Statusänderung wird mit der Methode ‚SetStatus' oder aber zukünftig mit speziellen Methoden zur Anbindung anderer Systeme (Bsp. Bestellverwaltung) durchgeführt. Bei erfolgreicher Statusänderung werden die Variablen aktualisiert, die Ergebnisse geloggt und der erreichte Status in die Datenbank geschrieben. Das Ende der Workflowinstanz wird entweder durch Erreichen des Zustands „Fertig" oder durch Aufruf der Methode ‚CancelOrder' ausgelöst. In beiden Fällen wird der entsprechende Status in die Datenbank geschrieben und danach die aktuelle Instanz beendet.

Das Logging erfolgt zusätzlich thread-lokal in einer Datei pro Auftrag. Dazu wird eine eigene Code-Activity erstellt, die mit der zu loggenden Meldung und dem aktuellen Status des Auftrags aufgerufen wird. Wird der Workflow abgeschlossen, wird die erzeugte Log-Datei in einen Archiv-Ordner verschoben. Dies bietet eine zusätzliche Übersicht über nicht abgeschlossene Instanzen. Für das Logging des gesamten Dienstes werden die Funktionalitäten von AppFabric verwendet, hier wird als Schwelle für den Schweregrad die Option „Warning" gewählt. Einzig für das Exception-Handling und das dazugehörige Flat-File-Logging wird wieder die bereits bekannte Enterprise-Library verwendet.

Einer der Hauptgründe für die Entscheidung zur Verwendung der Workflows für diese Aufgabe ist die Möglichkeit, laufende Instanzen zu persistieren. Für diese Aufgabe zieht die WF einen sogenannten InstanceStore (in unserem Fall der SQL-Server Express) heran. In diesem speichert sie den Zustand von sich im Stillstand befindlichen Workflows oder lädt Workflows, die weiter ausgeführt werden können. Ein Workflow befindet sich dann im Stillstand, wenn er auf das Eintreten eines Ereignisses wartet. Hierbei kann es sich – im Fall der WCF Workflow Services – um den Aufruf einer Servicemethode handeln. Neben dem automatischen Persistieren durch die Workflow Foundation existiert auch eine Aktivität, um eine Persistierung explizit anzufordern [steyer 2013, S. 503].

Von dieser Möglichkeit wird auch direkt nach der ‚StartAuftrag'-Receive/SendReply Aktivität Gebrauch gemacht. Wird dies unterlassen, wird folgende Fehlermeldung im Verlauf der Workflow-Abarbeitung gezeigt: „The execution of an InstancePersistenceCommand was interrupted because the instance has not yet been persisted to the instance store.". Microsoft empfiehlt bei Verwendung der Persistierung, diese zu einem sehr frühen Zeitpunkt explizit auszuführen [msdnwf2]. Damit wird auch gewährleistet, dass lang laufende Workflows nach einem Neustart bzw. Absturz des Rechners weiter ausgeführt oder bei Bedarf von einem anderer Rechner fortgesetzt werden können.

7.2.3 Implementierung des Presentation-Layers

Der Presentation-Layer stellt die Anzeige der Daten für den Benutzer dar. Er ist die Schnittstelle zwischen Mensch und Technik des MAT-Systems und bedient sich nur der Funktionen des Business-Layers. Da er von diesem physisch entkoppelt arbeiten kann, stellt auch der Presentation-Layer einen eigenen Tier dar.

Laut Theorie sollte die Anzeigeschicht möglichst keine Funktionen der Geschäftslogik beinhalten. Dies umzusetzen ist in der Praxis jedoch oft sehr aufwändig. Eine Methode, die diese Anforderungen erfüllt, ist die MVVW (Model-View-View Modell). Dabei stellt das Modell die Daten dar (in der Regel die direkten Klassen aus dem EF), die View ist die eigentliche Ansicht und das View-Modell ist die Verbindung der beiden. Jede Schicht kennt dabei nicht die Implementierung der anderen Schicht.

Für diese Arbeit wird dieses Pattern jedoch nicht umgesetzt. Da im IS die Kommunikation zwischen Business-Tier und User-Client über Rechnergrenzen hinweg erfolgt, werden stark vereinfachte Klassen zur schnellen Datenübertragung verwendet. Würden diese wieder mit den Informationen über das View-Modell erweitert werden, wäre die ohnehin schon kritische Performance um einiges schlechter. Würde das View-Modell auf Clientseite erzeugt werden, wäre die klare Trennung der Schichten genau zwischen Client und Dienst nicht mehr gegeben. Da das Informationssystem nur durch eine Art von Client bedient werden soll, ist der Aufwand zur Implementation um ein vielfaches höher, als der Mehraufwand bei zukünftigen Änderungen betragen wird.

Die Client-Anwendung wird als WPF-Anwendung ausgeführt. Bei der Windows Presentation Foundation erfolgt eine strikte Trennung zwischen Code und Oberfläche. Die Oberfläche wird dabei mit der auf XML basierenden Sprache XAML (Extensible Application Markup Language) beschrieben. Der Code der Anwendung wird in sogenannten Code-Behind Dateien gespeichert.

Ein Vorteil von WPF ist die einfache Möglichkeit der Datenbindung an Controls oder an Container, dann werden die Bindungen sogar vererbt. Von dieser Möglichkeit wird in dieser Arbeit reichlich Gebrauch gemacht. Als Datentyp für Auflistungen eignet sich besonders die generische Liste „ObservableCollection“, welche die Fähigkeit besitzt, Controls bei Datenänderungen zu benachrichtigen, damit diese die Anzeige aktualisieren. Dieser Typ wird zum Beispiel bei der Anbindung der Auftragsliste an die ListView im Hauptfenster des IS verwendet, wie in Abbildung 30 ersichtlich. Dabei ist das ListView-Control selbstständig für die Anzeige der Daten verantwortlich.

Der Client wird in einer Grundausführung für alle Bereiche von Farkalux ausgeführt. Zusätzliche Controls oder Optionen werden mittels Autorisierung ein- oder ausgeblendet.

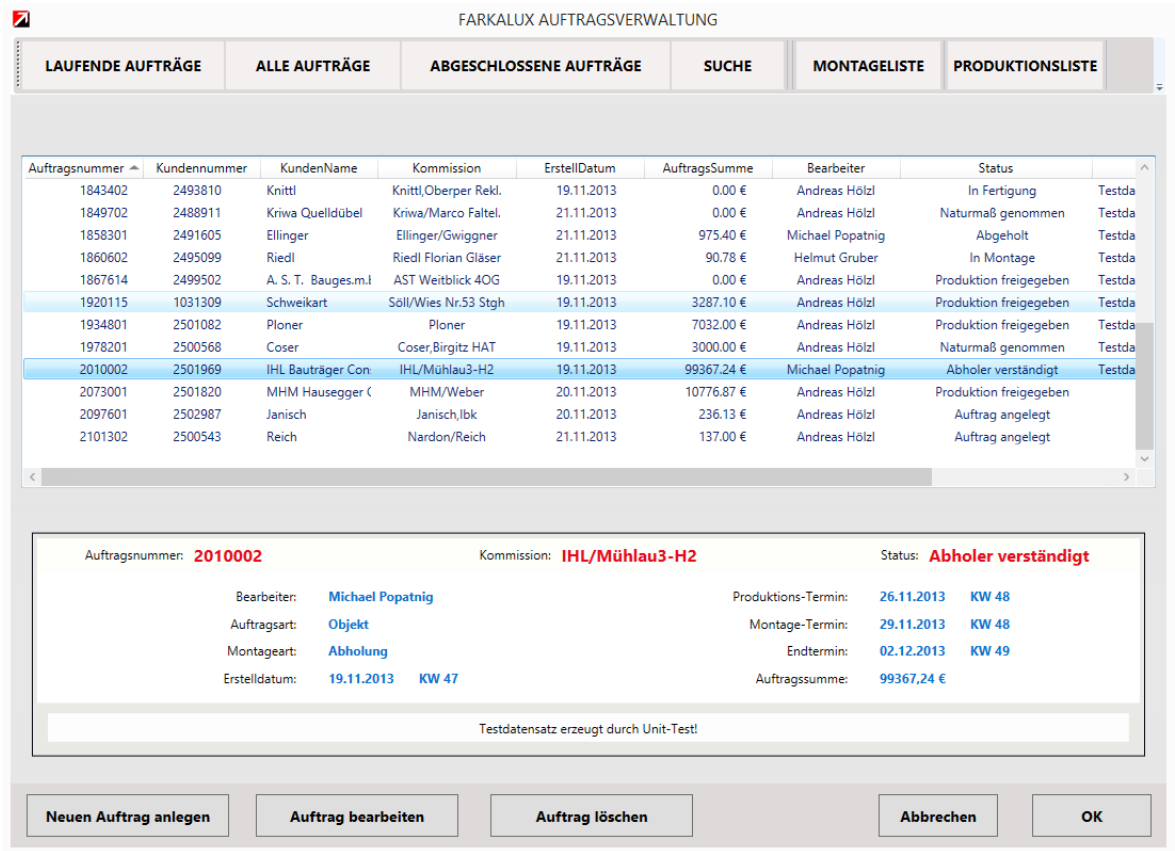


Abbildung 30: Hauptfenster der Auftragsverwaltung

7.2.3.1 Proxys verwenden

Die Dienste der Auftragsverwaltung werden vom Client als Proxy konsumiert. Dazu muss mit Visual Studio im Projekt des Clients ein Dienstverweis erstellt werden. Dabei wird die Option „Generieren Asynchroner Vorgänge zulassen“ angewählt, als Auflistungstyp wird die oben genannte ObservableCollection sowie die Option „Typen in Assemblys, auf die verwiesen wird, wiederverwenden“ gewählt. Durch die letzte Option werden die bekannten Klassen direkt eingebunden und nicht als Kopie neu erzeugt. Dies ermöglicht die Wiederverwendung der Validierung aus Abschnitt 7.2.2.5.

Da der Aufruf von Diensten auf entfernten Servern potentiell fehleranfällig ist, müssen diese Aufrufe in spezielle Fehlerbehandlungsroutinen eingebunden werden. Damit dies nicht bei jedem Aufruf speziell geregelt werden muss, wird eine generische, statische Klasse ‚Proxy‘ dafür verwendet. Der Aufruf der entsprechenden Methode wird in ein einziges try-catch-Konstrukt eingebunden, das Timeout- und Communication-Exceptions speziell abfängt.

7.2.3.2 Entwicklung des Hauptfensters

Die optische Ausführung des Hauptfensters ist stark an die in Abschnitt 6.2.5 erwähnte Pilot-Anwendung angelehnt (siehe Abbildung 30). Im oberen Teil sind auf der linken Seite die Steuerungselemente für die Listenanzeige untergebracht. Rechts befinden sich even-

tuelle Zusatzfunktionen, je nach Benutzerberechtigungen. Als zentrales Element in der Mitte steht die Auflistung der Aufträge entsprechend der gewählten Ansicht. Im unteren Bereich befindet sich einerseits die Detailanzeige des gerade ausgewählten Elements, andererseits die Funktionen dazu. Diese Elemente sind nur bei Anwahl eines Listenelements aktiv (siehe Abbildung 31).

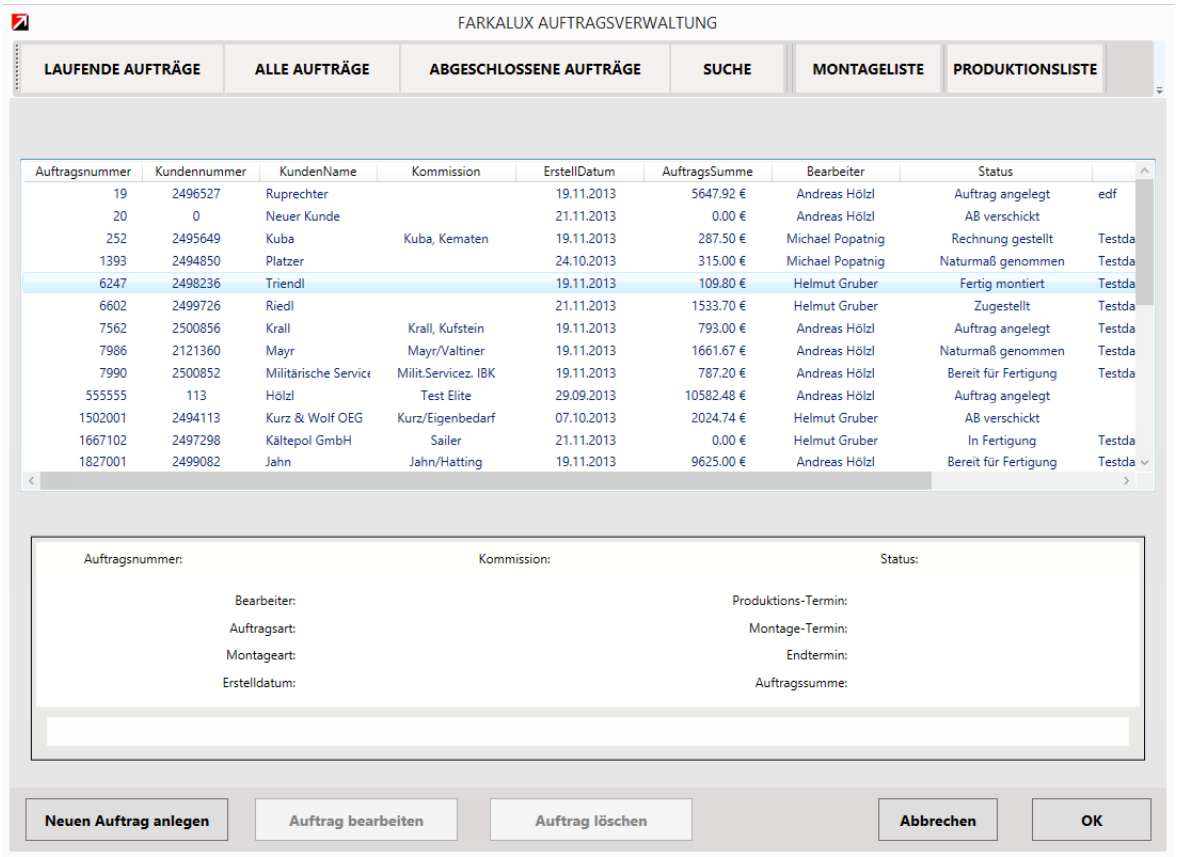


Abbildung 31: Hauptfenster mit inaktiven Funktionen

Beim Start der Anwendung wird sofort das Hauptfenster geladen. Befindet sich dabei der Dienst im Leerlauf, dauert der erste Dienstaufruf mindestens 8 sec. Deshalb wird beim Laden des Fensters der erste Aufruf der Methode ‚GetOrders' asynchron ausgeführt. Gleichzeitig wird ein Textlabel mit einem Wartetext angezeigt, bis der Inhalt der Liste geladen ist. Der Aufruf bei Umstellung der Anzeige erfolgt dann wieder synchron.

Zur Anzeige von kurzen Datumswerten in der Auflistung wird ein Date-Konverter benötigt. Dieser muss die Schnittstelle ‚IValueConverter' implementieren, welche die Methoden ‚Convert' und ‚ConvertBack vorschreibt', welche die eigentliche Umwandlung vornehmen.

In der Klasse ‚DateConverter' wird noch zusätzlich eine Methode eingefügt, die aus einem Datumswert einen String mit der Kurzform des Datums und der errechneten Kalenderwoche erzeugt.

7.2.3.3 Entwicklung des Fensters „Auftrag bearbeiten“

Das Fenster „Auftrag bearbeiten“ (siehe Abbildung 32) dient der Detailanzeige und Bearbeitung eines einzelnen Auftrags. Zudem wird es zur Anlage eines neuen Elements verwendet. Beim Laden des Fensters wird das aktuell ausgewählte oder ein leeres Element an den Datenkontext des Window gebunden, dadurch erben auch allen enthaltenen Controls diese Bindung. Einzig die ComboBox zur Auswahl eines freien Auftrages aus dem Fensterbausystem erhält ihren eigenen Datenkontext. Da die Erzeugung desselbigen, wie bereits im Abschnitt 7.2.2.6 angeführt, mehrere Sekunden dauert, wird auch dieser Methodenaufruf asynchron ausgeführt.

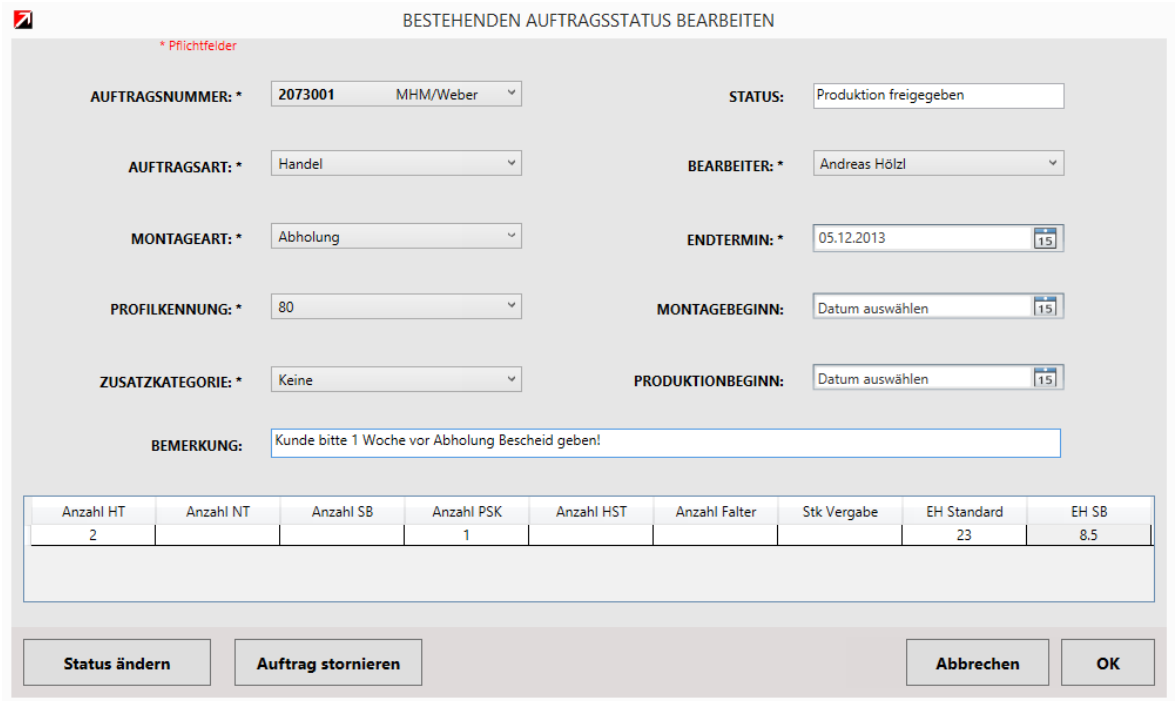


Abbildung 32: Fenster „Auftrag bearbeiten“

Abhängig vom jeweiligen Zweck (Neuanlage oder Änderung eines bestehenden Auftrags) erfolgen Anpassungen an der Oberfläche. Dies wird mittels einer booleschen Variablen ‚IstNeuerAuftrag‘ gesteuert. Damit diese in der Klasse und auch nach Abschluss der Eingaben beim Schließen des Fensters überprüft werden kann wird sie als öffentliche Eigenschaft angelegt. Im Fenster wird die Titelleiste angepasst und die Funktion „Auftrag stornieren“ ein- oder ausgeblendet.

Die Tests der Oberflächen werden in dieser Arbeit manuell mittels ausprobieren durchgeführt. Die Oberflächen beinhalten kaum komplexere Möglichkeiten, deshalb ist diese Form der Tests gerechtfertigt. Bei einem solchen Test zeigt sich, dass die vorhin erwähnte ComboBox beim Aufklappen mehrere Sekunden bis zur Anzeige der Liste benötigt.

```
<ComboBox Name="Cmbx_Auftrag" ItemTemplate="{StaticResource AduloAuftrag}" ... >
        <ComboBox.ItemsPanel>
            <ItemsPanelTemplate>
                <VirtualizingStackPanel />
            </ItemsPanelTemplate>
        </ComboBox.ItemsPanel>
</ComboBox>
```

Abbildung 33: Virtualisierung der ComboBox-Anzeige

Dies liegt darin begründet, dass in der Auflistung über 10000 Elemente vorliegen und diese zur besseren Übersicht als Data-Template angezeigt werden (siehe Abbildung 34). Damit das Rendern der Liste beschleunigt wird, kann die Option laut Abbildung 33 in XAML eingefügt werden. Diese ändert das Standardverhalten der ComboBox. Normalerweise wird dabei jedes Element einzeln generiert und dann für die Anzeige einzeln gerendert. Die Virtualisierung bewirkt, dass alle Inhalte gemeinsam erzeugt und auch in einem Durchlauf gerendert werden [msdnwpf2].

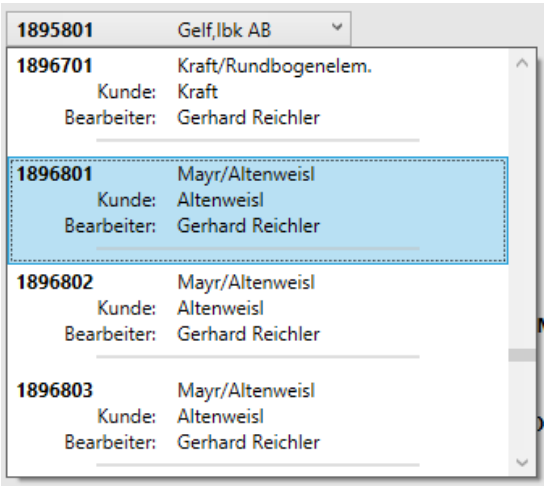


Abbildung 34: ComboBox mit DataTemplate

Damit die Validierung der Daten von den Controls übernommen wird, muss bei der Bindung der Zusatz ‚`ValidatesOnDataErrors=True`' angegeben werden. Damit kann die WPF automatisch Validierungsfehler beim jeweiligen Control anzeigen. Zusätzlich wird noch die Methode zum Speichern der Daten als Command ausgeführt und mit einem ‚CanExecute-Handler' ausgestattet. Dieser kann Überprüfungen durchführen, in unserem Fall ruft er die SelfValidation-Methode der Klasse auf. Liegen keine Fehler vor, werden das Command und der damit verbundene OK-Button freigeschalten. Somit kann der Benutzer den Button nur anwählen, wenn alle Eingaben valide sind. Ist dies nicht der Fall, wird zusätzlich ein rotes Ausrufezeichen angezeigt. Dieses beinhaltet im Tool-Tipp den Text der Validierungsfehler, wie in Abbildung 35 ersichtlich.

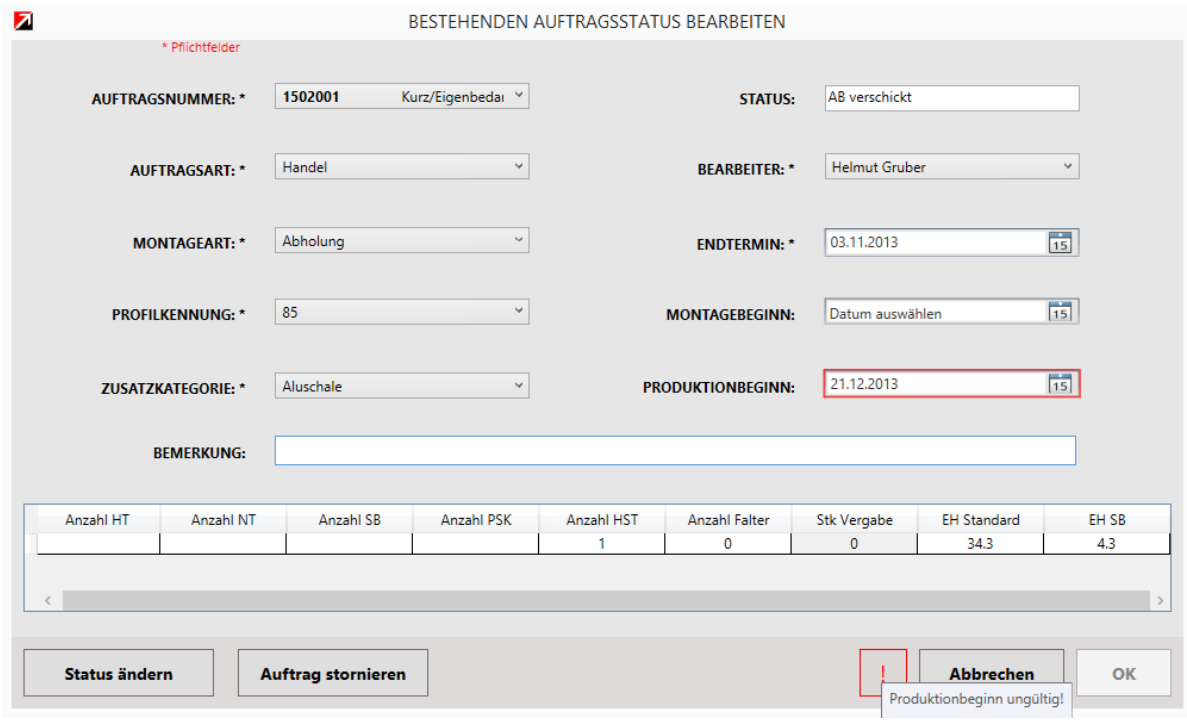


Abbildung 35: Anzeige von Validierungsfehlern

Das gleiche Prinzip wird auch beim Fenster zum Durchführen der Statusänderungen angewendet. Wie in Abbildung 36 ersichtlich, kann der Button „Status ändern“ noch nicht angewählt werden, weil kein gültiger neuer Status ausgewählt ist.

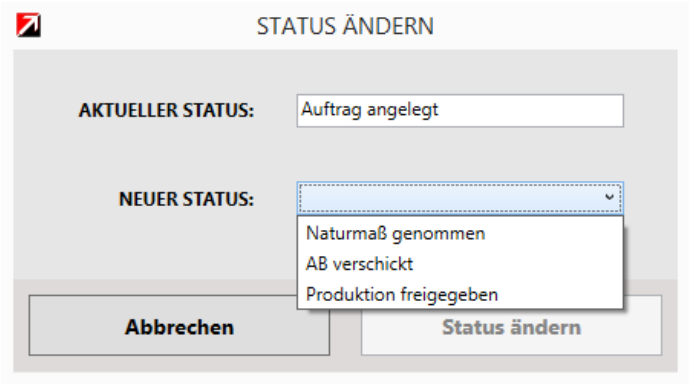


Abbildung 36: Fenster zur Statusänderung

Die Entwicklung des Informationssystems für die Auftragsverwaltung ist durch Erstellung der jeweiligen Bereiche im Daten-, Business- und Presentation-Layer abgeschlossen. Die Anwendung ist bereit für die Tests, die im nächsten Kapitel behandelt werden.

8 Test und Inbetriebnahme

Zum zwischenzeitlichen Feststellen von Teilergebnissen wurden während der Implementierung Unit-Tests für kleiner Abschnitte, wie zum Beispiel einzelne Funktionen oder Methoden, sowie Komponententests für die Dienste oder Layer verwendet. In diesem Abschnitt wird nun der Systemtest ausgeführt. Er dient zum Testen aller Komponenten gemeinsam und soll bestimmen, ob der umgesetzte Bereich bereit zur Inbetriebnahme wäre.

8.1 Einführung

Bei den Teiltests im vorherigen Kapitel war es immer wichtig, das Umfeld möglichst so einzuschränken, dass nur die zu testenden Methoden und Funktionen Einfluss auf das Ergebnis hatten. Dies kann unter Zuhilfenahme von sogenannten Mock-Objekten erfolgen, die echte Objekte oder Funktionen nachbilden. In dieser Arbeit wurde allerdings hauptsächlich eine andere Variante verwendet, nämlich die der Testdatensätze, die in der Entwicklungsdatenbank angelegt wurden. Der Zugriff auf die Testobjekte erfolgte dabei so direkt wie möglich. Dienste wurden lokal gehostet, Workflows lokal ausgeführt und der Daten-Layer wurde direkt in die Testapplikation eingebunden.

Parallel erfolgten Komponenten-Tests, welche die Funktionen am (teilweise) fertig gestellten System durchführten. Hier wurden die Testmethoden meist parallel zum entwickelten Programm geschrieben, dies schaffte bessere Voraussetzungen für das Debugging und bessere Probiermöglichkeit für den Programmierer.

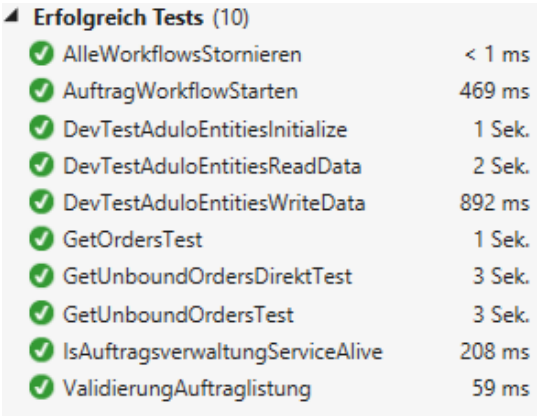


Abbildung 37: Testergebnisse aus automatisierten Teiltests

Mit Ausnahme der Oberflächentests handelte es sich hier stets um automatisierte Routinen, die relativ rasch durchgeführt werden konnten. Ein Auszug aus einer solchen Ergebnisliste ist in Abbildung 37 ersichtlich.

Der Systemtest in diesem Abschnitt soll nun die Software in seiner Gesamtheit erfassen, also alle einzelnen Komponenten und das Zusammenwirken aller Bestandteile unter Verwendung der in Abschnitt 7.1.4 angeführten Testdaten. Mit dem Systemtest soll bestimmt werden, ob die Software bereit zur Inbetriebnahme wäre.

Zu diesem Zweck wird das Informationssystem von zwei ausgewählten Key Usern auf alle vorgegebenen Funktionen hin getestet. Die Ergebnisse werden schriftlich festgehalten.

8.2 Testaufbau

Als Testclient dient ein Computer im Produktiveinsatz mit dem Betriebssystem Windows 7 Enterprise SP1 64 Bit und Installation des .NET-Frameworks 4.5 (Full Profile). Die Testpersonen sind jeweils mit ihren eigenen Benutzerkonten angemeldet.

Serverseitig verbleiben die Dienste auf dem Applikationsserver, allerdings werden sie im Application-Pool ‚Test' ausgeführt. Als Datenbank kommt die Development-Version laut Abschnitt 7.1.3 zur Verwendung, diese befindet sich ebenfalls schon auf dem Produktivsystem. Als Testkriterien dienen die funktionalen und nicht-funktionalen Anforderungen aus Kapitel 4. Nach einer kurzen Einführung in die Software spielen die Probanden Auftragsszenarien durch. Nach Absolvieren der Tests erfolgt eine kurze Dokumentation und ein Interview, in welchem die gemeinsamen Erfahrungen geteilt werden.

8.3 Ergebnisse

Die Befragung ergibt, dass das Hauptziel – die durchgängige Anwendung in allen Abteilungen – erreicht wurde. Das Informationssystem deckt die Bereiche Verkauf, Technik, Produktion, Lager, Montage und Verwaltung ab. Mittels der neuen Anwendung werden auch alle geforderten Prozessschritte von der Anlage eines Auftrags bis zur Abrechnung abgebildet.

Die funktionale Anforderung, dass nur die nächstmöglichen Schritte auswählbar sind, wird vom System ebenfalls erfüllt. In diesem Zuge wird allerdings bemängelt, dass das Setzen eines Status nicht mehr rückgängig gemacht werden kann. Wird so eine Änderung fälschlicherweise durchgeführt, kann der korrekte Zustand nur durch Löschen und Neuanlegen des Auftrags hergestellt werden.

Der Export der Aufträge für die Produktionsverwaltung funktioniert ebenso einwandfrei wie die Ausgabe der Aufträge für die Montageplanung.

Generell wird von den beiden Probanden die Software im Interview als sehr positiv beurteilt. Die Bedienung ist sehr intuitiv, Fehleingaben sind kaum möglich. Die Übersichtlichkeit der Anzeigen und Bedienelemente wird ebenfalls lobend erwähnt.

Als Wunsch wird eine zusätzliche Darstellung bei erfolgten Terminänderungen, zum Beispiel mittels eines Hinweistextes im oberen Bereich des Hauptfensters oder mit farblich abgesetzter Darstellung der Termine, geäußert.

Ein Fehler wird in den Systemtests aufgezeigt – eine Sicherheitsabfrage vor dem Löschen eines Auftrags ist nicht vorhanden. Somit könnte ein Auftrag unabsichtlich gelöscht werden. Dieser Fehler wird sofort behoben. Bei der Anwahl „Auftrag stornieren“ ist diese Sicherheitsabfrage vorhanden.

Die nichtfunktionalen Anforderungen werden durch den Entwickler überprüft. Die Erweiterungsfähigkeit ist dabei aufgrund der Auslegung als SOA und der Aufteilung in drei verschiede Layer gegeben. Die Erweiterung um eine Bestell-Lösung wurde im Konzept für das IS schon berücksichtigt. Die Ausdehnung auf andere Bereiche wie Sonnenschutz, Wintergarten oder Terrassenüberdachung kann aufgrund der Auslagerung der Fensteraufträge als eigenen Workflowservice ebenfalls leicht durchgeführt werden, indem zusätzliche Dienste für jeden Bereich angelegt werden.

Zusätzlich zum obigen Systemtest wird noch ein Belastungstest als Simulation von Mehrbenutzerzugriff durchgeführt. Hier werden automatisiert 100 Aufträge gleichzeitig angelegt. Dann werden die jeweiligen Status per Zufallsmodus verändert bis alle Aufträge abgeschlossen sind. Die Durchführung dauert ca. 30 Sekunden. Bei durchschnittlich 10 Statusänderungen pro Auftrag entspricht dies 33 Anfragen pro Sekunde. Diese werden ohne merkliche Auslastungsänderung vom Applikations-Server durchgeführt, wie in Abbildung 38 ersichtlich. Eine Steigerung auf 400 gleichzeitige Aufträge änderte das Ergebnis nicht.

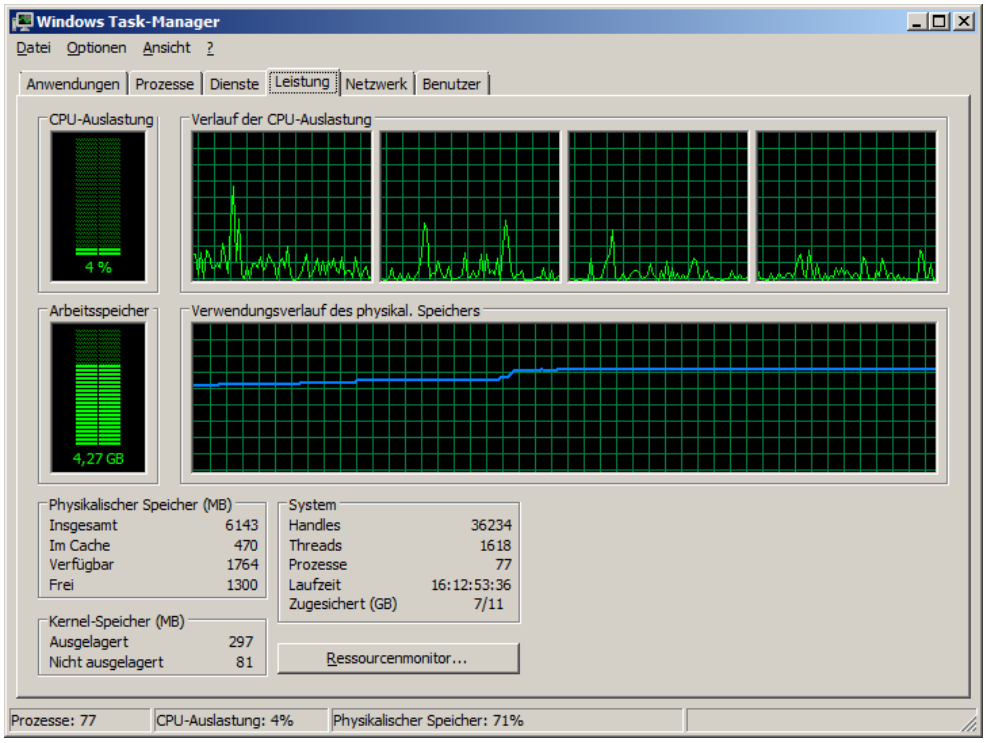


Abbildung 38: Auslastung am Applikationsserver

Die Autorisierung wird im Systemtest ebenso erfolgreich nachgewiesen. Die Funktionen der Montage bzw. der Produktion stehen in den anderen Bereichen nicht zur Verfügung. Ein Start der Anwendung ohne vorherige Anmeldung im Active Directory ist zwar möglich, allerdings können keine Daten abgerufen werden. Ein manuelles Aufrufen der Dienstmethoden in einem Web-Browser schlägt mit fehlenden Berechtigungen ebenso fehl.

Die Anforderungen bezüglich der Log-Möglichkeiten werden alle erfüllt. Die erzeugten Log-Dateien wurden dabei während der Programmierung ausführlich zum Debuggen benutzt.

Die Anforderung, dass Produktionsdaten nicht verändert werden, wird durch die Anbindung der Pervasive Daten als Sichten im EF-Modell Rechnung getragen. Eine versuchte Datenmanipulation mit dem SQL-Server-Management-Studio wird mit einer Fehlermeldung des Pervasive ODBC-Treibers abgebrochen. Zusätzlich wird der komplette Quellcode nach Operationen mit der Entität ‚AufDirAuszug' durchsucht, es werden dabei keine Zuweisungen gefunden.

9 Optimierung

Aufbauend auf den gesammelten Erfahrungen während der Entwicklung und der Tests aus dem vorhergehenden Abschnitt werden in diesem Kapitel Optimierungsmöglichkeiten aufgezeigt und teilweise auch umgesetzt.

9.1 Performanceoptimierungen

Die größte Schwachstelle bisher war die Performance der Lösung. Die Zugriffszeiten auf den PSQL-Server bereiteten in der Umsetzung bereits einige Probleme. Die Antwortzeiten sind in der aktuellen Ausführung zwar in einem akzeptablen Rahmen, könnten sich allerdings mit Steigerung der Auftragszahlen erhöhen. Eine Verschlechterung ist auf jeden Fall zu erwarten, falls für Funktionserweiterungen JOINS zwischen den SQL- und PSQL-Tabellen durchgeführt werden müssen.

Eine kleine Performance-Verbesserung kann erreicht werden, indem Abfragen, die nicht der Datenveränderung dienen, mit der Option „NoTracking“ durchgeführt werden. Beim Tracking werden Änderungen an den Objekten im Arbeitsspeicher verfolgt und an den Context gemeldet. Dieser kann dann bei Aufruf der Methode ‚SaveChanges‘ die Daten in der Datenbank entsprechend aktualisieren. In Abbildung 39 sind die Verhältnisse zwischen direktem SQL, Entity-Framework und Entity-Framework ohne Tracking ersichtlich. In unserer Anwendung bringt dies allerdings nur merkliche Vorteile bei Abfragen von Daten, die direkt am SQL-Server liegen.

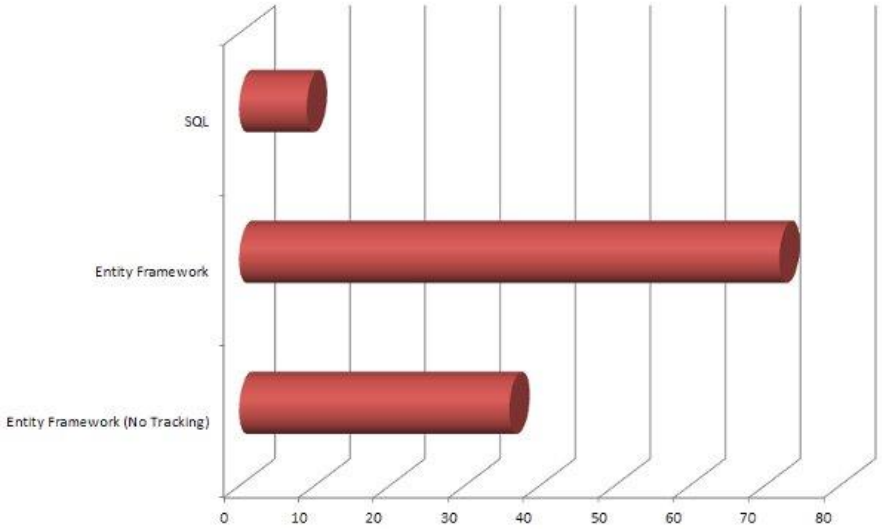


Abbildung 39: Performancevergleich Entity-Framework

[develop1]

Der Versuch, verbesserte Abfragezeiten durch Verwendung von vorgefertigten Sichten am SQL-Server zu erreichen, wird in der aktuellen Ausführung teilweise unternommen, dieses Thema könnte allerdings in manchen Bereichen (zum Beispiel am PSQL-Server direkt) noch etwas ausgereizt werden.

9.2 Anwendungsverbesserungen

In den Systemtests wurden bereits einige Verbesserungsvorschläge der Anwendung genannt, darunter die verbesserte Anzeige von Terminänderungen. Dies kann durch benutzerabhängiges Einblenden eines Buttons mit der Anzahl der geänderten Termine realisiert werden. Bei Klick auf den Button kann eine Liste der Terminänderungen jeweils mit Alt- und Neutermin angezeigt werden. Dazu müsste im Dienst Auftragsverwaltung eine entsprechende Funktion zur Ausgabe der Aufträge mit den geänderten Terminen erstellt werden. Die Tabellen der Datenbank benötigen dafür eine Erweiterung um zusätzliche Eigenschaften.

Der Wunsch bei der Auftragsabwicklung auch einen Schritt zurück machen zu können, könnte durch Änderung des Workflowservices umgesetzt werden. Hier stellt sich allerdings die Frage, ob dies bei allen Status sinnvoll ist. Das Auftragsverwaltungssystem soll die einzelnen Schritte der Abwicklung vorgeben, Rückschritte sind laut Anforderungsanalyse prinzipiell nicht vorgesehen. Im Fall von „Produktion freigegeben“ wäre diese allerdings denkbar.

Eine hilfreiche Verbesserung wäre das automatische Setzen von bestimmten Status. Dies könnte zum Beispiel bei der Rechnungslegung bzw. beim Rechnungsdruck durchgeführt werden, aber auch beim Druck der Lieferpapiere. Damit dies möglich wird, müsste ein entsprechendes Plug-In für das Programm Adulo erstellt werden. Dieses kann eine entsprechende Dienstmethode zum Verändern des Status aufrufen. Im Falle der Rechnungslegung könnten zusätzlich Informationen über den tatsächlichen Rechnungsbetrag übermittelt werden, die, von den entsprechenden Office-Reports, ausgewertet werden können.

10 Zusammenfassung

Im letzten Kapitel erfolgt zuerst ein kurzer Rückblick der gesamten Arbeit. Anschließend werden die wesentlichen Ergebnisse zusammengefasst und ein Ausblick auf eine eventuelle Weiterführung des Themas gegeben.

10.1 Zusammenfassung der Erstellung

Die Arbeit startete mit dem Ziel, ein Informationssystem bei Farkalux zu konzipieren und einen ausgewählten Bereich umzusetzen. Dazu mussten die bestehenden Systeme und Arbeitsweisen erhoben und analysiert werden. Diese Erfassung und Bewertung des Ist-Zustandes war mit erheblichem Aufwand verbunden, weil keine Dokumentationen oder Aufzeichnungen vorlagen. Der Vergleich der unterschiedlichen Lösungen wurde mit einer Nutzwertanalyse durchgeführt. Das Ergebnis ist eine komplette Übersicht aller Systeme bei Farkalux mit Hauptaugenmerk auf den entsprechenden Schwachstellen.

Im Anschluss wurden dann die notwendigen Grundlagen zur Anwendungsentwicklung geschaffen. Dabei ging es hauptsächlich um einen Erkenntnisgewinn in den Bereichen Datenverbindungen mit .NET-Anwendungen sowie Softwareentwicklung im Enterprise-Umfeld.

Danach wurden aufbauend auf den bisherigen Erkenntnissen die Anforderungen an das Konzept für das neue Informationssystem definiert. Außerdem wurde mit der Auftragsverwaltung der Fensteraufträge der Teilbereich, der zur Umsetzung gelangte, ausgewählt. Damit war die Grundlage zur Applikationsentwicklung geschaffen und diese Phase konnte gestartet werden.

Es wurden die bisher verwendeten Daten hinsichtlich Speicherort, Verwendung und Umsetzung im neuen System behandelt. Alle für den Prozess vorhandenen Daten mussten dabei in das neue System eingebunden oder übernommen werden. Mit der Entscheidung für den SQL-Server und der Verbindung zum bestehenden DBMS wurde die zukünftige, zentrale Datenhaltung bei Farkalux bestimmt.

Mit der Architekturerstellung und Softwaremodellierung wurde die genaue Ausführung der Anwendung definiert. Dazu erfolgte mehrfach eine gezielte Auswahl nach Kriterien. Mit diesen Festlegungen ist das Konzept vollständig. Gleichzeitig wurden die Rahmenbedingungen für eine erfolgreiche Programmierung geschaffen. Zusätzlich wurden entsprechende Testsysteme und Datenbanken eingerichtet.

Bei der eigentlichen Implementierung musste verstärkt Aufmerksamkeit auf die Verwendung der bestehenden Daten in der Fensterbau-Software in Bezug auf Geschwindigkeit und Kompatibilität gelegt werden. Die Programmierung der Lösung wurde nach Layern strukturiert, die Zwischenergebnisse dabei mit Unit-Tests sichergestellt.

Die abschließenden Tests bestätigen das Erreichen der gesetzten Ziele. Die Lösung wurde von den Testbenutzern gut angenommen und wäre bereit zur Inbetriebnahme.

Aus den Tests und Entwicklungserfahrungen wurden im vorletzten Kapitel noch einige Optimierungsmöglichkeiten abgeleitet und teilweise auch umgesetzt.

10.2 Ergebnisse

Die Kernfrage dieser Arbeit lautet:

Wie kann ein Informationssystem bei Farkalux unter Berücksichtigung des speziellen betrieblichen Umfelds erstellt werden?

Diese Frage konnte hauptsächlich in den Kapiteln 5 und 6 beantwortet werden. Der SQL-Server und die darin eingerichtete Verbindung zur Bestandsdatenbank schaffen einen zentraler Datenspeicher. Für den Zugriff darauf wird mit dem EF eine leistungsstarke und doch einfach zu verwendende Technologie ausgewählt. Der Datenzugriff durch die vorhandenen Reports in Excel erfolgt mittels ODBC, somit können teilweise bestehende Lösungen weiter verwendet werden. Das IS besitzt eine serviceorientierte Architektur. Dieser modulare Aufbau und die logische Unterteilung in drei Layer garantieren eine einfache Erweiterbarkeit, sowie die geforderte stufenweise Entwicklungs- und Einführungsmöglichkeit. Die Umsetzung dieser Architektur mittels der WCF und AppFabric als Host gehören dabei ebenso zum Konzept des IS wie die Aufteilung der Funktionen auf den Auftrags-, den Auftragsverwaltungs- und den Bestellverwaltungsdienst. Mit der Verwendung der Enterprise-Library für die Fehlerbehandlung, für die Datenvalidierung und für das Logging werden weitere offene Punkte bestimmt. Für die Authentifizierung und Autorisierung nutzt das IS den Dienst „Active Directory“. Die Ausführung des Clients als WPF-Anwendung und die Anzeige der unterschiedlichen Funktionen nach den jeweiligen Berechtigungen schließen das Konzept ab. Damit ist die eingangs erwähnte Kernfrage vollständig beantwortet.

Mit dem Konzept werden alle Ziele aus dem Abschnitt 4.1 erreicht, sowie alle Teilfragen beantwortet. Das IS ist trotz der Aufteilung in verschiedene Dienste nach außen hin eine einheitliche, durchgängige Anwendung. Die Bereiche der Auftragsabwicklung und des Angebotswesens werden in Zusammenarbeit mit der Software Adulo abgedeckt, die Bestellverwaltung wird ausschließlich im IS umgesetzt. Die Anwendung kann in allen Abteilungen verwendet werden, die rollenbasierten Berechtigungen steuern dabei die angezeigten Funktionen. Durch die Verwendung der zentralen, relationalen Datenbank können

die vorhandenen Systeme teilweise konsolidiert bzw. ersetzt werden. Eine einfache Erweiterbarkeit ist ebenso gegeben wie die geforderte stufenweise Umsetzbarkeit.

Durch die Umsetzung konnte das Konzept getestet und zusätzliche Erfahrungen gesammelt werden. Die Umsetzbarkeit und Funktionsfähigkeit können dabei vollständig bestätigt werden. Die verwendeten Technologien eignen sich allesamt für die gestellten Aufgaben. Als besonders positiv aufgefallen sind dabei der SQL-Server und die WF. Beim SQL-Server in der Express Variante sind die Unterstützung durch .NET, Visual Studio, AppFabric und den Office-Programmen sowie keine Lizenzkosten bis zu einer Datenbankgröße bis 10 GB die Highlights. Bei der WF können die einfache Modellierung von Geschäftsprozessen mit asynchroner Kommunikation und die Möglichkeit der Persistierung besonders begeistern. Für den laufenden Betrieb fehlen allerdings noch die Erfahrungen. Hier ist besonders interessant wie diese Technik auf nachträgliche Veränderungen, vor allem bei bereits laufenden Workflows, reagiert.

Die Hosting Plattform IIS in Verbindung mit AppFabric hat sich bis jetzt ebenfalls bewährt. Weitere Erfahrungen im Dauerbetrieb in den Punkten Stabilität und Health-Monitoring müssen dabei aber erst gemacht werden.

Die Anbindung der Daten per Linked-Server stellt den größten Schwachpunkt der Lösung dar. Diese Variante führt zwar alle Daten an einem Punkt zusammen, teilweise kann der Zugriff darauf jedoch zu lange dauern. In solchen Fällen müssten die Daten der Fensterbau-Software in eine Datenbank am SQL-Server repliziert werden. Auf welche Art und Weise dies am besten realisiert werden könnte, müsste in einem eigenen Projekt festgestellt werden.

Die Konsolidierung der Daten an einer Stelle ist eine wertvolle Grundlage für ein erweiterbares, zusammenhängendes Informationssystem, aber auch für das Unternehmen generell. Bei Farkalux war diese Art von Lösungen bisher nicht umsetzbar. Mit den Daten in einem zentralen, relationalen System und dem frei programmierbaren Zugriff darauf können einige große Probleme des Unternehmens beseitigt werden.

Der größte Nutzen für Farkalux entsteht durch die erreichte schnelle, zentrale Informationsbereitstellung des Systems für die jeweiligen Benutzer. Die Zahl an manuellen Rückfragen und vergessenen oder fehlgeleiteten Informationen sollte sich drastisch reduzieren, was wiederum zu einer Effizienzsteigerung und einer besser Planbarkeit führt. Diese Vorteile machen sich vor allem in der Produktions- und Montageplanung positiv bemerkbar.

Mit der Erhebung und Beurteilung des Bestandes wurde ebenfalls ein gefordertes Teilziel erreicht. Dabei wurden bereits interessante Ergebnisse ersichtlich. Die Erhebung liefert eine Dokumentation aller verwendeten Systeme. Dabei wird deutlich, wie heterogen und dezentral die verwendeten Lösungen sind. Die meisten Bestandsysteme sind von lokaler Bedeutung, Software für Mehrbenutzerzugriff ist kaum in Verwendung. Mit der Prioritätsbewertung werden die Lösungen nach deren Wichtigkeit für das Unternehmen gereiht. In

Verbindung mit der Schwachstellenanalyse wird klar, dass für Farkalux wichtige Informationssysteme teilweise gut funktionieren, aber auch welche Lösungen die gestellten Anforderungen schlecht erfüllen und nicht mehr am Stand der Technik sind.

Generell wurde bewiesen, dass mittels gezielter Analyse und darauf aufbauender Umsetzung eine Strukturierung mit einer einzelnen Anwendung, bei entsprechend technologischer Ausarbeitung, möglich ist. Dies lässt die Schlussfolgerung zu, dass weitere Bereiche umgestellt werden können. Dabei muss allerdings für jeden Teilbereich überprüft werden, ob eine vollwertige Einbindung wirtschaftlich sinnvoll realisiert werden kann. Der in dieser Arbeit betroffene Bereich stellt nämlich den Hauptanteil am Umsatz von Farkalux dar, in weniger starken Produktbereichen könnte eine vereinfachte Variante zielführender sein.

Mit der Umsetzung wurde auch klar, dass der Aufwand zum Planen und Umsetzen einer Anwendung dieser Art recht erheblich ist. Vor allem die Datenanbindung an den Bestand stellte eine große Herausforderung dar. Die Erstellung eines kompletten Systems, wie zum Beispiel ein eigenes ERP-System, scheint daher nicht zweckmäßig zu sein. Für den Fall, dass so ein System notwendig wird, ist es besser auf eine bestehende Applikation zurückzugreifen. Wenn auch die Anpassung des Systems an das Unternehmen und die teilweise Umorganisation der Firma sowie die Lizenzkosten einen beträchtlichen Aufwand darstellen, so ist dieser wahrscheinlich doch geringer als eine komplette Neuentwicklung.

Ein großer Vorteil dieser Arbeit ist, dass viele Grundsatzentscheidungen getroffen wurden. Bei Weiterentwicklungen kann darauf zurückgegriffen werden. Dadurch können diese viel effizienter durchgeführt werden. Die Erkenntnis, dass hier ein System geschaffen werden konnte, das in Etappen die vorhandenen Anforderungen von Farkalux erfüllen kann, ist wirtschaftlich sehr wichtig. Aufgrund der überschaubaren Firmengröße und der großen Produkt- und Abwicklungsvielfalt würde eine Komplettumstellung viele finanzielle und humane Ressourcen binden. Eine sehr lange Amortisierungsdauer wäre die Folge. Mit dem geplanten IS können die wichtigsten Änderungen für Farkalux kostengünstig umgesetzt werden, mittelfristig kann mit dieser Lösung gut gearbeitet werden. Eine positive Auswirkung auf das Betriebsergebnis sollte die Folge sein.

10.3 Ausblick

Vier gravierende weltweite Änderungen haben das betriebliche Umfeld stark verändert. Die erste Entwicklung ist die Entstehung und Stärkung der globalen Wirtschaft. Der zweite Punkt ist der Wandel von Industriegesellschaften hin zu wissens- und informationsbasierten Dienstleistungsgesellschaften. Der dritte Punkt betrifft die Änderungen der Unternehmen selbst, vor allem bezüglich der Unternehmensformen und der Organisation. Die vierte Entwicklung ist die Entstehung des vernetzten Unternehmens. Diese Veränderungen im betrieblichen Umfeld und im Geschäftsklima bringen für Organisationen und deren Geschäftsführung eine Reihe weiterer Herausforderungen mit sich [lauden 2010, S. 8].

Um diesen neuen Anforderungen zu begegnen, werden Informationssysteme eingesetzt. Was ein Unternehmen in fünf Jahren realisieren möchte, hängt meist davon ab, was seine Systeme heute leisten. Die Erhöhung der Marktanteile, Herstellung hochqualitativer oder kostengünstiger Produkte, Entwicklung neuer Produkte und steigende Produktivität der Mitarbeiter und der Produktionsprozesse hängen mehr oder weniger von der Art und Weise sowie der Qualität der eingesetzten Informationssysteme in der Organisation ab [lauden 2010, S. 12].

Farkalux verfügt nun über ein Konzept, wie ein solches System aussehen könnte. Durch die Teilumsetzung wurde das Konzept bezüglich Funktion und Umsetzbarkeit bestätigt. Nach Freigabe durch die Geschäftsführung kann eine Umsetzung des gesamten Informationssystems oder eine Einführung des Teilbereichs starten. Die Inbetriebnahme der Auftragsverwaltung hätte den Vorteil, dass zusätzliche Erfahrung im Umgang mit der zentralen Anwendung gesammelt werden kann. Diese kann bei der Entwicklung des restlichen Systems einfließen.

Zur Inbetriebnahme müssten folgende Schritte durchgeführt werden:

- Produktionsdatenbank erstellen (Schema aus vorh. Datenbank)
- Einbindung der PSQL-Datenbank per Linked-Server
- Anpassung von ‚AduloUnboundOrders' (nur Aufträge neueren Datums anzeigen)
- Anpassung der Verbindungszeichenfolge
- Verschieben der Dienste in die Produktionssite am IIS
- Produktions-Application-Pool zuordnen
- Logging anpassen (nicht mehr im Modus „Verbose")
- Ausführliche Fehlermeldungen gegen einfachere Texte tauschen
- Dienstreferenzen aktualisieren
- Client neu erzeugen und in Freigabe kopieren
- Registrierungsschlüssel für Anwendungsprotokoll verteilen
- Verknüpfung auf Client verteilen
- Offline schalten der Entwicklungsdatenbank und -dienste
- Funktionstests
- Entfernen der Projektliste
- Einrichten der Belegnummernvergabe in Adulo
- Einschulung der betroffenen Mitarbeiter
- Während der Anfangsphase verstärkte Systemüberwachung

Danach könnten die bestehenden Auswertungen angepasst werden, damit sie die zentralen Daten nutzen. Von dieser Änderung betroffen wären die Verkaufsauswertungen für ADM, Händler und Objekte. Wenn auch der Rest des Konzepts umgesetzt wird, würden erstmals bei Farkalux alle auftragsrelevanten Informationen an einem Punkt abrufbar sein. Der Informationsfluss wäre stark verbessert, ein effizienteres Durchführen der alltäglichen Aufgaben wäre möglich. Außerdem könnten die meisten bestehenden Lösungen in Excel

ersetzt werden und die zentrale Datenbasis würde um viele Informationen erweitert werden.

Die angeführte verbesserte Anbindung des Fensterbauprogramms mit Plug-Ins zum automatischen Setzen von bestimmten Statusinformationen wäre ebenfalls eine Thematik mit viel Effizienzpotential. Die genaue Art der Umsetzung könnte dabei in einem eigenen Projekt behandelt werden.

Eine Fortführung der Thematik im Zuge einer Umsetzung oder Weiterentwicklung ist meiner Meinung nach aus folgenden Gründen anzuraten:

1. Für einen Teil der bestehenden Systeme wurde die schlechte Eignung nachgewiesen. Diese sollen ersetzt werden.
2. Der Nutzen des Informationssystems für Farkalux wäre sehr hoch.
3. Der zur Umsetzung gelangte Bereich wäre bereit zur Inbetriebnahme. Diese könnte ohne großen Aufwand erfolgen.
4. Während der Ausführung dieser Arbeit konnte viel Erfahrung und Wissen in den relevanten Bereichen gesammelt werden.
5. Eine Umsetzung neuer Funktionen kann mit Anlehnung an die jetzige Vorgehensweise viel effizienter durchgeführt werden.
6. Die Benutzer des Systems können Erfahrung mit einer maßgeschneiderten Anwendung machen, dies fördert Ideen für zukünftige Verbesserungen.

Als Fazit kann man festhalten, dass mit dieser Arbeit ein Grundstein zur Verbesserung der Informationslandschaft bei Farkalux gelegt wurde. Der Prozess befindet sich erst am Anfang. Es gibt noch viel zu tun, aber der Weg zeigt eindeutig in eine positive Richtung.

Literatur

[adulo1] http://www.adulo.de/downloads/aduloversion.aspx, verfügbar am 28.8.2012, 12:14

[develop1] http://windowsdeveloper.de/Neuerungen-im-Entity-Framework-5.0, verfügbar am 26.11.2013, 18:13

[geirhos 2011] Geirhos, Matthias: Professionell entwickeln mit Visual C# 2010, Bonn, Galileo Press, 2011

[geirhos 2013] Geirhos, Matthias: Professionell entwickeln mit Visual C# 2012, Bonn, Galileo Press, 2013

[kansy 2013] Kansy, Thorsten: Datenbankprogrammierung mit .NET 4.5, München, Carl Hanser Verlag, 2013

[kühnel 2012] Kühnel, Andreas: Visual C# 2012 – Das umfassende Handbuch, Bonn, Galileo Computing, 2012

[lauden 2010] Lauden, Kenneth; Lauden, Jane; Schoder, Detlef: Wirtschaftsinformatik – Eine Einführung, München, Pearson Studium, 2010

[louis 2013] Louis, Dirk; Kansy, Thorsten; Strasser, Shinja: Microsoft Visual C# 2012 – Das Entwicklerbuch, Köln, O'Reilly Verlag, 2013

[lutz 2011] Lutz, Heinrich; Heinzl, Armin; Riedl, Rene: Wirtschaftsinformatik, Berlin/Heidelberg, Springer, 2011

[meier 2008] Meier, J.D.; Farre, Carlos; Taylor, Jason; Bansode, Prashant; Gregersen, Steve; Sundararajan, Madhu; Boucher, Rob: Improving Web Services Security, Redmont, Microsoft, 2008

[mertens 2012] Mertens, Peter; Bodendorf, Freimut; König, Wolfgang; Picot, Arnold; Schumann, Matthias; Hess, Thomas: Grundzüge der Wirtschaftsinformatik, Berlin/Heidelberg, Springer, 2012

[msdnwpf1] http://msdn.microsoft.com/de-de/library/aa970268(v=vs.110).aspx, verfügbar am 22.11.2013, 12:22

[msdnwpf2] http://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingstackpanel(v=vs.110).aspx, verfügbar am 22.11.2013, 15:20

[msdnwf1] http://msdn.microsoft.com/de-de/library/hh305677.aspx, verfügbar am 16.8.2013, 17:23

[msdnwf2] http://msdn.microsoft.com/en-us/library/ff729670(v=vs.110).aspx, verfügbar am 20.11.2013, 1:47

[msdnsql1] http://msdn.microsoft.com/query/dev10.query?appId=Dev10IDEF1&l=DE-DE&k=k(VDT.DESIGNERS.PROPERTIES.RELATIONSHIP)%3bk(VS.PROPERTIES)&rd=true, verfügbar am 3.10.2013, 21:50

[orghb 2012] Bundesverwaltungsamt (Hrsg.): Handbuch für Organisationsuntersuchungen und Personalbedarfsermittlung, BMI 2012 - Gesamtredaktion: Bundesverwaltungsamt, 2012

[panther 2012] Panther, Robert: Richtig einsteigen: Datenbank entwickeln mit SQL-Server 2012, Köln, O'Reilly, 2012

[pervasive1] http://www.pervasive.com/database/Home/Products/PSQLVx.aspx , verfügbar am 28.8.2013, 12:09

[pervasive2] Pervasive PSQL Programmer's Guide, Austin, Pervasive Software Inc., 2013

[pervasive3] http://www.pervasivedb.com/psqlv11/Documents/PSQLv11SP3_update4_readme_adonet.htm, verfügbar am 3.6.2013, 19:25

[pervasive4] Pervasive Advanced Operations Guide, Austin, Pervasive Software Inc., 2013

[pervasive5] SQL-Engine Reference, Austin, Pervasive Software Inc., 2013

[schulte 2001] Schulte, Gert: Material- und Logistikmanagement, München, Oldenbourg Wissenschaftsverlag, 2001

[scribner 2007] Scribner, Kenn: Microsoft Windows Workflow Foundation - Schritt für Schritt, Unterschleißheim, Microsoft Press, 2007

[steyer 2013] Steyer, Manfred; Schwichtenberg, Holger; Fischer, Matthias; Krause, Jörg: Verteilte Systeme und Services mit .NET 4.5, München, Hanser-Verlag, 2013

[wiki 2013] http://de.wikipedia.org/wiki/.NET, verfügbar am 17.8.2013, 17:09

A Anlagen

Anlagen 1 – Ergebnisse der Erhebung

Datensysteme in der Produktion:

Bezeichnung	Version	Art	Verwendung
Adulo	11.9h	Datenbank-anwendung	Produktionsausdrucke, Maschinenansteuerung, Erzeugung der Bauteiletiketten, Optimierung und Materialbedarfsauswertung
Gealan Bestellung		Excel	Bestellformular
Gealan Profilübersicht		Excel	Übersicht Kunststoffprofile und Bestellungen
Inventurliste Produktion		Excel	zur jährlichen Inventur
Mindestbestand Lagerprofile		Excel	zur wöchentlichen Mindestbestandsprüfung
diverse Bestelllisten Produktion		Excel	Bestellformular und teilweise Bestellübersicht
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung
Produktionsübersicht Fenster		Excel	Produktionssteuerung der Fensteraufträge, Schnittstelle zu Vertrieb, Technik und Geschäftsführung
Halbfertigen-Liste		Excel	Übersicht aller fertig produzierten Aufträge, Kontrolle der Faktura
Prämiensystem Produktion		Excel	Mitarbeiterbeurteilung, Prämienberechnung, Teamarbeit-Verwaltung
Zeitconsens	2.8.4	Datenbank-anwendung	Anwesenheitszeiterfassung
Innenfensterbankliste		Excel	Bestellverwaltung Innenfensterbänke
Außenfensterbankliste		Excel	Bestellverwaltung Außenfensterbänke
Glasbestellungen		Excel	Bestellverwaltung Gläser
Übersichtsliste Fensteraufträge		Excel	Auftragsübersicht
Wintergarten Wochen Planung		Excel	Produktionssteuerung Wintergärten

Tabelle 8: Datensysteme in der Produktion

Datensysteme im Lager:

Bezeichnung	Version	Art	Verwendung
Montageplan		Excel	Übersicht der zu kommissionierenden Aufträge
Bestellübersicht Alutüren		Excel	Übersicht Bestellungen Alutüren
Inventurliste Lager		Excel	zur jährlichen Inventur
Montageartikelbeschaffung		Excel	Übersicht sämtlicher Montageartikel
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung
Produktionsübersicht Fenster		Excel	Produktionssteuerung der Fensteraufträge, Schnittstelle zu Vertrieb, Technik und Geschäftsführung
Außenfensterbankliste		Excel	Bestellverwaltung Außenfensterbänke
Glasbestellungen		Excel	Bestellverwaltung Gläser

Tabelle 9: Datensysteme im Lager

Datensysteme in der Montage:

Bezeichnung	Version	Art	Verwendung
Montageplan		Excel	Montagesteuerung
Montagestunden-erfassung		Excel und VBA	Zuteilung Stunden zu Aufträgen
Montage Soll-Ist Auswertung		Excel	Soll-Ist Vergleich
Montage unbezahlte Stunden		Excel	Erfassung der unbezahlten Arbeiten
Prämiensystem Montage		Excel	Mitarbeiterbeurteilung, Prämienberechnung, Teamarbeit-Verwaltung
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung
Sonnenschutzmontage		Exchange öffentl. Kalender	Montagesteuerung
Produktionsübersicht Fenster		Excel	Produktionssteuerung der Fensteraufträge, Schnittstelle zu Vertrieb, Technik und Geschäftsführung
Außenfensterbankliste		Excel	Bestellverwaltung Außenfensterbänke
Glasbestellungen		Excel	Bestellverwaltung Gläser

Tabelle 10: Datensysteme Montage

Datensysteme in der Verwaltung:

Bezeichnung	Version	Art	Verwendung
Angebote		Word	Angebote für Sonnenschutz, Wintergärten und Terrassenüberdachungen
Adressensammlung Serienbriefe		Access	Adressen für Serienbriefe
Telefonliste		Excel	Telefonnummern Mitarbeiter und Subfirmen
Angebotssummen		Excel	Gesamtsumme Angebote pro Mitarbeiter
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung office@farkalux.at als öffentl. Ordner
Adulo	11.9h	Datenbank-anwendung	Auftragsfaktura
Halbfertigen-Liste		Excel	Kontrolle der Faktura
Finanzbuchhaltung Orlando	2012.02	Datenbank-anwendung	Finanzbuchhaltung, Anlagenbuchhaltung, Kostenrechnung
Deckungsbeitrags-auswertung		Excel	Übersicht zur Kostenrechnung
Elba	5.3.3.0	Anwendung	Schnittstelle zu Banksystem
CPU Lohn	11.11.3022	Datenbank-anwendung	Lohnverrechnung
Zeitconsens	2.8.4	Datenbank-anwendung	Anwesenheitszeiterfassung
Elda	4.0.0	Anwendung	Schnittstelle zu Sozialversicherungsträger

Tabelle 11: Datensysteme Verwaltung und Buchhaltung

Datensysteme in der Technik:

Bezeichnung	Version	Art	Verwendung
Zeiterfassung		Excel	Aufteilung der Arbeitszeit nach Bereichen
Übersichtsliste Fensteraufträge		Excel	Auftragsnummernvergabe, Auftrags-übersicht, Daten für Verkaufsauswertungen
Adulo	11.9h	Datenbank-anwendung	Auftragsanlage und Positionsdefinition, Kalkulation, Erzeugen von Auftrags-bestätigungen und Lieferscheinen
Innenfensterbankliste		Excel	Bestellverwaltung Innenfensterbänke
Außenfensterbankliste		Excel	Bestellverwaltung Außenfensterbänke
Glasbestellungen		Excel	Bestellverwaltung Gläser
Glaspreislisten		Excel	Einkaufspreisvergleich versch. Hersteller
Bestellübersicht Alutüren		Excel und VBA	Übersicht Bestellungen Alutüren
ADM Verkaufsauswertung		Excel	Verkaufsauswertung Direktverkauf
sonstige Verkaufsauswertung		Excel	Verkaufsauswertung für Handel und Objekt
Projektordner		Dateifreigabe	Speicherung von auftragsrelevanten Unterlagen zu Großobjekten
Wintergarten Wochen Planung		Excel	Produktionssteuerung Wintergärten
Wintergärten Projektordner		Dateifreigabe	Speicherung von auftragsrelevanten Unterlagen zu Wintergartenaufträgen
Überdachungen Projektordner		Dateifreigabe	Speicherung von auftragsrelevanten Unterlagen zu Überdachungsaufträgen
Übersichtsliste Überdachungen		Excel	Übersicht aller Überdachungsaufträge
Montageplan		Excel	Information zu geplanten Montagen
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung
Sonnenschutzmontage		Exchange öffentl. Kalender	Information zu geplanten Montagen
Produktionsübersicht Fenster		Excel	Information zu Fensteraufträgen in der Produktion

Tabelle 12: Datensysteme Technik

Datensysteme im Vertrieb:

Bezeichnung	Version	Art	Verwendung
Adulo	11.9h	Datenbank-anwendung	Angebotswesen Fenster
Angebote		Word	Angebote für Sonnenschutz, Wintergärten und Terrassenüberdachungen
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung Weiterleitung von Anfragen, Terminplanung
Überdachungen-Kalkulation		Excel	Kalkulation von Überdachungen
Wintergarten-kalkulation		Excel	Kalkulation von Wintergärten
Visus	5.6	Anwendung	Visualisierung von Wintergärten u. Überdachungen
Ausschreibungsliste		Excel	Übersicht Objektverkauf
ADM Verkaufsauswertung		Excel	Verkaufsauswertung Direktverkauf
sonstige Verkaufsauswertung		Excel	Verkaufsauswertung für Handel und Objekt

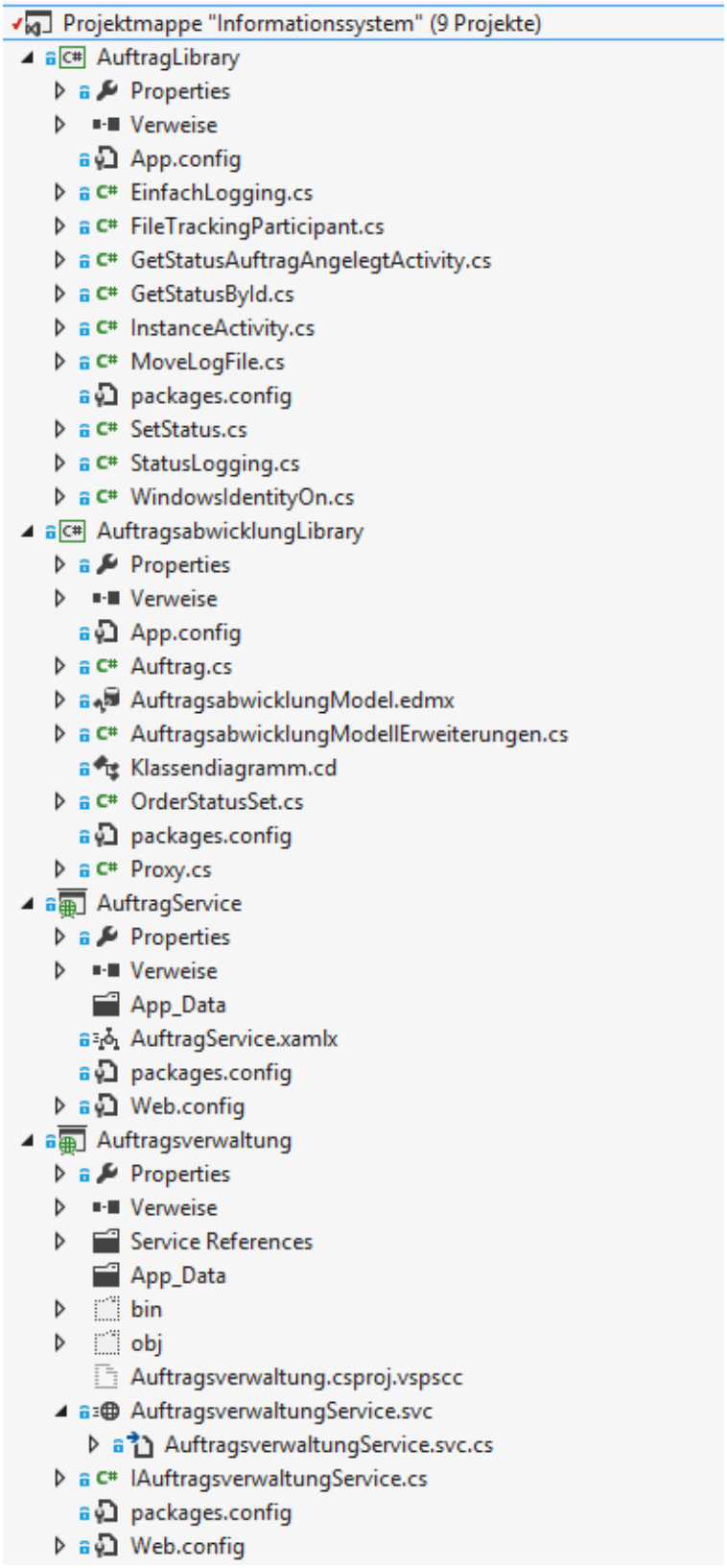
Tabelle 13: Datensysteme Vertrieb

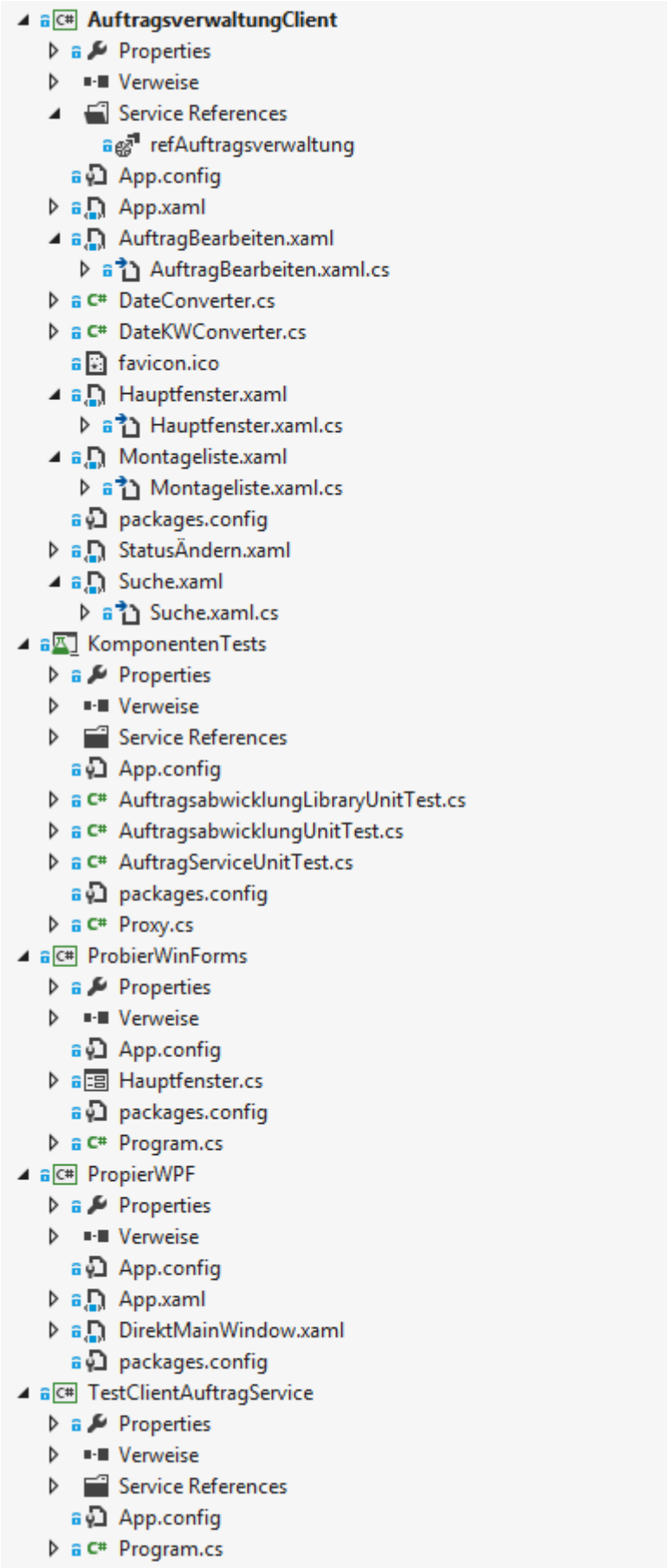
Datensysteme in der Geschäftsführung:

Bezeichnung	Version	Art	Verwendung
Adulo	11.9h	Datenbank-anwendung	Faktura Objekte
Produktionsübersicht Fenster		Excel	Übersicht Auslastung Produktion
Outlook	2010 SP 1	Client-anwendung	Mail-, Kalender- und Adressverwaltung
KVP		Excel	Verwaltung des kontinuierlichen Verbesserungsprozesses
Provisionsabrechnung		Excel	Provisionsermittlung Direktverkäufer
Zeiterfassung		Excel	Kontrolle der Techniker
Ausschreibungsliste		Excel	Übersicht Objektverkauf
ADM Verkaufsauswertung		Excel	Verkaufsauswertung Direktverkauf
sonstige Verkaufsauswertung		Excel	Verkaufsauswertung für Handel und Objekt

Tabelle 14: Datensysteme Geschäftsführung

Anlagen 2 – Übersicht der Visual Studio Solution




AuftragsverwaltungClient
Properties
Verweise
Service References
refAuftragsverwaltung
App.config
App.xaml
AuftragBearbeiten.xaml
AuftragBearbeiten.xaml.cs
DateConverter.cs
DateKWConverter.cs
favicon.ico
Hauptfenster.xaml
Hauptfenster.xaml.cs
Montageliste.xaml
Montageliste.xaml.cs
packages.config
StatusÄndern.xaml
Suche.xaml
Suche.xaml.cs
KomponentenTests
Properties
Verweise
Service References
App.config
AuftragsabwicklungLibraryUnitTest.cs
AuftragsabwicklungUnitTest.cs
AuftragServiceUnitTest.cs
packages.config
Proxy.cs
ProbierWinForms
Properties
Verweise
App.config
Hauptfenster.cs
packages.config
Program.cs
PropierWPF
Properties
Verweise
App.config
App.xaml
DirektMainWindow.xaml
packages.config
TestClientAuftragService
Properties
Verweise
Service References
App.config
Program.cs

Anlagen 3 – Inhalte der CD-ROM

Auf der beiliegenden CD-ROM befinden sich die digitale Ausgabe dieser Arbeit und die Visual-Studio-Solution der Umsetzung. Anbei erfolgt eine Auflistung der enthaltenen Ordner:

E:\Diplomarbeit
E:\Informationssystem
E:\Informationssystem\AuftragLibrary
E:\Informationssystem\AuftragsabwicklungLibrary
E:\Informationssystem\AuftragService
E:\Informationssystem\Auftragsverwaltung
E:\Informationssystem\AuftragsverwaltungClient
E:\Informationssystem\Bestellservice
E:\Informationssystem\KomponentenTests
E:\Informationssystem\packages
E:\Informationssystem\PropierWPF
E:\Informationssystem\TestClientAuftragService
E:\Informationssystem\TestWinForms
E:\Informationssystem\AuftragLibrary\bin
E:\Informationssystem\AuftragLibrary\obj
E:\Informationssystem\AuftragLibrary\Properties
E:\Informationssystem\AuftragLibrary\bin\Debug
E:\Informationssystem\AuftragLibrary\bin\Release
E:\Informationssystem\AuftragLibrary\bin\Debug\de
E:\Informationssystem\AuftragLibrary\obj\Debug
E:\Informationssystem\AuftragLibrary\obj\Release
E:\Informationssystem\AuftragLibrary\obj\Debug\TempPE
E:\Informationssystem\AuftragLibrary\obj\Release\TempPE
E:\Informationssystem\AuftragService\App_Data
E:\Informationssystem\AuftragService\bin
E:\Informationssystem\AuftragService\obj
E:\Informationssystem\AuftragService\Properties
E:\Informationssystem\AuftragService\Properties\PublishProfiles
E:\Informationssystem\AuftragService\bin\de
E:\Informationssystem\AuftragService\obj\Debug
E:\Informationssystem\AuftragService\obj\Release
E:\Informationssystem\AuftragService\obj\Debug\Package
E:\Informationssystem\AuftragService\obj\Debug\ProfileTransformWebConfig
E:\Informationssystem\AuftragService\obj\Debug\TempPE
E:\Informationssystem\AuftragService\obj\Debug\TransformWebConfig
E:\Informationssystem\AuftragService\obj\Debug\Package\PackageTmp
E:\Informationssystem\AuftragService\obj\Debug\Package\PackageTmp\bin
E:\Informationssystem\AuftragService\obj\Debug\Package\PackageTmp\bin\de
E:\Informationssystem\AuftragService\obj\Debug\ProfileTransformWebConfig\transformed
E:\Informationssystem\AuftragService\obj\Debug\TransformWebConfig\assist
E:\Informationssystem\AuftragService\obj\Debug\TransformWebConfig\original

E:\Informationssystem\AuftragService\obj\Debug\TransformWebConfig\transformed
E:\Informationssystem\AuftragService\obj\Release\TempPE
E:\Informationssystem\AuftragsabwicklungLibrary\bin
E:\Informationssystem\AuftragsabwicklungLibrary\obj
E:\Informationssystem\AuftragsabwicklungLibrary\Properties
E:\Informationssystem\AuftragsabwicklungLibrary\bin\Debug
E:\Informationssystem\AuftragsabwicklungLibrary\bin\Release
E:\Informationssystem\AuftragsabwicklungLibrary\bin\Debug\de
E:\Informationssystem\AuftragsabwicklungLibrary\bin\Release\de
E:\Informationssystem\AuftragsabwicklungLibrary\obj\Debug
E:\Informationssystem\AuftragsabwicklungLibrary\obj\Release
E:\Informationssystem\AuftragsabwicklungLibrary\obj\Debug\edmxResourcesToEmbed
E:\Informationssystem\AuftragsabwicklungLibrary\obj\Debug\TempPE
E:\Informationssystem\AuftragsabwicklungLibrary\obj\Release\edmxResourcesToEmbed
E:\Informationssystem\AuftragsabwicklungLibrary\obj\Release\TempPE
E:\Informationssystem\Auftragsverwaltung\App_Data
E:\Informationssystem\Auftragsverwaltung\bin
E:\Informationssystem\Auftragsverwaltung\obj
E:\Informationssystem\Auftragsverwaltung\Properties
E:\Informationssystem\Auftragsverwaltung\Service References
E:\Informationssystem\Auftragsverwaltung\Properties\PublishProfiles
E:\Informationssystem\Auftragsverwaltung\Service References\refAuftragService
E:\Informationssystem\Auftragsverwaltung\bin\de
E:\Informationssystem\Auftragsverwaltung\obj\Debug
E:\Informationssystem\Auftragsverwaltung\obj\Release
E:\Informationssystem\Auftragsverwaltung\obj\Debug__testMsDeployConnection__
E:\Informationssystem\Auftragsverwaltung\obj\Debug\Package
E:\Informationssystem\Auftragsverwaltung\obj\Debug\ProfileTransformWebConfig
E:\Informationssystem\Auftragsverwaltung\obj\Debug\TempPE
E:\Informationssystem\Auftragsverwaltung\obj\Debug\TransformWebConfig
E:\Informationssystem\Auftragsverwaltung\obj\Debug\Package\PackageTmp
E:\Informationssystem\Auftragsverwaltung\obj\Debug\Package\PackageTmp\bin
E:\Informationssystem\Auftragsverwaltung\obj\Debug\Package\PackageTmp\Service References
E:\Informationssystem\Auftragsverwaltung\obj\Debug\Package\PackageTmp\Service References\r
E:\Informationssystem\Auftragsverwaltung\obj\Debug\Package\PackageTmp\bin\de
E:\Informationssystem\Auftragsverwaltung\obj\Debug\ProfileTransformWebConfig\transformed
E:\Informationssystem\Auftragsverwaltung\obj\Debug\TransformWebConfig\assist
E:\Informationssystem\Auftragsverwaltung\obj\Debug\TransformWebConfig\original
E:\Informationssystem\Auftragsverwaltung\obj\Debug\TransformWebConfig\transformed
E:\Informationssystem\Auftragsverwaltung\obj\Release\Package
E:\Informationssystem\Auftragsverwaltung\obj\Release\TempPE
E:\Informationssystem\AuftragsverwaltungClient\bin
E:\Informationssystem\AuftragsverwaltungClient\obj
E:\Informationssystem\AuftragsverwaltungClient\Properties
E:\Informationssystem\AuftragsverwaltungClient\Service References
E:\Informationssystem\AuftragsverwaltungClient\Properties\DataSources
E:\Informationssystem\AuftragsverwaltungClient\Service References\refAuftragsverwaltung
E:\Informationssystem\AuftragsverwaltungClient\bin\Debug
E:\Informationssystem\AuftragsverwaltungClient\bin\Release
E:\Informationssystem\AuftragsverwaltungClient\bin\Debug\de
E:\Informationssystem\AuftragsverwaltungClient\bin\Release\de

E:\Informationssystem\AuftragsverwaltungClient\obj\Debug
E:\Informationssystem\AuftragsverwaltungClient\obj\Release
E:\Informationssystem\AuftragsverwaltungClient\obj\Debug\TempPE
E:\Informationssystem\AuftragsverwaltungClient\obj\Release\TempPE
E:\Informationssystem\Bestellservice\App_Data
E:\Informationssystem\Bestellservice\bin
E:\Informationssystem\Bestellservice\obj
E:\Informationssystem\Bestellservice\Properties
E:\Informationssystem\Bestellservice\bin\de
E:\Informationssystem\Bestellservice\obj\Debug
E:\Informationssystem\Bestellservice\obj\Release
E:\Informationssystem\Bestellservice\obj\Debug\TempPE
E:\Informationssystem\Bestellservice\obj\Release\TempPE
E:\Informationssystem\KomponentenTests\bin
E:\Informationssystem\KomponentenTests\obj
E:\Informationssystem\KomponentenTests\Properties
E:\Informationssystem\KomponentenTests\Service References
E:\Informationssystem\KomponentenTests\Properties\DataSources
E:\Informationssystem\KomponentenTests\Service References\refAuftragService
E:\Informationssystem\KomponentenTests\Service References\refAuftragsverwaltungService
E:\Informationssystem\KomponentenTests\bin\Debug
E:\Informationssystem\KomponentenTests\bin\Release
E:\Informationssystem\KomponentenTests\bin\Debug\de
E:\Informationssystem\KomponentenTests\bin\Release\de
E:\Informationssystem\KomponentenTests\obj\Debug
E:\Informationssystem\KomponentenTests\obj\Release
E:\Informationssystem\KomponentenTests\obj\Debug\TempPE
E:\Informationssystem\KomponentenTests\obj\Release\TempPE
E:\Informationssystem\PropierWPF\bin
E:\Informationssystem\PropierWPF\obj
E:\Informationssystem\PropierWPF\Properties
E:\Informationssystem\PropierWPF\bin\Debug
E:\Informationssystem\PropierWPF\bin\Release
E:\Informationssystem\PropierWPF\bin\Debug\de
E:\Informationssystem\PropierWPF\bin\Release\de
E:\Informationssystem\PropierWPF\obj\Debug
E:\Informationssystem\PropierWPF\obj\Release
E:\Informationssystem\PropierWPF\obj\Debug\TempPE
E:\Informationssystem\PropierWPF\obj\Release\TempPE
E:\Informationssystem\TestClientAuftragService\bin
E:\Informationssystem\TestClientAuftragService\obj
E:\Informationssystem\TestClientAuftragService\Properties
E:\Informationssystem\TestClientAuftragService\Service References
E:\Informationssystem\TestClientAuftragService\Service References\refAuftragServiceStartAuftrag
E:\Informationssystem\TestClientAuftragService\bin\Debug
E:\Informationssystem\TestClientAuftragService\bin\Release
E:\Informationssystem\TestClientAuftragService\bin\Debug\de
E:\Informationssystem\TestClientAuftragService\obj\Debug
E:\Informationssystem\TestClientAuftragService\obj\Debug\TempPE
E:\Informationssystem\TestWinForms\bin
E:\Informationssystem\TestWinForms\obj

E:\Informationssystem\TestWinForms\Properties
E:\Informationssystem\TestWinForms\bin\Debug
E:\Informationssystem\TestWinForms\bin\Release
E:\Informationssystem\TestWinForms\bin\Debug\de
E:\Informationssystem\TestWinForms\bin\Release\de
E:\Informationssystem\TestWinForms\obj\Debug
E:\Informationssystem\TestWinForms\obj\Release
E:\Informationssystem\TestWinForms\obj\Debug\TempPE
E:\Informationssystem\TestWinForms\obj\Release\TempPE
E:\Informationssystem\packages\EnterpriseLibrary.Common.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.Data.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.Logging.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.WCF.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.Logging.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.Logging.Database.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.Validation.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WCF.6.0.1304.0
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WPF.6.0.1304.0
E:\Informationssystem\packages\EntityFramework.5.0.0
E:\Informationssystem\packages\EntityFramework.6.0.1
E:\Informationssystem\packages\EntityFramework.de.6.0.1
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9
E:\Informationssystem\packages\Unity.3.0.1304.1
E:\Informationssystem\packages\EnterpriseLibrary.Common.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Common.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.Common.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.Data.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Data.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.Data.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.Logging.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.Logging.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.Logging.6.0.1304.0\lib\NET4
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.WCF.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.WCF.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.ExceptionHandling.WCF.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.Logging.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Logging.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.Logging.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.Logging.Database.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Logging.Database.6.0.1304.0\scripts
E:\Informationssystem\packages\EnterpriseLibrary.Logging.Database.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.Logging.Database.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.Validation.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Validation.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.Validation.6.0.1304.0\lib\NET45
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WCF.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WCF.6.0.1304.0\tools

E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WCF.6.0.1304.0\lib\NET4
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WPF.6.0.1304.0\lib
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WPF.6.0.1304.0\tools
E:\Informationssystem\packages\EnterpriseLibrary.Validation.Integration.WPF.6.0.1304.0\lib\NET4
E:\Informationssystem\packages\EntityFramework.5.0.0\Content
E:\Informationssystem\packages\EntityFramework.5.0.0\lib
E:\Informationssystem\packages\EntityFramework.5.0.0\tools
E:\Informationssystem\packages\EntityFramework.5.0.0\lib\net40
E:\Informationssystem\packages\EntityFramework.5.0.0\lib\net45
E:\Informationssystem\packages\EntityFramework.6.0.1\content
E:\Informationssystem\packages\EntityFramework.6.0.1\lib
E:\Informationssystem\packages\EntityFramework.6.0.1\tools
E:\Informationssystem\packages\EntityFramework.6.0.1\lib\net40
E:\Informationssystem\packages\EntityFramework.6.0.1\lib\net45
E:\Informationssystem\packages\EntityFramework.6.0.1\lib\net40\de
E:\Informationssystem\packages\EntityFramework.6.0.1\lib\net45\de
E:\Informationssystem\packages\EntityFramework.de.6.0.1\lib
E:\Informationssystem\packages\EntityFramework.de.6.0.1\lib\net40
E:\Informationssystem\packages\EntityFramework.de.6.0.1\lib\net45
E:\Informationssystem\packages\EntityFramework.de.6.0.1\lib\net40\de
E:\Informationssystem\packages\EntityFramework.de.6.0.1\lib\net45\de
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9\help
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9\lib
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9\tools
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9\lib\Net40
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9\lib\Net401
E:\Informationssystem\packages\Microsoft.Activities.Extensions.2.0.6.9\lib\Net45
E:\Informationssystem\packages\Unity.3.0.1304.1\lib
E:\Informationssystem\packages\Unity.3.0.1304.1\tools
E:\Informationssystem\packages\Unity.3.0.1304.1\lib\Net45
E:\Informationssystem\packages\Unity.3.0.1304.1\lib\NetCore45
E:\Informationssystem\packages\Unity.3.0.1304.1\lib\wp8

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Ellbögen, den 31.12.2013

Andreas Hölzl